Autoregressive D3PM for uniform spanning tree sampling on grid graphs

Pierre-Gabriel Berlureau
INRIA - ENS Paris
pierre-gabriel.berlureau@ens.psl.eu

Marc Lelarge
INRIA - ENS Paris
marc.lelarge@inria.fr

Abstract

Denoising diffusion probabilistic models (DDPMs) have demonstrated remarkable success in generating high-quality data across various domains. To extend this capability to discrete data, Discrete Denoising Diffusion Probabilistic Models (D3PMs) were introduced. In this work, we adapt D3PMs for the task of uniform spanning tree sampling on a grid graph, proposing an autoregressive approach that iteratively refines the sampled edge set along a fixed node schedule. Evaluations on grid graphs show that our approach effectively captures structural patterns, though the generated samples consistently lack full connectivity and contain cycles.

1 Introduction

Generative modeling has experienced remarkable advancements in recent years, with Denoising Diffusion Probabilistic Models (DDPMs) emerging as a powerful class of methods for high-quality data generation. Proposed by Ho et al. [3], DDPMs leverage a forward Markov process to iteratively add Gaussian noise to data, followed by a learned reverse process that denoises step-by-step to reconstruct the original data. These models have been applied successfully in various domains, including image [3] and waveform generation [6], achieving state-of-the-art results.

Extending this framework to discrete data poses unique challenges, as the inherent structure of discrete spaces is not naturally suited to Gaussian perturbations. Discrete Denoising Diffusion Probabilistic Models (D3PMs), introduced by Hoogeboom et al. [4], address this limitation by generalizing the diffusion process to discrete state spaces. D3PMs replace Gaussian noise with transition matrices designed to simulate corruption processes in discrete domains, such as multinomial noise or matrices inspired by Gaussian kernels in embedding spaces. This adaptation opens the door for D3PMs to excel in tasks like text generation and discrete graph synthesis [1, 2, 4].

In this work, we focus on leveraging D3PMs for uniform spanning tree sampling on grid graphs. We propose an autoregressive approach to enhance the preservation of tree properties while sampling. Our methods' iteratively denoises edge sets along a fixed schedule of nodes in the underlying graph, restricting denoisers to Graph Convolutional Networks (GCNs) for efficiency and scalability. We focus on uniform spanning trees (USTs) because the tree property is a global structural property defined by two easily verifiable conditions: connectivity and acyclicity. This makes spanning trees particularly interesting for studying discrete generative models. Additionally, USTs on grid graphs can be efficiently sampled using Wilson's algorithm [8], which leverages loop-erased random walks to generate exact samples. However, Wilson's algorithm has a worst-case complexity of $\mathcal{O}(n^3)$, which limits its scalability for large graphs. As part of our work, we aim to propose a more efficient and scalable method for generating uniform spanning trees.

Contributions.

- We extend D3PMs to the task of sampling uniform spanning trees on grid graphs within an autoregressive framework.
- Our approach incorporates a fixed node schedule and utilizes GCNs as efficient, scalable denoisers.

- Experimental results highlight the method's capability to generate structured subgraphs, though challenges remain in ensuring full connectivity and cycle-free outputs.

The rest of this paper is structured as follows: Section 2 explores a direct adaptation of classical diffusion models using both continuous and discrete edge representations. Section 3 introduces our autoregressive approach. Finally, Section 4 presents experimental results and outlines future research directions.

2 Classic methods

2.1 Denoising Diffusion Probabilistic Models

Denoising Diffusion Probabilistic Models, introduced by Ho et al. [1], have become a foundational approach in generative modeling. DDPMs operate through a two-step process: a forward diffusion step and a reverse denoising step. In the forward process, Gaussian noise is incrementally added to the data over multiple time steps, progressively transforming it into pure noise. Formally, for an initial data point x_0, the forward process at time t is defined as:

$$q(x_t \mid x_{t-1}) = \mathcal{N}(x_t; \sqrt{\alpha_t}x_{t-1}, (1-\alpha_t)I) \tag{1}$$

where α_t is a variance schedule controlling the noise level at each step.

In the reverse process, a neural network is trained to approximate the posterior distribution $p(x_{t-1}, x_t)$, allowing the model to denoise the data step-by-step until a clean sample is reconstructed:

$$p(x_{t-1} \mid x_t) \approx \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \tag{2}$$

DDPMs have demonstrated state-of-the-art performance in tasks such as image generation [1] and audio synthesis [2], leveraging their ability to model complex data distributions.

