Altegrad Data Challenge Report

Anatole VAKILI, Julien DELAVANDE, Soël MEGDOUD

January 2025

1 Introduction

The goal of this project is to explore techniques to generate graphs with specific properties. To achieve this, 7 statistics were inserted into the text descriptions associated with graphs for the training set (number of nodes, of edges, of triangles etc.).

Our work explores several enhancements to the neural architecture, including contrastive learning techniques and, finally, a non-machine learning approach.

2 Baseline Model

The descriptions are initially not processed using language processing techniques, as the statistics are extracted directly from each description and injected into the model. We also belive that embedding the entire descriptions would not increase performance as it would lose information.

The baseline model is a latent diffusion model. An encoder encodes the graph in a denser latent space in which the diffusion process is carried out using an MLP. The MLP is trained to successively de-noise the latent representation of the noisy graph to which the 7 concatenated statistics are added.

The first step was to determine the maximum performance of this architecture. We ran a grid-search for several hyper-parameters: Hidden dimension size for the encoder/decoder, learning rate, dropout, dimensionality of conditioning vectors for generation etc. It turns out that these changes in hyper-parameters had very little effect on the MAE calculated on the Kaggle leaderboard. The base score of 0.917 could only be improved to 0.889, which is not conclusive. It is interesting to note that this better configuration includes a dropout (0.1) and an encoding dimension of 128.

3 Contrastive Learning Improvement

3.1 Overview

Contrastive learning [1] aims to learn an embedding space where "positive" samples are pulled together and "negative" samples are pushed apart. In *graph contrastive learning*, we treat augmented views of the same graph as positives, while different graphs in the batch act as negatives. Our model also incorporates a *text-derived* 7D condition vector $\mathbf{c}$ for each graph G. We align the graph embedding z with c_{emb} via an InfoNCE-like loss:

$$\mathcal{L}_{\text{contrastive}} = -\sum_{i=1}^{B} \log \frac{\exp\big(\text{sim}(z_i, c_{\text{emb},i})/\tau\big)}{\sum_{j=1}^{B} \exp\big(\text{sim}(z_i, c_{\text{emb},j})/\tau\big)},$$

added to the VAE's reconstruction–KLD loss:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{VAE}} + \lambda\,\mathcal{L}_{\text{contrastive}}.$$

3.2 Methods and Experiments

Graph–Text Contrastive. We use the classic VGAE and introduce a *graph–text* contrastive term. Each mini-batch provides *positive pairs* $(z_i, c_{\text{emb},i})$ for the same sample (graph embedding vs. text embedding) and *negative pairs* $(z_i, c_{\text{emb},j})$ for $j \neq i$. This InfoNCE-style objective is then added to the usual loss. We evaluated various configurations (batch size, dropout) using our MAE calculation.

Table 1: Results of contrastive approaches.

Values of Lambda	Batch Size	local MAE
0.1	256	1.0721
0.5	256	0.9918
0.5	512+dropout	**0.9881**
10	512+dropout	0.9927
Baseline model		1.0224

Graph–Graph Contrastive. To further enhance robustness, we introduced a *graph–graph* contrast. For each original graph, we create two *augmented views* (slight edge/node-feature perturbations), encode them as z_1 and z_2, then project these to z'_1 and z'_2 via small MLPs. The *positive pairs* are (z'_1, z'_2) from the same graph, and *negative pairs* are (z'_1, z'_j) for $j \neq 2$ in the batch. Pulling positive pairs together while pushing negative pairs apart (again via InfoNCE) is intended to yield a more robust latent space. However, our final MAE do not improve that much 0.9852 (with $\lambda_{\text{g2t}} = \lambda_{\text{g2g}} = 0.5$), suggesting that further domain-specific tuning may be needed. These augmentations ensured that the two views retained the underlying semantic meaning of the original graph while introducing slight structural and feature-based perturbations. The total loss is:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{VAE}} + \lambda_{\text{g2t}}\mathcal{L}_{\text{contrastive,g2t}} + \lambda_{\text{g2g}}\mathcal{L}_{\text{contrastive,g2g}}.$$

3.3 Conclusion

Overall, our MAE did not drop below 0.9852. We suspect insufficient negative diversity, and potentially suboptimal augmentations or hyperparameters. Further tuning of domain-specific augmentations or temperature and weight parameters may be required.

4 Improving Graph Diffusion Architecture Inspired by Image Generation Techniques

The initial diffusion model encoded graphs into a latent space. The diffusion process was then carried out using a Multilayer Perceptron (MLP) that operated in the latent space. Information from graph descriptions was concatenated with the graph's latent representation to guide the diffusion process.

While this approach was functional, it exhibited several limitations. The MLP-based diffusion process may be to simple for the denoising task and the concatenation of descriptive statistics with the latent vector did not provide a robust mechanism to steer the generation process in a controlled manner.

In this section, we present the architectural improvements of the diffusion model, inspired by methods used for image generation guided by prompts.

