Neural Graph Generation with Conditioning

ALTEGRAD 2024-2025 - Data Challenge

Bryan Chen
École Polytechnique, Palaiseau, France
bryan.chen@polytechnique.edu

Lorenzo Dunau
ENS Paris-Saclay, Gif-sur-Yvette, France
lorenzo.dunau@ens-paris-saclay.fr

Martin Jolif
ENS Paris-Saclay, Gif-sur-Yvette, France
martin.jolif@ens-paris-saclay.fr

Abstract

Graph generation has emerged as a fundamental task in machine learning, with applications spanning cheminformatics, social network analysis, and transportation systems. Existing models often struggle with capturing complex structural and semantic properties, particularly when conditioned on diverse user-defined inputs like textual descriptions. In this work, we present a Neural Graph Generator (NGG) [1] leveraging variational graph autoencoders (VGAE) [2] and latent diffusion models [3] for efficient and feature-conditioned graph generation. The NGG framework encodes graphs into a compact latent space, applies diffusion processes to enable controlled generation, and decodes them into graph structures aligned with the input properties. Notably, our model extends traditional approaches by conditioning on textual embeddings, allowing for the generation of graphs tailored to user-specified characteristics. This work offers a robust pathway for synthesizing novel graphs with diverse applications in both academic and industrial domains.

1 Introduction

Graph generation is a fundamental problem in network science and machine learning, enabling the creation of data with specific topological properties tailored to various applications. The challenge addressed in this project focuses on generating graphs conditioned on textual descriptions that specify their structural properties. These properties are essential in characterizing and differentiating networks and include:

- **Number of nodes**: Determines the size of the graph and represents entities within the network.
- **Number of edges**: Captures the overall connectivity by defining the relationships between nodes.
- **Average degree**: Indicates the average connectivity of nodes, providing insights into network density.
- **Number of triangles**: Represents fundamental motifs in networks, influencing clustering and transitivity.
- **Global clustering coefficient**: Measures the likelihood that a node's neighbors are interconnected, highlighting network cohesion.
- **k-core**: Defines dense subgraphs where each node is connected to at least k other nodes, aiding in the identification of resilient regions.
- **Number of communities**: Reflects modular structures in the graph, indicating groups of densely connected nodes.

The dataset provided for this challenge serves as the foundation for training, validation, and testing. It is organized into three subsets:

- **Training set**: Contains 8,000 graph-description pairs, where graph files (in edgelist and graphml formats) are accompanied by textual descriptions specifying their properties.
- **Validation set**: Comprises 1,000 samples, similar in format to the training set, and is used for hyperparameter tuning and model evaluation during development.
- **Test set**: Consists of 1,000 textual descriptions in a single file (`test.txt`), with no associated graphs provided. This subset evaluates the generalization performance of the models on unseen data.

Each textual description provides detailed information about the structural attributes of the graph, such as its size, connectivity, clustering, and modularity. These descriptions act as conditioning inputs for the generation process, challenging the models to produce graphs that match the specified criteria. **Example:**

```
graph_7, In this graph, there are 15 nodes connected by 14 edges.  On
average, each node is connected to 1.8666666666666667 other nodes.  Within
the graph, there are 0 triangles, forming closed loops of nodes.  The
global clustering coefficient is 0.  Additionally, the graph has a maximum
k-core of 1 and a number of communities equal to 4.
```

This description specifies a graph with 15 nodes and 14 edges, characterized by low connectivity (average degree of approximately 1.87) and no clustering (as indicated by the absence of triangles and a global clustering coefficient of 0). Additionally, the graph exhibits a maximum k-core of 1, suggesting minimal local density, and is partitioned into 4 distinct communities. The task is to generate a graph matching these criteria, leveraging the textual description as input.

By addressing such examples, we aim to develop models capable of reliably translating textual descriptions into graphs, advancing both theoretical and practical aspects of conditional graph generation.

2 Our Approach

To address the challenge of generating graphs that satisfy specified structural criteria, we propose a comprehensive methodology comprising a Graph Attention Network (GAT) [4] encoder as part of a Variational Graph Autoencoder (VGAE), and a Conditional Denoising Diffusion Model.