Adaptation. In order to adapt DDPMs to the task of uniform spanning tree sampling on a $n \times n$ grid graph we apply the diffusion process over its edges. To do so, we chose an ordering of those edges and map each of them to a continuous value in $[0, 1]$ representing its state. This gives us an initial vector representation $x_0 \in \mathbb{R}^d$ - where d is the number of edges - such that $x_0[i]$ is the state value of the i-th edge. We interpret the continuous state value of the i-th edge as its presence if $x_0[i] \geq 0.5$ and its absence if $x_0[i] < 0.5$. Thus, for a given spanning tree T on the $n \times n$ grid graph, $x_0[i]$ is equal to 1 if the i-th edge is in T and 0 if this is not the case. After applying the noising process to x_0, we reshape x_0 to form a grayscale image in which the continuous state value of the i-th edge is placed at its corresponding location in the grid representation of the underlying grid graph. This allows us to use a CNN as a denoiser, treating the task as generating images of uniform spanning tree on a grid graph using a classic DDPM.

Sinusoidal embedding. At each step we augment x_t by concatenating it along a new dimension with the classic sinusoidal embedding in order to give the current time step information to the model. Sinusoidal embeddings provide a continuous, differentiable method for encoding positional or temporal information, initially proposed by Vaswani et al. (2017) in the Transformer architecture [7]. The encoding for a given position or timestep t in a d-dimensional space is defined as:

$$PE(t, 2i) = \sin\left(\frac{t}{10000^{\frac{2i}{d}}}\right), \; PE(t, 2i+1) = \cos\left(\frac{t}{10000^{\frac{2i}{d}}}\right) \tag{3}$$

where i indexes the embedding dimensions. This formulation ensures that each dimension of the embedding corresponds to a sinusoidal function of a different frequency, allowing the model to infer relative positions from linear combinations of these embeddings. The periodic nature of sine and cosine functions enables the model to generalize to sequences or timesteps not seen during training.

2.2 Discrete Denoising Diffusion Probabilistic Models

While DDPMs excel in continuous data domains, their extension to discrete data poses significant challenges due to the incompatibility of Gaussian noise with discrete structures. To address this,

Hoogeboom et al. [3] introduced Discrete Denoising Diffusion Probabilistic Models, which adapt the diffusion framework to discrete state spaces.

In D3PMs, the forward process is defined by a transition matrix Q_t that governs the probability of transitioning between discrete states at each time step. This replaces Gaussian noise with discrete corruption mechanisms, such as multinomial noise or transitions inspired by Gaussian kernels in embedding spaces. The reverse process similarly involves learning the transition probabilities to denoise the discrete data iteratively.

Formally, for scalar discrete random variables with K categories $x_t, x_{t-1} \in 1, \ldots, K$ the forward transition probabilities can be represented by matrices: $[Q_t]_{i,j} = q(x_t = j \mid x_{t-1} = i)$. Denoting the one-hot version of x with the row vector x, we can write

$$q(x_t \mid x_{t-1}) = \text{Cat}(x_t; p = x_{t-1}Q_t) \tag{4}$$

where $\text{Cat}(x; p)$ is a categorical distribution over the one-hot row vector x with probabilities given by the row vector p, and $x_{t-1}Q_t$ is to be understood as a row vector-matrix product.

The reverse process seeks to approximate:

$$p(x_{t-1} \mid x_t) \approx P_\theta(x_{t-1} \mid x_t, t) \tag{5}$$

with P_θ parameterized by a neural network.

D3PMs have been successfully applied to tasks such as non-autoregressive language modeling [3] and discrete graph generation [4], highlighting their flexibility in handling structured discrete data.

Using this framework, we use the same mapping of edges to their state value as in the continuous framework. Except in this case x_t is in $\mathbb{R}^{d\times 2}$, as $x_t[i] \in \mathbb{R}^2$ corresponds to the categorical distribution of the i-th edge's state at time t over the $K = 2$ possible categories: $x_t[i][0]$ is the probability of the i-th edge presence at time t and $x_t[i][1]$ the probability of its absence.

Leveraging the discrete state space, we chose a Graph Convolutional Network (GCN)-based denoiser. At time step t, our initial node embedding is the concatenation of current adjacent edges' states and the sinusoidal positional encoding of the current time step. In order to predict the state of the edge $\{i, j\}$ at time $t - 1$, a GCN computes new embeddings h_i and h_j for nodes i and j, then the concatenation of h_i and h_j is fed to a Feedforward network to predict the categorical distribution of the edge $\{i, j\}$. As a result, we treat the denoising process as a classification task over edges of the underlying graph.