4.1 Proposed Architecture Inspired from Image Generation

Recent advancements in image generation [2] employ a U-Net architecture with cross-attention to effectively guide the generation process using prompts.

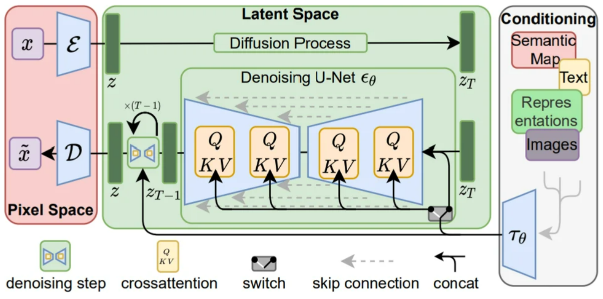

Figure 1: Latent Diffusion Model with U-net for Image Generation.

Inspired by these techniques, we redesigned the diffusion model:

- **U-Net-Based Diffusion:** Replace the MLP with a U-Net architecture to improve the model's capacity to process and refine graph representations. The U-Net consists of an encoder-decoder structure with skip connections that allow the model to preserve important graph features during the refinement process.
- **Cross-Attention Mechanism:** At each step of the diffusion process, descriptive statistics from the graph descriptions are incorporated using a cross-attention mechanism. This ensures that the generation process remains aligned with the specified graph properties.

The U-Net architecture consists of an encoder-decoder design with n blocks. Each bloc includes fully connected layers, RELU activations, BatchNorm1d and cross-attention layers. Skip connections preserve important features during encoding and decoding.

To optimize the performance of the U-Net architecture, we conducted experiments by varying two key parameters: the number of blocks in the U-Net, to change the depth of the model, and the dimension of the statistics projections.

4.2 Conclusion

We evaluated different configurations with MAE of the 7 features extracted from the graphs generated by the model on the val set. The best model showcased 0.8890 on the kaggle leaderboard.

Table 2: Performance of U-Net with varying hyperparameters.

Number of Blocks	Stat. Embedding Dimension	MAE
1	7	0.9901
1	32	**0.9890**
3	7	0.9945
3	32	0.9923
Baseline model		1.0224

The results show that the use of a U-net barely improves on the results obtained with a simple basic MLP, and there are several possible reasons for this. The size of the statistics encoding projection does not really have an impact on the results, perhaps because projecting statistics into a larger space does not add any information. If the task had been to generate graphs from written sentences rather than from extracted statistics, the attention mechanism would have made more sense because it would have been able to link certain parts of the sentence to certain nodes in the graph. Unfortunately, the statistics given are aggregated data at the global level of the graph, which limits the local use of attention.

It also seems that the smaller the size of the U-net, the better the results, which justifies the use of a simple MLP for this task. In addition, the U-net architecture may be better suited to processing images at several scales, but does not necessarily make sense for a graph. Unfortunately, it can be said that using a U-net with cross-attention as a diffusion model does not significantly improve performance and is therefore better than a simple MLP for this task.

5 Property-based Loss Regularization

5.1 Motivation

Our initial variational autoencoder (VAE) used a standard reconstruction + KL divergence (KLD) loss. To better preserve graph-level properties such as edge count, average degree, and clustering coefficient, we introduced an additional *property-based loss* term.

5.2 Implementation

The `VariationalAutoEncoder` reconstructs an adjacency matrix $\hat{A}$ from an input graph G with adjacency A. The losses are defined as:

$$\mathcal{L}_{\text{recon}}(A, \hat{A}) = \|A - \hat{A}\|_1, \quad \mathcal{L}_{\text{KLD}}(\mu, \log\sigma^2) = -\tfrac{1}{2}\sum_i \big(1 + \log\sigma_i^2 - \mu_i^2 - \exp(\log\sigma_i^2)\big),$$

$$\mathcal{L}_{\text{prop}}(\widehat{p}, p) = \left\| \tfrac{\widehat{p}-\mu}{\sigma} - \tfrac{p-\mu}{\sigma} \right\|_1,$$

where $\widehat{p}$ and p represent predicted and ground-truth graph properties, normalized by mean μ and standard deviation σ. The total loss is:

$$\mathcal{L}_{\text{new}} = \mathcal{L}_{\text{recon}} + \beta\,\mathcal{L}_{\text{KLD}} + \alpha\,\mathcal{L}_{\text{prop}}.$$

The property-based loss approximates seven global graph statistics $\widehat{p}$ from the (soft) adjacency matrix $\hat{A}$, designed to be fast during training and as differentiable as possible, with smooth approximations used for non-differentiable operations. The node count is approximated by summing smooth indicators of non-empty rows in $\hat{A}$. Edges and average degree are derived from the sum of upper triangular entries, while triangles use the trace of $\hat{A}^3$ (divided by six). The global clustering coefficient scales triangles by connected triplets. The approximate max k-core value is computed as the smoothed maximum row sum, and the number of communities is estimated by counting eigenvalues of the Laplacian $L = D - \hat{A}$ below a soft threshold. These features form a vector compared against ground-truth properties in the loss.