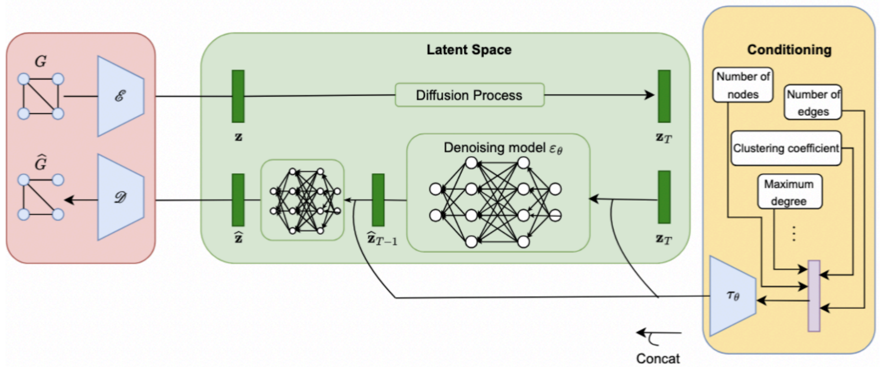

Figure 1: Overview of our approach

2.1 Prepossessing

For the conditional generation in the latent space we tried to use several conditioning vectors:

- Extraction of the properties to obtain a vector like this for example for the graph 7: $[15, 14, 1.8666666666666667, 0, 0, 1, 4]$
- A BERT [5] (Bidirectional Encoder Representations from Transformers) sentence embedding of the graph text description
- A CLIP [6] (Contrastive Language-Image Pre-Training) text embedding of the graph text description
- A concatenation of the two first embedding items (Extraction of the properties and a BERT sentence embedding)

We observed that the "naive" properties extraction enabled us to obtain a better MAE score on the test set.

2.2 Graph Attention Network (GAT) Encoder

The Graph Attention Network (GAT) encoder is a core component of our Variational Graph Autoencoder (VGAE). Its role is to learn node-level and graph-level embeddings by dynamically assigning attention weights to edges, capturing both local and global relationships in the graph structure. Experimental results showed that the GATEncoder outperformed other architectures, such as the Graph Isomorphism Network (GIN) [7] and the Chebyshev Spectral Graph Convolutional Network (ChebConv) [8], due to its ability to model weighted relationships and capture complex structural patterns effectively.

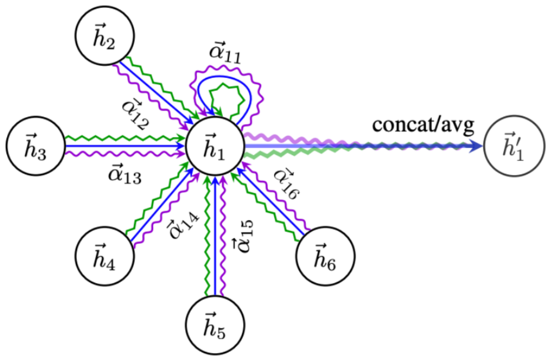

Figure 2: GAT Encoder multi-head attention = 3

Model	**MAE**
NGG [1] where VGAE with ChebEncoder	0.15287
NGG [1] where VGAE with GINEncoder	0.16894
NGG [1] where VGAE with GATEncoder	0.15163

Table 1: Mean Absolute Error (MAE) for different models with batch_size = 64, epochs_autoencoder = 400, epochs_denoise = 200, n_layers_denoise = 3, n_layers_encoder = 3, n_layers_decoder = 5 and CosineAnnealingWarmRestarts scheduler

Our GAT encoder is specifically tailored to the VGAE framework, where it is used to encode graph data into a latent space representation. It consists of:

- **Multi-head attention layers**: Each GATConv layer applies multiple attention heads, which learn distinct attention patterns and aggregate them for richer embeddings.
- **Conditioning mechanism**: A conditioning vector representing graph properties (e.g., number of nodes, clustering coefficient) is concatenated to the graph-level embeddings, enabling the encoder to take into account structural constraints.
- **Global pooling**: Node-level features are aggregated using global add pooling to produce graph-level embeddings.
- **Batch normalization and fully connected layer**: These ensure stability and map the graph-level embedding to a compact latent representation of fixed dimensionality (*latent_dim*).

While exploring alternative encoders, we tested the ChebEncoder based on Chebyshev polynomials. However, despite its theoretical ability to capture spectral graph properties, the ChebEncoder did not achieve results comparable to the GATEncoder. Thus, the GATEncoder was retained for its superior ability to model complex graph relationships.

