NetDiff: Deep Graph Denoising Diffusion for Ad Hoc Network Topology Generation

Félix Marcoccia[1, 2*], Gilles Monzat de Saint Julien[2], Cédric Adjih[3], Paul Mühlethaler[1]

[1]Inria Paris
[2]Thales SIX
[3]Inria Saclay

Abstract

This paper introduces NetDiff, an expressive graph denoising diffusion probabilistic architecture that generates high-performance link topologies for wireless ad hoc networks. Such networks, with directional antennas, can achieve unmatched throughput and scalability when the communication links are designed to provide good geometric properties, notably by reducing interference between these links while respecting diverse physical constraints. How to craft such a link assignment algorithm remains a real problem. Deep graph generation offers multiple advantages compared to traditional approaches: it allows to relieve the network nodes of the communication burden caused by the search of viable links and to avoid resorting to heavy combinatorial methods to find a good link topology. Given that graph neural networks sometimes tend to struggle with global, structural properties, we augment the popular graph transformer with cross-attentive modulation tokens in order to improve global control over the predicted topology. We also incorporate simple node and edge features, as well as additional loss terms, to facilitate the compliance with the network topology physical constraints. A network evolution algorithm based on partial diffusion is proposed to maintain the network topology over time when the nodes are moving. Our results show that the generated topologies are realistic, require only minor correction steps to be operational, and establish NetDiff as a viable solution for maximizing the benefits offered by directional antennas.

Introduction

Ad hoc networks allow to overcome the need for a centralized router or access point, and rely on node to node communication where each node is both a receiver and a transmitter and can relay data for other nodes. Such networks can be difficult to manage, and often rely on combinatorial optimization techniques to avoid interference between communications. These occur when a receiving antenna enters the emission beam of a node with which it is not supposed to communicate. When the nodes are mobile, most optimization methods become irrelevant because of their heavy computation time and routing network protocols such as OLSR (Jacquet et al. 2001) or AODV (Perkins and Royer 1999) with omnidirectional antennas are generally the preferred solution. Such protocols are widely used and present many practical advantages, but are limited in terms of achievable throughput, notably because of the high interference brought about by the omnidirectional antennas (Gupta and Kumar 2000). Using directional antennas allows for higher theoretical performance (Yi, Pei, and Kalyanaraman 2003), but requires making coherent antenna steering decisions, which have to be planned and computed interdependently. In order to be valid, the links motivating the antenna steering decisions should also respect several physical constraints, such as the placement of the antennas on the nodes, or a maximum emission range. Traditionally, optimizing such networks aims at scheduling the communications, or at finding appropriate routing schemes for the network packets. The rise of Software-Defined Networking (Kreutz et al. 2014) allows to access, gather and process all the nodes' information. The use of directional antennas allows for high throughput and reduced antenna emission beams. The combination of these two improvements makes it possible to optimize the performance of a network by directly searching for an efficient geometric configuration of the link topology: the carefully selected links would natively provide global connectedness, low interference and good network performance, even with minimal temporal scheduling and simple, straightforward routing. While it is often done, for static ground networks, using combinatorial optimization (Feng et al. 2016; Benhamiche, da Silva Coelho, and Perrot 2019), it requires a computationally heavy algorithm to converge, which is not suitable for most application cases where real time computation is needed, and nearly impossible when the nodes are moving. Even when using strong heuristics, the execution time of such methods can hardly be achieved in less than a few dozens of seconds because of the high number of variables and the interdependence between them. Designing a static link assignment algorithm between nodes should also fail since it would not be viable as soon as the nodes move and swap places. Luckily, graph neural networks are permutation invariant. Using deep graph neural networks to escape the combinatorial nature of the problem by leveraging efficient pattern learning to capture viable graph properties and reproduce them hence seems a particularly good solution to this problem. Our

*This work was carried out by Félix Marcoccia as part of his PhD thesis at Thales SIX.

framework consists in setting up a dataset of viable network link topologies, obtained by computationally expensive algorithms, and learning to replicate their construction patterns for new unseen sets of nodes, with a neural network.

We propose to extend the denoising diffusion (Ho, Jain, and Abbeel 2020) framework to solve the problem by

- Enriching the nodes and edges with intermediate-level features
- Proposing additional loss terms to facilitate the compliance with the constraints
- Deriving a novel graph-level mechanism, Cross-Attentive Modulation (CAM) tokens, to enable more expressive and deeper structural and global control over the inferred edges
- Introducing a simple way to extend our framework to cover the temporal evolution of the network where small topology reconfigurations are needed because of the nodes' movement.