Graph Convolutional Network (GCN). Graph Convolutional Networks (GCNs) extend traditional convolutional neural networks to non-Euclidean domains, such as graphs, enabling the learning of node representations by aggregating features from local neighborhoods. The propagation rule for a single GCN layer, as introduced by Kipf and Welling [5], is given by

$$H^{l+1} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{6}$$

where $\tilde{A} = A + I$ is the adjacency matrix with added self-loops, $\tilde{D}$ is the diagonal degree matrix of $\tilde{A}$, $H^{(l)}$ represents the node feature matrix at layer l, $W^{(l)}$ is the learnable weight matrix, and $\sigma(\cdot)$ is a non-linear activation function, typically ReLU. Its node-wise formulation is given by:

$$x_i^{l+1} = \Theta^T \sum_{j\in\mathcal{N}(i)\cup\{i\}} \frac{e_{j,i}}{\sqrt{d_i^- d_j^-}} x_j^l \tag{7}$$

with d_i^- the in-degree of the i-th node. This formulation ensures that features are normalized by node degrees, facilitating stable training and mitigating issues arising from varying neighborhood sizes.

3 Autoregressive Method

As the adaptions of classic methods showed no convincing results in the conducted experiments, we opted for an autoregressive approach. This consists of slower and smoother noising and denoising processes, allowing for more confidence in past predictions.

3.1 Noising

We adapt the discrete framework by fixing a schedule over nodes, then we subdivide each time step of the D3PM noising process into as much node steps as the number of nodes in the graph. Formally, at time step t and node step k,

$$x_{t,k}[i] = \begin{cases} y_t[i] \text{ if the } i\text{-th edge is adjacent to any node with index lower than } k \text{ in the schedule} \\ y_{t-1}[i] \text{ else} \end{cases} \tag{8}$$

where $x_{t,k}[i]$ denotes the categorical distribution of the i-th edge at time step t and node step k and $y_t[i]$ denotes the categorical distribution of the i-th edge at time step t in the classic D3PM noising process.

3.2 Denoising

We employ the same model than in the classic D3PM approach as a denoiser. At time t and scheduled node k, the l-th scheduled node embedding consists in the concatenation of the following features:

- Whether $l \leq k$, one-hot encoded.
- The edge state of the l-th node's adjacent edges.
- The sinusoidal embedding corresponding to the $(t-1)$-th time step if $l > k$, to the t-th timestep if $l \leq k$.
- l modulo $2 \times L$ where L is the depth of the graph convolution, one-hot encoded.

Although the model already receives the current time step information through the third feature, this information does not capture the noise level across all nodes, as the time step now varies across different parts of the graph. To compensate for this, we incorporate additional features into the embedding, ensuring the model can better interpret the local progression of the denoising process. While the node's relative position within the schedule is implicitly contained in the third feature, we observed that explicitly providing this information via the first feature significantly improved model performance. Furthermore, we hypothesize that the poor performance of the classic D3PM framework is partly due to the absence of sufficiently rich positional information. To address this, we leverage the node schedule to introduce a localized positional encoding within the node embeddings, incorporated as the fourth feature.

3.3 Sampling

We adapt the sampling process from [2] to suit our method in Algorithm 1.

3.4 Variant

While Algorithm 1 enables corrections over time by iterating through the entire graph (following the node schedule) at each time step, we also investigate an alternative approach that prioritizes short-term confidence in past predictions by swapping the loop order in Algorithm 2. This adjustment ensures that the graph is fully denoised with respect to the previously scheduled nodes before proceeding further in the process.

4 Experiments

The code of the experiments can be found here .

Algorithm 1 Autoregressive sampling process

$\forall i\ x_{T,N}[i] \sim \mathcal{B}ern(\frac{1}{2})$
for $t = T, \ldots, 1$ **do** ▷ T the chosen number of time steps
  for $n = N, \ldots, 1$ **do** ▷ N the number of nodes in the graph
    Compute $p_\theta(x_{t,n-1} \mid x_{t,n})$
    $x_{t,n-1} \sim p_\theta(x_{t,n-1} \mid x_{t,n})$
  end for
  if $t > 1$ **then**
    Compute $p_\theta(x_{t-1,N} \mid x_{t,1})$
    $x_{t-1,N} \sim p_\theta(x_{t-1,N} \mid x_{t,1})$
  end if
end for

Algorithm 2 Autoregressive sampling process (Variant)