2.3 Variational Graph Autoencoder (VGAE)

The Variational Graph Autoencoder (VGAE) is the backbone of our approach, combining the GATEncoder as its encoder component and a Decoder to reconstruct adjacency matrices. The VGAE operates by encoding input graphs into a probabilistic latent space, where each graph is represented as a distribution rather than a point. This allows for the generation of diverse and flexible graph structures. The VGAE consists of:

- **Encoder**: The GATEncoder encodes graph features and structural properties into a latent space representation.
- **Latent space sampling**: Two fully connected layers, μ (mean) and $\log \sigma^2$ (variance), map the encoded representation to a latent space. A reparameterization trick is used to sample embeddings from the distribution.
- **Decoder**: The Decoder reconstructs the adjacency matrix of the graph from the latent embedding. We explored two architectures for the decoder:
    1. **The standard Decoder**: This uses a simple fully connected architecture with Gumbel-softmax to produce binary edges. Despite its simplicity, it achieved strong reconstruction accuracy during experiments.
    2. **The improved Decoder**: This architecture incorporates a wider network for enhanced expressivity, multiple layers with `LayerNorm` and `ReLU` activations, learnable temperature parameters, and dropout regularization for better generalization. The improved decoder generates logits for adjacency matrices, enforces symmetry, and applies a straight-through estimator during training. However, despite these enhancements, it underperformed compared to the standard decoder. The increased complexity appears to introduce optimization challenges or risks of overfitting, resulting in lower reconstruction accuracy on both the training and validation sets.

The VGAE is trained using a combination of reconstruction loss (Binary Cross-Entropy) and KL divergence loss. Binary Cross-Entropy is particularly well-suited for this task due to the binary nature of adjacency matrices, where edges are represented by 1 and non-edges by 0. The probabilistic outputs of the decoder align naturally with this representation, allowing the BCE loss to effectively measure the discrepancy between the predicted edge probabilities and the true adjacency matrix. Additionally, BCE accommodates the sparsity often observed in real-world graphs, as it penalizes errors asymmetrically, focusing on correctly predicting edges, which are typically less frequent than non-edges. This choice ensures the model efficiently learns the underlying structure of graphs while maintaining robustness to imbalances in the data.

2.4 Denoising Diffusion Model

To enhance the robustness and fidelity of graph generation, we employ a Conditional Denoising Diffusion Model. This model follows a progressive noise addition and removal process, inspired by diffusion in physics. The approach is as follows:

- **Forward process**: Noise is incrementally added to the latent graph embeddings over a series of timesteps, controlled by a beta schedule, simulating a diffusion process.
- **Reverse process**: The denoising model ϵ_θ learns to remove this noise step by step, reconstructing the original embeddings from random noise.

Beta scheduling for diffusion: The beta schedule, which determines the amount of noise added at each timestep, plays a crucial role in the forward diffusion process. We experimented with four beta schedules:

- **Cosine beta schedule**: Inspired by the cosine annealing schedule, this method applies a cosine function to control the noise addition, ensuring smooth transitions. It is often effective in other diffusion-based models.
- **Linear beta schedule**: This schedule linearly increases the beta values from a small initial value ($\beta_{start} = 0.0001$) to a higher final value ($\beta_{end} = 0.02$) over the timesteps. This provides a gradual and predictable noise addition, making it easier for the model to learn the reverse process.
- **Quadratic beta schedule**: This schedule increases beta values quadratically, allowing for slower noise addition at the beginning and faster noise addition at the end.
- **Sigmoid beta schedule**: The noise addition follows a sigmoid curve, starting slow, increasing sharply in the middle, and tapering off towards the end.

Choice of linear beta schedule: Among the tested schedules, the linear beta schedule provided the best performance in our experiments. Its gradual and uniform noise addition simplifies the reverse denoising process, enabling the model to reconstruct latent embeddings with higher accuracy. Specifically, the linear schedule balances the trade-off between ease of training and effective noise distribution, allowing the model to focus on learning both global and fine-grained graph structures. The linear beta schedule is defined as:

$$\beta_t = torch.linspace(\beta_{start}, \beta_{end}, timesteps)$$

where β_{start} and β_{end} are the minimum and maximum noise levels, and *timesteps* is the total number of steps in the diffusion process.

Training the denoising model: The denoising model ϵ_θ is trained to predict and remove noise from the corrupted embeddings $z_T, z_{T-1}, \ldots, z_0$. At each timestep t, the model minimizes the discrepancy between the predicted noise and the true noise added during the forward process. We use a noise prediction loss, such as L_1 or L_2, to optimize this objective:

$$\mathcal{L}_{denoise} = loss(predicted\_noise, true\_noise)$$

The learned embeddings are then passed to the decoder to reconstruct the adjacency matrix of the graph. This approach ensures that the generated graphs adhere to the specified structural properties, even in the presence of noise.