Related Work

Inferring a set of links corresponding to a set of nodes is often treated as a generative problem in order to capture the joint distribution of the links, and grasp the graph as a whole, better than multiple standard link predictions would. It can then be modeled as a probabilistic problem that implies estimating the conditional joint distribution of the edges E knowing V, which is a classical formulation for conditional generative models. As discussed in (Zhu et al. 2022), graph generative models can be divided into five categories: autoregressive models, variational autoencoders, normalizing flows, generative adversarial networks and denoising diffusion probabilistic models (DDPMs). Autoregressive models rely on implicitly inducing a node ordering in order to process them sequentially, which (You et al. 2018) does by operating a breadth-first search to learn and generate coherent sequences of nodes and conform edges. In (Cao and Kipf 2022), the authors use a generative adversarial network helped by objective-driven reinforcement learning, an iterative is proposed in (Wang et al. 2017). Such a method relies on manipulating noise with a multilayer perceptron that does not fulfill node-conditioned architectures' need for permutation equivariance. In (Martinkus et al. 2022), authors propose to generate spectral properties before generating the graph accordingly, unfortunately, the sampling and generation process is once again not fully fitted for node-conditioned generation. Flow-based models, such as autoregressive (Shi et al. 2020) are also a possible solution for graph generation. Such iterative methods are generally incompatible or sub-optimal for node-conditioned generation as the iterative sampling or sequential node generation process does not allow for proper node-conditioning. Several variational methods such as (Kipf and Welling 2016) or (Simonovsky and Komodakis 2018) have been proposed. The first one relies on a node-level sampling procedure that assumes nodes' independence, the second one uses a graph level latent space that is then decoded into a graph. The main difficulty of such a method is that it requires complex matching algorithms to compare the reconstructed nodes and edges with the labeled ones. When the decoding is node-conditioned, where each node is concatenated with the latent vector and the decoding processed at node level, the conditioning signal can serve as a node identifier (a reconstructed node to which an attribute a_k is concatenated will be compared to the labeled node with attribute a_k if the attributes are unique). Such decoding can help to avoid the matching process and keep the permutation equivariance, but the model then suffers from the bottlenecks attached to such a simple architecture, especially given that the decoder linear layers should be applied at node level. More recently, Langevinesque (Welling and Teh 2011) variational methods have become popular to generate graphs, in particular denoising diffusion probablistic models, as seen in (Hoogeboom et al. 2022), (Vignac et al. 2023b) or (Vignac et al. 2023a) that respectively provide rotation invariance and adaptative discrete noise schedule. In (Wang et al. 2024a), authors use an autoencoder architecture to encode rich molecular properties into a latent space that is decoded using diffusion. Graph neural networks have been applied to the field of communication networks in various ways, one of the most famous ones being RouteNet (Rusek et al. 2020; Ferriol-Galmés et al. 2023) which allows to forecast the performance of a network, or (Azzouni, Boutaba, and Pujolle 2017) which infers routing schemes based on the evolution of a network. While fewer works have proposed building network topologies, (Lei et al. 2019) provides a temporal generative framework that aims at refining a network's topology based on its previous states, and (Wang et al. 2024b) uses a DDPM to optimize the link topology of a small sensors and communication network. DDPMs are particularly powerful to grasp the interdependence of the inferred links, as the intermediate diffusion steps allow the model to iteratively consult and adjust them. Many graph neural networks (Scarselli et al. 2009; Li et al. 2017) have been proposed, from graph convolutional networks (Kipf and Welling 2017) to graph attention networks (Veličković et al. 2018), or the expressive (Maron et al. 2020). Recently, graph transformers (Dwivedi and Bresson 2021) have proven to be the go-to architecture in DDPM frameworks. Their edge-conditioned attention-based message passing is particularly powerful in modeling interdependent relationships between nodes and edges. Such models, despite their expressiveness, tend to struggle to capture global features. To alleviate this issue, in (Vignac et al. 2023a), authors use statistical features, or graph properties, to condition FiLM (Perez et al. 2017; Brockschmidt 2020) layers applied on the nodes and edges. For similar reasons, (Darcet et al. 2023) introduces registers to serve as attention relays for image patches to store and retrieve information.

Background

Our problem

The ad hoc directional antennas networks we seek to build must provide connectedness, high throughput capability and hence incorporate interference constraints in their topology generation algorithm. In depth link topology optimization

allows to aim for a light, fast transmitting/emitting 2-slot scheduling, with virtually no further temporal scheduling, which grants high theoretical performance. It is translated in the graph problem as a parity label P on the nodes, and as a constraint on the links, which are hence only possible between odd pairs of nodes. Our nodes also feature antennas, which are needed to form links. We consider each node to have four orthogonal antennas, each of which covering a $\frac{\pi}{2}$ angle. Two links in a single antenna sector imply that the antenna must schedule its communications to use alternatively each link. Our approach then consists in imitating the results of a greedy iterative network topology algorithm. Without entering technical details, the process can be sketched as follows:

Algorithm 1: Greedy Network Topology Algorithm

```
Initialize the set of all possible links.
while the graph is not connected do
    while there are links to examine do
        Select the best link according to a heuristic
          (e.g., link length).
        if the interference from other links
          is below the threshold then
            Connect the nodes with the pair of antennas
              offering the best throughput.
        else
            Abandon the link.
```

While its greedy nature makes it far less computationally expensive than exact methods, it is still at least $O(m^2)$, with m the number of physically possible links, even when simplifying some of the computations and loops. While different kinds of algorithms can be used for similar purposes, iterative episodes of interference computation, as well as systematic constraints checks, are necessarily computation heavy. We then seek to find a neural network that outputs a set of edges $E = \{e_{1,1}, e_{1,2}, ..., e_{n,n}\}$ and the corresponding parity $P \in [\![0, 1]\!]^n$, from the nodes information $V = \{v_1, v_2 v..., v_n\}$, which together constitute the graph G, with n being the number of nodes. The nodes are described by their spatial coordinates c and their rotation.

Node-conditioned Denoising Diffusion Probabilistic Model

Recent works have proven Denoising Diffusion Probabilistic Models (DDPMs) to be particularly effective to tackle the task of generating plausible graphs. In order to approximate the distribution that we assume our data to follow, graph DDPMs propose, in the continuous setting, to search for a $p_\theta^t(\mathbf{G^{t-1}}|\mathbf{G^t})$ that approximates a markovian reverse noise process. This, contrary to directly maximizing the likelihood of $p_\theta(\mathbf{E}|V)$, is directly tractable. In our case, we want to compute the edges E given the nodes V, hence we apply the denoising process on edges only. We would then wish to find $p_\theta^t(\mathbf{E^{t-1}}|\mathbf{E^t}, V) = p_\theta^t(\mathbf{E^{t-1}}|\mathbf{G^t})$.

NetDiff relies on discrete denoising with data-aware state transition matrices, as it proved to be the most effective method in (Vignac et al. 2023a). We then directly model $p_\theta^t(\mathbf{E}|\mathbf{G^t})$. The noise is applied to uniform state transition matrices Q defining the evolution of the edges through the diffusion process. Our transition matrices Q are proportional to a scalar m that is correlated with the true distribution of the dataset links.

The transition matrix at step t is given by $\bar{Q}^t = \bar{\alpha}^t I + \bar{\beta}^t m$, where $\bar{\alpha}^t = \cos(0.5\pi(t/T + s)/(1 + s))^2$ with a small s and $\bar{\beta}^t = 1 - \bar{\alpha}^t$.
The learning procedure solely resides in minimizing $\sum_{ij} l_{BCE}(E, p_{(\theta,t,ij)}(G^t))$.