$\forall i\ x_{T,N}[i] \sim \mathcal{B}ern(\frac{1}{2})$
for $n = N, \ldots, 1$ **do**
  for $t = T, \ldots, 1$ **do**
    Compute $p_\theta(x_{t-1,n} \mid x_{t,n})$
    $x_{t-1,n} \sim p_\theta(x_{t-1,n} \mid x_{t,n})$
  end for
  if $n > 1$ **then**
    Compute $p_\theta(x_{T,n-1} \mid x_{1,n})$
    $x_{T,n-1} \sim p_\theta(x_{T,n-1} \mid x_{1,n})$
  end if
end for

Data. We trained the denoising model using our autoregressive method on a dataset of 10^4 spanning trees of a 32×32 grid graph. These spanning trees were uniformly sampled using Wilson's algorithm. The dataset generation and preprocessing required approximately one hour on a standard CPU, and the total storage space needed to store it was around 3.1 GB.

Chosen node schedule. The fixed node update schedule is illustrated in Figure 1 on the 9×9 grid graph. This schedule was consistently applied across all considered grid graphs, with appropriate resizing to match different dimensions.

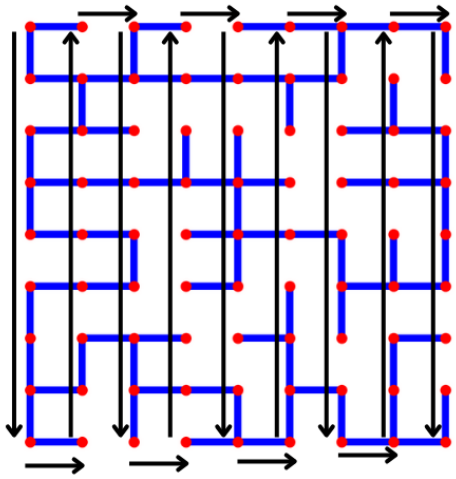

Figure 1: Chosen node schedule on the 9×9 grid graph

	Autorgressive method	Variant
Final loss	0.50	**0.21**
Final accuracy	0.71	**0.88**

Figure 2: Training results

Denoiser architecture. The denoiser consists of three main components:

- A three-layer feedforward network for initial feature extraction,
- A four-layer GCN for structured information propagation,
- A final three-layer feedforward network for prediction.

Training setup. The training process was conducted on CLEPS, INRIA's computing cluster, using a single GPU. The model was trained for 100 epochs with a batch size of 2, employing the Adam optimizer with an initial learning rate of 0.001. A Reduce-on-Plateau scheduler was used, leading to a slight improvement in final accuracy. We employ the Binary Cross Entropy Loss (BCELoss) as the objective function. The prediction accuracy is computed by assigning a score of 1 for each correctly classified edge—either when the model predicts a higher probability for a present edge or a lower probability for an absent edge. The final accuracy is obtained by dividing the total number of correct predictions by the total number of edges in the graph.

We set the number of time steps to 10, as increasing this parameter did not yield noticeable performance improvements. The sinusoidal embedding dimension was kept at the standard value of 100. The entire training process took approximately one hour per model. The final losses and accuracies are presented in Figure 2.

Results. We evaluated the sampling method on 32×32, 48×48, and 64×64 grid graphs, using three key criteria to assess the quality of the generated spanning trees:

- Number of non-trivial connected components: This metric quantifies how well the model maintains connectivity, as generated samples often fail to be fully connected.
- Proportion of tree-structured components: Among the non-trivial connected components, we measure the fraction that are valid trees, assessing the model's ability to enforce acyclicity within connected components.
- Number of square cycles: As a simple proxy for cycle formation, we count the number of four-node square cycles in the generated graphs, which provides insight into the model's effectiveness at avoiding spurious loops.

To ensure a fair comparison, we computed the average values of these criteria over 10 sampled spanning trees for each grid size. The results are presented in Figure 3, Figure 4, and Figure 5, corresponding to the 32×32, 48×48, and 64×64 grid graphs, respectively.

Performance comparisons suggest that the autoregressive method generally outperforms the variant across the selected evaluation criteria, despite exhibiting lower accuracy during training. A qualitative visual analysis, illustrated in Figure 6 and Figure 7, reveals notable structural differences between the generated samples. In particular, samples produced by the autoregressive method deviate from a uniform distribution, frequently exhibiting long, straight-line patterns that would be unlikely in a truly uniform sampling process. Conversely, while the variant method generates samples that visually resemble a uniform distribution more closely, it performs worse statistically in terms of producing valid spanning trees.