Integration with conditioning: To align the diffusion process with graph-specific constraints, we incorporate conditioning into the model. Graph properties such as the number of nodes, edges, and clustering coefficients are processed by a conditioning network and integrated into the denoising process using several MLPs (one for the conditioning embedding, another for the positional time embedding, and a final one to process both time, conditioning and the data together). This ensures that the reconstructed graphs satisfy the desired properties while maintaining structural realism.

3 Results

The results obtained in this challenge are based on several evaluations of our Variational Graph Autoencoder (VGAE) model. The primary objective was to generate graphs that meet a specific set of statistical criteria: the number of nodes, the number of edges, average degree, the number of triangles, global clustering coefficient, maximum k-core, and the number of communities.

To assess the performance of the model, various evaluation metrics were employed. These metrics provide insight into how closely the model's predictions align with the true statistical properties of the graphs.

3.1 Evaluation Metrics

Mean Absolute Error (MAE): Unlike MSE, MAE measures the average absolute difference between the predicted and true values. This metric is less sensitive to large errors, but still provides a good indication of the overall performance of the model. MAE is often used in conjunction with MSE to provide a balanced view of performance, capturing systematic differences without overly penalizing outliers.

3.2 Main Changes in Model Parameters for better result

During the course of the challenge, several adjustments were made to the model's parameters to improve performance:

- **Dropout:** The dropout rate, initially set to 0.0, was increased to 0.1. This introduced regularization into the model, helping to prevent overfitting and improving generalization.
- **Batch size:** The batch size, which defines the number of examples processed at once, was reduced from 256 to 128. This change led to more stable convergence during training.
- **Number of epochs for the autoencoder:** The number of epochs for the autoencoder was increased from 200 to 300 (and even 9,000), allowing the model more time to learn the graph structures and improve the accuracy of its predictions.
- **Number of epochs for the diffuser:** The number of epochs for the diffuser was increased from 100 to 500. This was done to ensure the model had sufficient time to accurately learn the intricate relationships and dynamics of the data distribution. Extending the training duration allowed the diffuser to produce more robust and reliable outputs by better approximating the underlying patterns of the dataset.
- **Number of layers:** The number of layers in the encoder was increased from 2 to 3, while the number of layers in the decoder was increased from 3 to 5. Adding these layers improved the model's capacity to capture more complex graph structures.
- **Loss function:** The reconstruction loss function was changed from a squared error to a Binary Cross-Entropy (BCE) loss to better suit the discrete properties of graphs.

$$BCE(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]$$

- **Scheduler:** The learning rate scheduler, initially based on StepLR, was replaced by CosineAnnealing-WarmRestarts. This change allowed for dynamic adjustment of the learning rate, facilitating better model convergence. The formula used for the CosineAnnealingWarmRestarts scheduler is:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 + \cos\left(\frac{t - T_i}{T_i}\pi\right)\right)$$

  where η_t represents the learning rate at step t, η_{min} and η_{max} denote the minimum and maximum learning rates respectively, t is the current training step, and T_i is the length of the current period before the next restart.

4 Conclusion

In this challenge, we explored different approaches for generating graphs that adhere to specific structural properties. By leveraging Graph Attention Networks (GAT), Variational Graph Autoencoders (VGAE), and denoising models, we were able to generate graphs that closely match the target criteria. Our evaluation metrics, including MSE, MAE, and normalized error, provided insights into the strengths and limitations of each model.

References

[1] I. Evdaimon, G. Nikolentzos, C. Xypolopoulos, A. Kammoun, M. Chatzianastasis, H. Abdine, and M. Vazirgiannis, “Neural graph generator: Feature-conditioned graph generation using latent diffusion models,” 2024. arXiv preprint arXiv:2403.01535.

[2] T. N. Kipf and M. Welling, “Variational graph auto-encoders,” 2016. arXiv preprint arXiv:1611.07308.

[3] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, “High-resolution image synthesis with latent diffusion models,” 2021. arXiv preprint arXiv:2112.10752.

[4] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph attention networks,” *ICLR*, 2017. arXiv preprint arXiv:1710.10903.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” 2018. arXiv preprint arXiv:1810.04805.

[6] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, “Learning transferable visual models from natural language supervision,” 2021. arXiv preprint arXiv:2103.00020.

[7] Z. Liu, C. Rodriguez-Opazo, D. Teney, and S. Gould, “How powerful are graph neural networks?,” *ICLR*, 2018. arXiv preprint arXiv:1810.00826v3.

[8] M. Defferrard, X. Bresson, and P. Vandergheynst, “Convolutional neural networks on graphs with fast localized spectral filtering,” *NeurIPS*, 2016. arXiv preprint arXiv:1606.09375.