Sampling with the DDPM to generate new graphs iteratively follows

$$E^{t-1} \sim \prod_{ij} \hat{p}^E_{(\theta,t,ij)}(G^t)\, Q(e_{ij}^{t-1}|e_{ij} = e, e_{ij}^t), \quad (1)$$

with $\hat{p_\theta}$ being an estimation of p by a neural network parametrized by θ, Q representing the distribution of the scheduled noisy transition matrix.
This formulation is particularly valuable since the making of intermediate graph predictions at each timestep t allows each edge and node to adjust its value depending upon the last observed prediction, which can help modeling the interdependence between the edges. We follow the same procedure to denoise a noisy parity label. For a training step t, we seek to minimize $\sum_{ij} l_{BCE}(P_{ij}(G), \hat{P}_{\theta,ij}(G^t))$. The trained model is given as inputs the noisy edges E^t, the nodes information V, and the normalized timestep t. Even though DDPMs denoise each edge independently, the denoising decisions should depend on the other edges in order to grasp the interdependence between them while iterating towards realistic solutions, which implies that the chosen model should model both edge to node and edge to edge dependencies. We choose to follow this framework without using an encoder-based architecture (using a graph neural network followed by a graph pooling layer) as our graphs did not seem to detain sufficient information (especially considering that the nodes should not be encoded in the latent space, the model being conditioned on the nodes), to be continuously encoded in an efficient manner.

Solution architecture

Graph Transformer with CAM tokens

We use a graph transformer, which takes the noisy edges E^t , the nodes V, and the diffusion timestep t as inputs, and outputs a prediction of the correct edges E, and a corresponding parity P. The timestep t is incorporated to the model using a FiLM (Perez et al. 2017) layer. Graph transformers are $O(n^2)$ complex and parallelizable, which guarantees much better scalability with the number of nodes than the at least $O(m^2)$ complex greedy algorithm 1. Attention-based mechanisms can even be made to be $O(n\, log(n))$ complex, using masked attention for lighter, sparse computations. Graph transformers use the input edges to modulate node attention and modulate back these

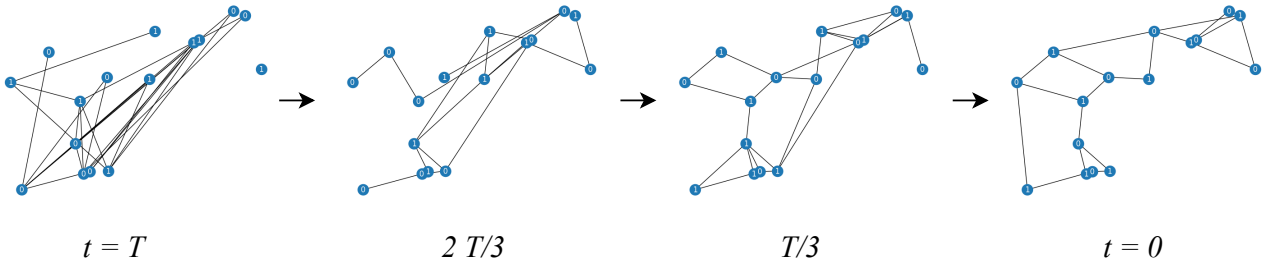

Figure 1: NetDiff iteratively denoises edges to form a plausible topology, nodes are not applied any noise.

edges as a function of the node attention product, which proves to be a great asset to compute coherent sets of links.

We augment the model with CAM tokens, which cross-attend to the graph and modulate (Perez et al. 2017; Brockschmidt 2020) the model's features accordingly in order to improve its global awareness. We use CAM tokens to modulate both nodes and edges. We hereby derive the CAM computations for nodes, they apply similarly to edges.

The value of a CAM token is initialized as

$$H^{\ell=0}_{\text{CAM}} = \omega_0, \tag{2}$$

which is a learnable parameter.
In order to gather information across the graph, CAM token updates its value following

$$H^{\ell}_{\text{CAM}} = \text{CrossAttn}(Q^{\ell-1}_{\text{CAM}}, K, V) + H^{\ell-1}_{\text{CAM}}, . \tag{3}$$

We use standard cross attention, which is expressed by

$$\text{CrossAttn}(Q^{\ell-1}_{CAM}, K, V) = \frac{Q^{\ell-1}_{CAM} K^T}{\sqrt{d_k}} V, \tag{4}$$

with $Q^{\ell}_{CAM} = W^Q H^{\ell-1}_{\text{CAM}}, \quad K = W^K H, \quad V = W^V H,$
H being the nodes' embeddings.

The token value is then mapped to a modulation as in (Perez et al. 2017). We choose γ and β to be simple affine layers. The modulation is conditioned by the product between the CAM token value H^{ℓ}_{CAM} and the value of each node.
Hence we have

$$\gamma^{\ell}(H^{\ell}_{\text{CAM}}, H) = W^{\gamma^{\ell}}(W_H H(W_{\text{CAM}} H^{\ell}_{\text{CAM}})^T) + b^{\gamma^{\ell}}, \tag{5}$$

$$\beta^{\ell}(H^{\ell}_{\text{CAM}}, H) = W^{\beta^{\ell}}(W_H H(W_{\text{CAM}} H^{\ell}_{\text{CAM}})^T) + b^{\beta^{\ell}}. \tag{6}$$

A simplified CAM update of the nodes (ignoring the specific architecture and activation function used) is then

$$H^{updated} = \gamma^{\ell} H + \beta^{\ell}. \tag{7}$$

A CAM token incorporating node and edge features follows

$$H^{\ell}_{\text{CAM}} = \text{FFN}(\text{CrossAttn}(Q^{\ell-1}_{\text{CAM}}, K, V), \\ \text{CrossAttn}(Q^{\ell-1}_{\text{CAM}}, K_{edges}, V_{edges})) + H^{\ell-1}_{\text{CAM}}. \tag{8}$$