Despite its theoretical advantage—with a complexity of $\mathcal{O}(n^2)$, which improves upon Wilson's algorithm's worst-case complexity of $\mathcal{O}(n^3)$—our implementation remains slower in practice. It takes 4 minutes and 58 seconds to sample 10 spanning trees on a 32×32 grid, whereas Wilson's algorithm generateq 10^4 trees (on average) in the same time. This highlights a critical limitation: the computational cost of model evaluation and probabilistic edge sampling in our implementation outweighs its theoretical benefits, making it impractical for large-scale sampling.

	Autoregressive method	Variant
Number of connected components	**28.80**	32.90
Ratio of trees	**0.95**	0.77
Number of squares	**11.00**	21.70

Figure 3: Average criteria value comparison of the autoregressive method and its variant on the 32×32 grid graph.

	Autoregressive method	Variant
Number of connected components	69.40	**66.20**
Ratio of trees	**0.97**	0.82
Number of squares	**29.70**	45.50

Figure 4: Average criteria value comparison of the autoregressive method and its variant on the 48×48 grid graph.

	Autoregressive method	Variant
Number of connected components	**121.90**	124.00
Ratio of trees	**0.98**	0.85
Number of squares	**51.00**	79.00

Figure 5: Average criteria value comparison of the autoregressive method and its variant on the 64×64 grid graph.

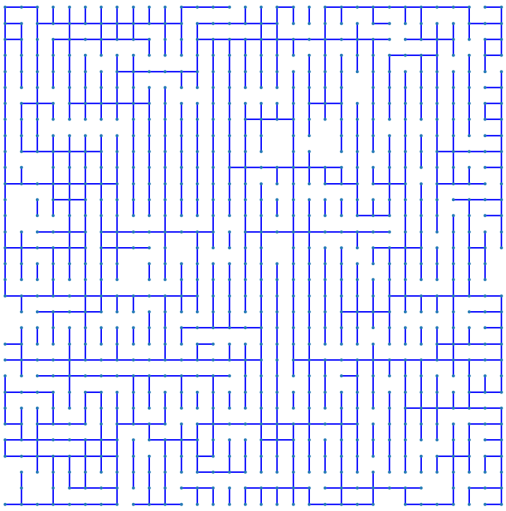

Figure 6: Autoregressive method sample

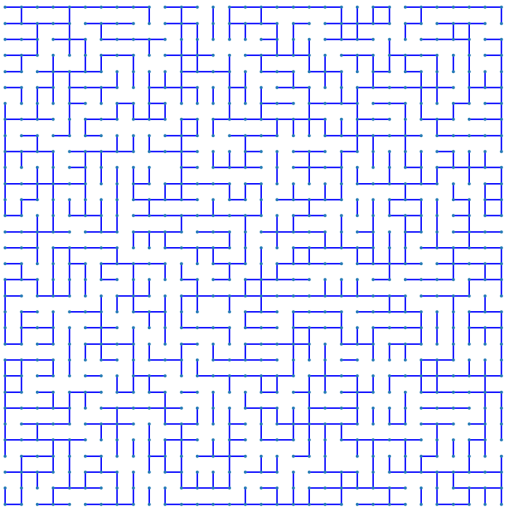

Figure 7: Variant sample

Acknowledgements

I would like to thank Marc Lelarge, my supervisor, for his guidance and support throughout this project, and for his extensive and valuable feedback on this report.

References

[1] Jacob Austin et al. "Structured Denoising Diffusion Models in Discrete State-Spaces". In: *Advances in Neural Information Processing Systems*. 2021. 1

[2] Kilian Konstantin Haefeli et al. "Diffusion Models for Graphs Benefit From Discrete State Spaces". In: *The First Learning on Graphs Conference*. 2022. URL: https://openreview.net/forum?id=CtsKBwhTMKg. 1, 4

[3] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising Diffusion Probabilistic Models". In: *Advances in Neural Information Processing Systems* (2020). 1

[4] Emiel Hoogeboom et al. "Argmax Flows and Multinomial Diffusion: Towards Non-Autoregressive Language Models". In: *Advances in Neural Information Processing Systems*. 2021. 1

[5] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2017. URL: https://arxiv.org/abs/1609.02907. 3

[6] Zhifeng Kong et al. "DiffWave: A Versatile Diffusion Model for Audio Synthesis". In: *International Conference on Learning Representations*. 2021. 1

[7] Ashish Vaswani et al. "Attention is All You Need". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017, pp. 5998–6008. URL: https://arxiv.org/abs/1706.03762. 2

[8] David Bruce Wilson. "Generating random spanning trees more quickly than the cover time". In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of Computing*. 1996. 1