5.3 Experiments and Results

We tested $\alpha \in [0.1, 1000]$ with fixed $\beta = 0.05$, evaluating the mean absolute error (MAE):

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left| \tfrac{\widehat{p}_i-\mu_i}{\sigma_i} - \tfrac{p_i-\mu_i}{\sigma_i} \right|.$$

Here $\widehat{p}$ is calculated using algorithms with no approximation. Importantly, the property-based loss was applied only during the training of the autoencoder. The MAE reported here corresponds to the fully trained diffusion model, where the autoencoder weights were frozen after being trained with this new loss.

Table 3: MAE results for different α values.

α	**MAE**
0 (Baseline)	1.0224
0.1	0.9686
1	0.6047
10	0.4062
100	**0.3876**
1000	0.4739

The best result was achieved with $\alpha = 100$, yielding an MAE of 0.3876. On the Kaggle test set, the model scored 0.26, though scaling differences might explain part of the improvement.

Conclusion and Limitations. The property-based loss improved the model, achieving the best diffusion-based performance. However, reliance on approximations and scaling inconsistencies across datasets remain challenges. Future work could refine property calculations, enhance normalization, and explore dynamic loss weighting or integration into the diffusion process.

6 Algorithmic Graph Generation Approach

6.1 Description of the Algorithm

To address the problem of generating graphs with precise global properties, we developed an algorithmic approach that builds graphs directly from a set of seven descriptive features: number of nodes, edges, triangles, clustering coefficient, average degree, maximum k-core, and number of communities. The methodology can be summarized in the following steps:

- **Initialization with the Stochastic Block Model (SBM):** The algorithm begins by creating an initial graph using the Stochastic Block Model [3]. Nodes are grouped into communities of similar size, and intra-community edge probabilities are adjusted to align with the clustering coefficient. This initialization provides a structured starting point that captures the desired community structure.

- **Edge Adjustment:** The total number of edges is fine-tuned to match the specified number. If the graph contains too many edges, random edges are removed. If it has too few, random edges are added while ensuring they respect the community structure.
- **Triangle and Clustering Optimization:** A rewiring mechanism adjusts the graph's local structure to align with the desired number of triangles and clustering coefficient. This step involves modifying edges to create or remove triangles while maintaining the overall edge count. It also targets the global clustering coefficient by promoting or reducing local density as needed.
- **K-Core Adjustment:** To satisfy the maximum k-core property, additional edges are strategically added around nodes with high degrees. This ensures the presence of subgraphs where all nodes have at least a specified degree.
- **Final Edge Fine-Tuning** The total number of edges is re-checked to ensure consistency with the target value after other adjustments. Random edges are added or removed to correct any deviations caused by earlier steps.
- **Output:** The resulting graph is converted into an adjacency matrix, representing the final structure that satisfies the specified global properties as closely as possible.

6.2 Performance and Results

This algorithmic approach yielded the best-performing model in our experiments, achieving a Mean Absolute Error (MAE) of **0.2643** on the loss metric and **0.1406** on the Kaggle leaderboard. These results outperform all other tested methods, including neural architectures and property-based loss enhancements.

The key to this success lies in the algorithm's explicit alignment with the problem's requirements. Unlike neural models, which rely on learning from examples and are prone to overfitting or misalignment, the algorithm explicitly targets the desired graph properties, ensuring accuracy and interpretability.

6.3 Conclusion

This algorithmic graph generation method demonstrates that, for problems with clearly defined numerical constraints, deterministic approaches can outperform learned models by directly addressing the target properties. While neural models offer flexibility and scalability for more complex tasks, algorithmic methods remain a powerful tool when the problem is well-constrained and interpretable. Future work could involve integrating this algorithm into a hybrid system, combining its deterministic precision with the adaptability of machine learning approaches.

7 Conclusion

In this project, we explored various approaches to generating graphs with specific properties, combining neural architectures and algorithmic techniques. Despite efforts to enhance model performance through contrastive learning, property-based loss regularization, and advanced diffusion architectures, these methods provided limited improvements over baseline results. However, our algorithmic approach demonstrated very good accuracy, achieving the best performance with an MAE of 0.2643 and a Kaggle leaderboard score of 0.1406.

This highlights the importance of aligning methods with problem-specific constraints: while neural networks offer flexibility, deterministic algorithms excel when precise properties are required. Future work could focus on integrating these complementary approaches to balance interpretability, scalability, and accuracy for more complex graph generation tasks.

References

[1] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. *CoRR*, abs/2002.05709, 2020.

[2] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. *CoRR*, abs/2112.10752, 2021.

[3] Elchanan Mossel, Joe Neeman, and Allan Sly. Stochastic block models and reconstruction, 2012.