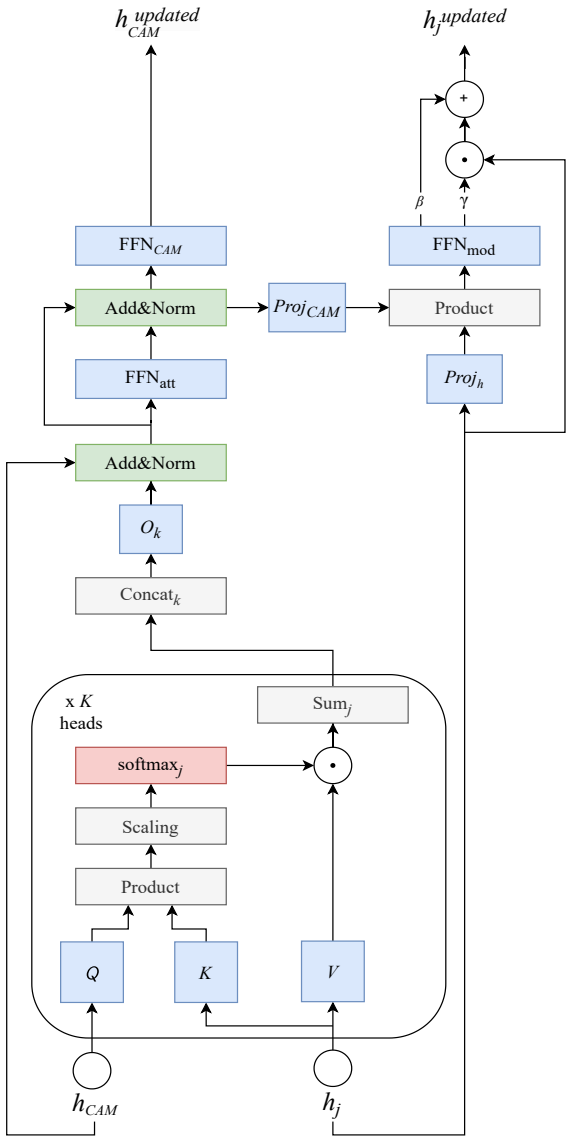

Figure 2: Architecture of a CAM token

CAM tokens then consist, for a light computational cost, in incorporating an expressive global mechanism which can help tracking important graph-level properties and adapting the behavior of the model accordingly. On the one hand, these tokens are valuable to relieve the nodes of the need to feature long range, high-level feature message passing, and could hence reduce oversmoothing (Wu et al. 2024). On the other hand, conditioning the features of the model on the observed graph enables more flexibility and facilitates the processing of difficult node layouts or unbalanced edges that tend to distort the attention-based message passing.

Additional features and loss components

We concatenate to each node the number of links that it features on each cardinal antenna sector. We penalize the model when a node features more than 1 link in an antenna sector. If $n_i^s = \sum_j e_{ij} \cdot \delta(s(i,j) = s)$, with $\delta(s(i,j) = s)$ being the indicative function equal to 1 if e_{ij} is in sector s, 0 otherwise, is the number of links on a given sector of a node i, the penalty for this node is then given by

$$\mathcal{L}_i^{sectors} = \sum_{s=1}^{4} \text{ReLU}(n_s^i - 1). \tag{9}$$

We concatenate to each edge its angle with the horizontal axis. The angle attribute for a given edge e_{ij} is then given by $\psi(e_{ij}) = \arctan\left(\frac{c_j^y - c_i^y}{c_j^x - c_i^x}\right)$. We apply a cosine loss, as used in (Garrido et al. 2022), in order to penalize acute angles, which are globally rare in the dataset since they physically correspond to sub-optimal links.
It is computed following

$$\mathcal{L}_{\cos} = 1 - \frac{1}{n} \sum_{h \in H} \frac{2}{k(k-1)} \sum_{1 \le i < j \le k} \frac{(c_j - c_h) \cdot (c_h - c_i)}{\|c_j - c_h\| \cdot \|c_h - c_i\|}, \tag{10}$$

with k the number of neighbors of the node h.
The added features and loss functions mostly aim at facilitating the respect of the physical placement of the antennas. Finally, we add a parity-related loss term that penalizes links between nodes of the same predicted parity:

$$\mathcal{L}_{odd} = \frac{\sum_{(i,j) \in E} \hat{e}_{ij} \cdot \left(1 - \left|\hat{P}(i) - \hat{P}(j)\right|\right)}{1 + \sum_{(i,j) \in E} \hat{e}_{ij}}. \tag{11}$$

During training, we start by optimizing the reconstruction BCE for 5 to 10 epochs. We then add the parity-related loss terms so that they weigh as much as the main loss. After another 5 to 10 epochs, the main BCE is generally stable and close to 0, we then add the cosine and sector loss terms and gradually increase them for at least 10 epochs.

Network evolution

Totally reconfiguring the whole network at every slight movement, every few seconds, could cause an important degradation of the quality of service as the packets would always be rerouted. The solution would also be computationally ineffective. Signaling resulting from a change in link topology would also cause a significant communication overhead. In order to solve this issue, we simply extend the diffusion paradigm to provide a simple yet efficient way to quickly update the network without causing drastic topology changes. The network topology simply follows an adapted denoising diffusion process with the last link topology $E^{previous}$ as a starting point, using the updated nodes' positions. It does not inherently differ from a standard noisy graph given that the matrix m used to apply noise is proportional to the real distribution of the edges and that we assume the movement-caused change of links to be uniform and homogeneous to our noise model. We quantify the total normalized amount of node movement and make the number of steps of diffusion to be proportional to that amount of movement. We choose not to apply any noise when the movement is sufficiently small, meaning that the network update diffusion process only uses the last topology and its new iterative predictions to generate the new graph. The process for significant reconfigurations then consists in sampling

$$E^{t-1} \sim \prod_{ij} \hat{p}^E_{(\theta,t,ij)}(G^t)\, Q(e_{ij}^{t-1} | e_{ij} = e, e_{ij}^t), \tag{12}$$

with $E^{t=T} = E^{previous}$, T being proportional with the quantity of nodes' movement measured since the previous reconfiguration. When nodes have only slightly moved, we use $\bar{R}^t = \bar{\alpha}^t I + \bar{\beta}^t E^{previous}$ in the noise schedule. The sampling then follows

$$E^{t-1} \sim \prod_{ij} \hat{p}^E_{(\theta,t,ij)}(G^t)\, R(e_{ij}^{t-1} | e_{ij} = e, e_{ij}^t). \tag{13}$$

We also follow the guided sampling diffusion framework introduced in (Dhariwal and Nichol 2021), with $\sum_{ij} l_{BCE}(E_{ij}^{previous}, \hat{E_{ij}}^t)$ replacing the classifier loss, with a scaling factor depending on the amount of nodes' movement since the last reconfiguration. Graphs obtained using this method present properties similar to the ones obtained using the general framework. They are also about **21 %** more similar to the previous link topology than if they had been produced using the general framework. For a typical mobility scenario (where each node randomly moves with a maximal amplitude of 3/10th of the diagonal of the operational zones), we obtain satisfactory results following only 10 to 15 diffusion steps, resulting in 3.3 to 5 times faster computation. Using the minor reconfiguration framework, which replaces noise with the previous topology, we produce **41 %** more similar graphs than if using the general framework, and obtain satisfactory results following only 7 to 10 diffusion steps. Experimental results and observations are to be found in the appendix 1.

Results

Benchmarks

NetDiff follows 50 diffusion steps. The graph transformer that we use features 10 graph transformer blocks. We trained our models on 22k samples, using Pytorch (Paszke et al. 2019), AdamW (Loshchilov and Hutter 2019) with high (0.1 to 0.17) weight decay to avoid overfitting on our

Model	Efficiency ↑	Connectedness ↑	Isolated nodes ↓	Saturated nodes ↓	Parity ↑	Length ↑
NetDiff	**1.99**	99.13 %	0.29 %	**12.31 %**	**98.65 %**	98.97 %
NetDiff (no features)	1.89	99.07 %	0.32 %	14.83 %	98.64 %	**99.01 %**
DDPM-GT	1.78	98.12 %	0.34 %	17.08 %	98.59 %	98.87 %
GraphVAE	0.66	**99.82 %**	**0.12 %**	96.08 %	61.33 %	84.87 %
GT-VAE	1.16	98.82 %	0.39 %	49.71 %	93.48 %	97.93 %

Table 1: Constraint respect of different architectures

rather small dataset, and learning rates between 1e-4 and 1e-7. We conduct our experiments with an Intel Xeon(R) E5-2650v3 at 2.30 GHz CPU and a Tesla T4 GPU. The entire diffusion runs in $\sim$ 480 ms on GPU and 2 seconds on CPU without any optimization. We use a noise schedule that pushes the model to rather create supernumerary links than the contrary, as it is easier to remove links afterwards than to create new links manually. The benchmarks are conducted using thousands of random sets of 16 nodes, unseen during training. Although the model is theoretically agnostic to the number of nodes, featuring sets of different numbers of nodes in the dataset allows for better generalization and scalability, as detailed in the appendix 2. We compare NetDiff to the existing node-conditioned generative architectures: a standard graph transformer DDPM baseline, similar to the (Vignac et al. 2023a), GraphVAE (Simonovsky and Komodakis 2018) and another graph variational autoencoder, GT-VAE, using a more expressive graph-transformer-based decoder. NetDiff (no features) refers to our architecture without the handcrafted features and additional loss terms.

A crucial aspect of our problem is to respect the strict constraints that govern the dataset networks. In Table 1, "**Efficiency**" refers to the proportion of connected graphs normalized by the number of predicted links. It is an important metric since, while the topologies have to be connected, generating too many links creates saturation and interferences. The target algorithm has a **2.28** efficiency. "**Connected**" describes the absolute proportion of connected graphs, "**Isolated nodes**" - the proportion of nodes that do not feature any link, "**Saturated nodes**" - the proportion of nodes that feature more than 4 links, "**Parity**" - the proportion of links that respect the generated parity, "**Length**" - the proportion of links that respect the maximal possible range. We observe that NetDiff respects most of the constraints faithfully, provides high efficiency and only shows minor difficulties to respect the maximal amount of links per node. While it could be improved, it is also due to the decision to push the generation to generate slightly more links in order to facilitate graph post-processing. Individual link constraints seem easier to respect. Our model outperforms vanilla graph transformer DDPM by a consequent margin, especially in structural connectivity-related constraints such as node saturation. The handcrafted features, in addition to the CAM token, also seem to improve the performance of the model. Variational-autoencoder-based architectures struggle at modeling the desired graphs: while the vanilla version does not seem to be able to even distinguish plausible edges and nearly predicts every possible link, GT-VAE predicts individually plausible edges but does not use them to form fully coherent graphs. We found that the encoded latent was nearly ignored which, as mentioned previously, is probably due to the node-conditional nature of the generation task. It should be noted that GraphVAE performs best in connectedness and isolated nodes because it predicts such a disproportionately high number of links that the predicted topologies are inherently connected. NetDiff then brings substantial improvements over a standard discrete denoising graph transformer and vastly outperforms one-shot variational methods.

	Omnidirectional	GT-VAE	**NetDiff**
Throughput ↑	50.16	66.31	**95.05**

Table 3: Avg instantaneous throughput upper bound (Mbps)

Property	NetDiff	DDPM-GT	GraphVAE	GT-VAE	Target
Number of links	**49.74**	55.22	151.81	85.67	43.90
Average link length	**1.04**	1.05	1.51	1.07	0.85
Link throughput	**1.60**	1.49	0.67	1.33	1.72
Saturated antennas	**16.19** %	24.59 %	90.04 %	42.71 %	9.74 %

Table 2: Some properties of the generated graphs compared to the original ones

In Table 2, we highlight in bold the model whose the evaluated property is the most similar to the target's. NetDiff stands out as a clear improvement over the other benchmarked models. Generated graphs feature more links than the dataset samples, but their links are rather similar in terms of length. Our model generates graphs whose simulated per-link throughput (in Mbps, using 10 Mbps as the source node throughput) lies close to the original graphs', despite showing almost twice as many saturated antennas (antenna sectors which feature more than one link). The results show the capacity of our architecture to capture high-level connectivity pattern rules since NetDiff shows significant improvement in the number of links and the amount of saturated antennas. GraphVAE does not seem to discriminate links in an efficient manner, GT-VAE also seems overoptimistic in its predictions.

In Table 3, we showcase theoretical instantaneous throughputs corresponding to an upper bound of the amount of information that the network can exchange at the same time. We do not feature any scheduling or routing, as they are outside of our scope. We seek to measure in what extent NetDiff-generated networks do respect the theoretical assumptions made in (Yi, Pei, and Kalyanaraman 2003; Gupta and Kumar 2000) regarding the performance of directional antennas. We use similar geometrical constructions for interference patterns. The omnidirectional protocol is based on a simplified OLSR where the nodes are considered to communicate with all the neighbors in their range, which is about 1/4th of the distance between the two most distant nodes in the graph. We consider that the received interference decreases quadratically with the distance to the interfering emitter. If the interference is too high, the link is required to delay its communication and is hence considered inactive at the ongoing timestep. We use a single available frequency, a directional antenna beamwidth of $\frac{\pi}{3}$ for NetDiff and GT-VAE, and Shannon's capacity as the theoretical upper bound for channel performance. We consider the directional antenna gain to be 2 times superior to the omnidirectional one.
NetDiff provides significantly higher instantaneous throughput upper-bounds than the omnidirectional counterpart. This measurement primarily depends on, and is inversely proportional to, the level of interference and the number of links deactivated to mitigate interference. It confirms that the created topologies feature a sufficient amount of low-interference links to achieve the expected throughput improvement compared to the omnidirectional approach. The poorer GT-VAE-based topologies do not stand out as a great improvement over the omnidirectional counterpart, and suggest that directional antennas do indeed require a proper link topology to obtain an important performance improvement. NetDiff also scales much better than the omnidirectional counterparts by design, results in appendix).

Constraint	**Our Loss**	Standard BCE
Saturated antennas $\downarrow$	**16.19 %**	23.38 %
Parity $\uparrow$	**98.65 %**	67.07 %

Table 4: Zoom on loss functions effects

Our additional loss function terms greatly fulfill their task, displaying important improvement on the model's ability to respect the constraint they refer to. The cosine and sector loss terms seem to diminish antenna-wise supernumerary links by a significant amount, while the parity respect loss term appears mandatory for the model to respect the parity link constraint. Hence, we observe NetDiff, equipped with CAM tokens, shows great control over the connectivity of the generated graphs, and that the additional features and loss terms are valuable in order to respect the constraints of the problem. It greatly outperforms a graph transformer DDPM and displays convincing results, close to the original graphs', while providing higher throughput than reference network protocols.

Operational use

In order for NetDiff to be robust, we implement a simple algorithm to correct the network provided by the DDPM. Its execution time is **at most 250 ms** in our experiments.

Algorithm 2: Correction Algorithm

for each node in the graph **do**
    Assign each link to the sector it belongs to based on its angle.
    Delete wrong parity links.
    if multiple links are present in the same sector **then**
        Keep only the shortest one.
Check if the graph is connected.
while the graph is not connected **do**
    for each pair of disconnected components **do**
        Connect the closest pair of nodes with opposing free sectors.

As mentioned in the prelude of the section, we push our model to generate more links than in the dataset graphs, as removing a few redundant ones is not computationally expensive, while having to add them is a bit more difficult. We could check, for each removed link, if the removal would disconnect the graph, though, in practice it almost never does and would be corrected by the components-connecting loop. We can accept having a few saturated antennas as they will simply alternatively serve each of their links depending on the demand, resulting in a minor loss of throughput.

Conclusion

In conclusion, NetDiff provides a flexible and expressive denoising diffusion architecture to imitate network topologies obtained with a heavy algorithm. Such topologies grant optimized connectivity patterns to avoid interference and achieve high network performance using directional antennas. Compared to their omnidirectional counterparts, NetDiff-generated topologies achieve substantially higher instantaneous throughputs, fully leveraging the potential of directional antennas. This enables scheduling with really few time frames and facilitates a straightforward routing process. The presented architecture benefits from cross-attentive modulation tokens, which help generating coherent graphs with realistic structural properties. The additional loss terms greatly facilitate the enforcement of the constraints and are particularly effective in ensuring the parity of communications. Futhermore, our partial diffusion method provides a significant computational gain and enhances network stability, especially for minor reconfigurations. Our empirical results show that the proposed architecture enables significant improvements over the popular denoising graph transformer, and vastly outperforms one-shot generative methods. Future work will explore further optimizations in link verification processes and incorporate NetDiff in realistic network environments.

Appendices

A Mobility model, results and observations

We conduct the following experiments in order to evaluate the same properties as the ones evaluated in the general framework. We apply gaussian noise to the nodes' coordinates of true topologies, and apply NetDiff on the new set of nodes, with the previous set of edges as the starting point of the diffusion. The standard evolution corresponds to a noisy partial reconfiguration of 15 steps with a normalized gaussian movement of maximum amplitude of 0.3 and the minor evolution to the less noisy small reconfiguration of 10 steps with a normalized gaussian movement of maximum amplitude of 0.1. We chose the number of steps that seemed to offer the best computation time/performance ratio.

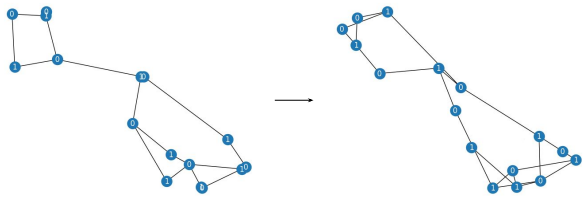

Figure 3: A true network topology (on the left) follows a standard reconfiguration of 15 diffusion steps. While the generated topology shares structural similarities with the base topology, important changes have been made (0.3 normalized node movement).

Reconfiguration	**Connected** ↑	**Isolated nodes** ↓	**Saturated nodes** ↓	**Parity** ↑	**Length** ↑
Standard reconfiguration	98.85 %	0.33 %	14.09 %	98.50 %	98.04 %
Minor reconfiguration	98.87 %	0.33 %	12.89 %	98.64 %	99.01 %

Table 5: Constraint adherence with and without architectural augmentations

We can observe that constraint adherence is close to the one observed using the standard framework, with only a slight degradation when using the standard reconfiguration. We found out the added continuity-enforcing guidance to be responsible of most of this loss of performance.

Property	Standard reconfiguration	Minor reconfiguration
Number of links	49.84	47.62
Average link length	1.06	1.04
Link throughput	1.49	1.55
Saturated antennas	16.70 %	15.68 %
Continuity	33.21 %	38.72 %
Diffusion steps	15	10

Table 6: Properties of the generated graphs using the two evolutional diffusion algorithms

The generated graphs have properties that seem close to the ones obtained using the general framework. **Continuity** measures the proportion of links that were also present in the previous topology. Using the general diffusion framework, we would obtain a **27.43%** continuity score. We applied a rather weak guidance that we empirically found not to affect the quality of the generated graphs. The performance/guidance trade-off could be addressed following the framework developed in (Li et al. 2022). The small number of diffusion steps allows for up to 5 times faster graph generation.

We hereby provide two examples of graphs obtained using the mobility methods: the first one using the standard noisy partial reconfiguration, the second one using the minor reconfiguration.

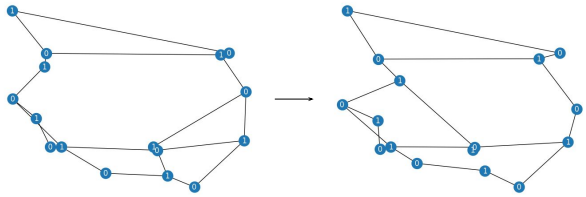

Figure 4: A true network topology (on the left) follows a minor reconfiguration of 10 diffusion steps. Very few links have been changed during the reconfiguration (0.1 normalized node movement).

For each of the above reconfigurations, a correction algorithm following guidelines operated one link change/removal. The predicted graphs were then almost natively correct, resulting in a $\sim$ (500 for inference + 50 for correction) ms execution time.

B Network size flexibility

While our framework is theoretically agnostic to the size of the input networks, obtaining similar performance on 16 or 32-node networks does require the model to "see" 32-node-networks in the training stage. We then trained the model in order to provide similar performances as the ones detailed in Tables 1 and 2 for 16-node-networks while generalizing better to 32-node-networks (40+ nodes networks need to be partitioned for communication stability so the case is not covered in this work). We used a reduced dataset of 16 and 32-node networks with a 60/40 ratio to retrain our model. Since the training on various sizes could not be batched, we only retrained our model for an equivalent of 2 epochs on 15k samples.

Model	Connected ↑	Isolated nodes ↓	Saturated nodes ↓	Parity ↑	Length ↑
NetDiff (32)	98.76 %	0.37 %	19.06 %	96.74 %	99.20%

Table 7: Constraint adherence for 32 node-networks

The created topologies grant similar constraint adherence as their 16-node-counterparts except for the node saturation, which seems harder to respect there.

Property	NetDiff (32)	Target
Number of links	108.75	96.44
Average link length	0.81	0.70
Link throughput	2.69	3.13
Saturated antennas	20.89 %	9.91 %

Table 8: Some properties of the generated graphs compared to the original ones

The model tends to create proportionally more supernumerary links than for 16-node-networks, which suggests that the model tends to be a bit biased by the number of nodes. It obviously affects the antenna saturation metric and explains the node saturation observed beforehand. NetDiff still provides satisfactory link lengths and simulated throughput. Creating a more balanced and continuous dataset between 16 and 32-node-networks would greatly improve the results. The link throughput is higher than in the 16-node-scenarii because the nodes are closer to each other, resulting in better signal to noise ratios.

	Omnidirectional	**NetDiff**
Throughput ↑	92.34	**288.70**

Table 9: Avg instantaneous throughput upper bound (Mbps)

NetDiff, which relies on advanced spatial reuse to provide high-performance link topologies, seems to provide much better scaling than the omnidirectional-antenna-based networks, which suffer from a significant interference increase.

References

Azzouni, A.; Boutaba, R.; and Pujolle, G. 2017. NeuRoute: Predictive Dynamic Routing for Software-Defined Networks. arXiv:1709.06002.

Benhamiche, A.; da Silva Coelho, W.; and Perrot, N. 2019. Routing and Resource Assignment Problems in Future 5G Radio Access Networks.

Brockschmidt, M. 2020. GNN-FiLM: Graph Neural Networks with Feature-wise Linear Modulation. arXiv:1906.12192.

Cao, N. D.; and Kipf, T. 2022. MolGAN: An implicit generative model for small molecular graphs. arXiv:1805.11973.

Darcet, T.; Oquab, M.; Mairal, J.; and Bojanowski, P. 2023. Vision Transformers Need Registers. arXiv:2309.16588.

Dhariwal, P.; and Nichol, A. 2021. Diffusion Models Beat GANs on Image Synthesis. arXiv:2105.05233.

Dwivedi, V. P.; and Bresson, X. 2021. A Generalization of Transformer Networks to Graphs. arXiv:2012.09699.

Feng, W.; Li, Y.; Jin, D.; Su, L.; and Chen, S. 2016. Millimetre-Wave Backhaul for 5G Networks: Challenges and Solutions. *Sensors*, 16(6): 892. Number: 6 Publisher: Multidisciplinary Digital Publishing Institute.

Ferriol-Galmés, M.; Paillisse, J.; Suárez-Varela, J.; Rusek, K.; Xiao, S.; Shi, X.; Cheng, X.; Barlet-Ros, P.; and Cabellos-Aparicio, A. 2023. RouteNet-Fermi: Network Modeling With Graph Neural Networks. *IEEE/ACM Transactions on Networking*, 31(6): 3080–3095.

Garrido, Q.; Damrich, S.; Jäger, A.; Cerletti, D.; Claassen, M.; Najman, L.; and Hamprecht, F. 2022. Visualizing hierarchies in scRNA-seq data using a density tree-biased autoencoder. arXiv:2102.05892.

Gupta, P.; and Kumar, P. 2000. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2): 388–404.

Ho, J.; Jain, A.; and Abbeel, P. 2020. Denoising Diffusion Probabilistic Models. arXiv:2006.11239.

Hoogeboom, E.; Satorras, V. G.; Vignac, C.; and Welling, M. 2022. Equivariant Diffusion for Molecule Generation in 3D. arXiv:2203.17003.

Jacquet, P.; Muhlethaler, P.; Clausen, T.; Laouiti, A.; Qayyum, A.; and Viennot, L. 2001. Optimized link state routing protocol for ad hoc networks. In *Proceedings. IEEE International Multi Topic Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century.*, 62–68.

Kipf, T. N.; and Welling, M. 2016. Variational Graph Auto-Encoders. arXiv:1611.07308.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. arXiv:1609.02907.

Kreutz, D.; Ramos, F. M. V.; Verissimo, P.; Rothenberg, C. E.; Azodolmolky, S.; and Uhlig, S. 2014. Software-Defined Networking: A Comprehensive Survey. arXiv:1406.0440.

Lei, K.; Qin, M.; Bai, B.; Zhang, G.; and Yang, M. 2019. GCN-GAN: A Non-linear Temporal Link Prediction Model for Weighted Dynamic Networks. ArXiv:1901.09165 [cs].

Li, X. L.; Thickstun, J.; Gulrajani, I.; Liang, P.; and Hashimoto, T. B. 2022. Diffusion-LM Improves Controllable Text Generation. arXiv:2205.14217.

Li, Y.; Tarlow, D.; Brockschmidt, M.; and Zemel, R. 2017. Gated Graph Sequence Neural Networks. arXiv:1511.05493.

Loshchilov, I.; and Hutter, F. 2019. Decoupled Weight Decay Regularization. arXiv:1711.05101.

Maron, H.; Ben-Hamu, H.; Serviansky, H.; and Lipman, Y. 2020. Provably Powerful Graph Networks. arXiv:1905.11136.

Martinkus, K.; Loukas, A.; Perraudin, N.; and Wattenhofer, R. 2022. SPECTRE: Spectral Conditioning Helps to Overcome the Expressivity Limits of One-shot Graph Generators. arXiv:2204.01613.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Köpf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. arXiv:1912.01703.

Perez, E.; Strub, F.; de Vries, H.; Dumoulin, V.; and Courville, A. 2017. FiLM: Visual Reasoning with a General Conditioning Layer. arXiv:1709.07871.

Perkins, C.; and Royer, E. 1999. Ad-hoc on-demand distance vector routing. In *Proceedings WMCSA'99. Second IEEE Workshop on Mobile Computing Systems and Applications*, 90–100.

Rusek, K.; Suárez-Varela, J.; Almasan, P.; Barlet-Ros, P.; and Cabellos-Aparicio, A. 2020. RouteNet: Leveraging Graph Neural Networks for Network Modeling and Optimization in SDN. *IEEE Journal on Selected Areas in Communications*, 38(10): 2260–2270. Conference Name: IEEE Journal on Selected Areas in Communications.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2009. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1): 61–80. Conference Name: IEEE Transactions on Neural Networks.

Shi, C.; Xu, M.; Zhu, Z.; Zhang, W.; Zhang, M.; and Tang, J. 2020. GraphAF: a Flow-based Autoregressive Model for Molecular Graph Generation. arXiv:2001.09382.

Simonovsky, M.; and Komodakis, N. 2018. GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders. ArXiv:1802.03480 [cs].

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. arXiv:1710.10903.

Vignac, C.; Krawczuk, I.; Siraudin, A.; Wang, B.; Cevher, V.; and Frossard, P. 2023a. DiGress: Discrete Denoising diffusion for graph generation. arXiv:2209.14734.

Vignac, C.; Osman, N.; Toni, L.; and Frossard, P. 2023b. MiDi: Mixed Graph and 3D Denoising Diffusion for Molecule Generation. arXiv:2302.09048.

Wang, C.; Ong, H. H.; Chiba, S.; and Rajapakse, J. C. 2024a. GLDM: hit molecule generation with constrained graph latent diffusion model. *Briefings in Bioinformatics*, 25(3): bbae142.

Wang, H.; Wang, J.; Wang, J.; Zhao, M.; Zhang, W.; Zhang, F.; Xie, X.; and Guo, M. 2017. GraphGAN: Graph Representation Learning with Generative Adversarial Nets. arXiv:1711.08267.

Wang, J.; Liu, Y.; Du, H.; Niyato, D.; Kang, J.; Zhou, H.; and Kim, D. I. 2024b. Empowering Wireless Networks with Artificial Intelligence Generated Graph. arXiv:2405.04907.

Welling, M.; and Teh, Y. W. 2011. Bayesian Learning via Stochastic Gradient Langevin Dynamics. In *International Conference on Machine Learning*.

Wu, X.; Ajorlou, A.; Wu, Z.; and Jadbabaie, A. 2024. Demystifying Oversmoothing in Attention-Based Graph Neural Networks. arXiv:2305.16102.

Yi, S.; Pei, Y.; and Kalyanaraman, S. 2003. 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, 108–116. New York, NY, USA: Association for Computing Machinery. ISBN 978-1-58113-684-5.

You, J.; Ying, R.; Ren, X.; Hamilton, W. L.; and Leskovec, J. 2018. GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models. arXiv:1802.08773.

Zhu, Y.; Du, Y.; Wang, Y.; Xu, Y.; Zhang, J.; Liu, Q.; and Wu, S. 2022. A Survey on Deep Graph Generation: Methods and Applications. arXiv:2203.06714.