Evolutionary Refinement of Generative Graph Topologies: A Hybrid WGAN-GA Approach

James Sargant
Computer Science
Brock University, Canada
js17sy@brocku.ca

Seyedeh Ava Razi Razavi
Computer Science
Brock University, Canada
ua23vc@brocku.ca

Renata Dividino
Computer Science
Brock University, Canada
rdividino@brocku.ca

Sheridan Houghten
Computer Science
Brock University, Canada
shoughten@brocku.ca

***Abstract*—Generating realistic graph-structured data is challenging due to discrete connectivity, varying graph sizes, and class-specific structural patterns. Recent Generative Adversarial Networks (GAN)-based graph generation methods improve edge modelling by learning connectivity and matching class-specific density distributions. However these models still exhibit noticeable deviations such as in degree and spectral distribution when compared to real graphs, indicating that important structural properties are not fully preserved. This work aims to reduce these deviations by refining the graphs produced by an existing GAN-based graph generator framework with a Genetic Algorithm (GA). In the GAN framework, the generator produces both node features and connectivity patterns, while a GNN-based critic evaluates graph realism and class consistency to ensure global structural and class alignment. Building on this foundation, we apply a GA to refine the edges of generated graphs. The refinement process guides synthetic graphs toward closer agreement with real data, while preserving diversity and novelty. Experimental results show that the GA refinement consistently lowers combined Maximum Mean Discrepancy (MMD) compared to the base model, leading to graphs that more closely match real structural patterns. This demonstrates that evolutionary refinement is an effective and flexible way to correct residual structural deviations in GAN-based graph generators, improving their suitability for realistic graph synthesis and data augmentation.**

I. Introduction

Graph-structured data arises in many application areas, including social networks, molecular graphs, biological systems, and communication networks, where nodes and edges capture important information about underlying processes and system properties. The generation of realistic synthetic graphs has become increasingly relevant for applications such as privacy-preserving data sharing and data augmentation for graph-based machine learning models. However, generating graphs introduces fundamental challenges that differ from those of traditional data synthesis in Euclidean spaces. Unlike images or text, which have regular grid structures and fixed dimensionality, graphs exhibit irregular topologies, variable sizes, and no canonical node ordering. As a result, effective graph generation methods must learn complex dependencies between graph structure and node attributes while preserving the structural properties that define different graph types.

Early deep learning approaches to graph generation have explored various strategies to address these challenges. Methods such as DeepGMG [7] and GraphRNN [14] model graphs through sequential generation processes, while GraphVAE [11] leverages variational autoencoders to learn latent graph representations. Generative Adversarial Networks (GANs) have shown particular promise, with GraphGAN [12], EGraphGAN [13], and MolGAN [3] demonstrating the potential of adversarial training for graph synthesis. LGGAN [5] further advanced the field by introducing labeled graph generation with comprehensive evaluation metrics including Maximum Mean Discrepancy (MMD) over degree, clustering, and orbit statistics. However, these GAN-based approaches often suffer from training instability, mode collapse, and difficulties in generating graphs with variable sizes.

The introduction of Wasserstein GANs (WGANs) [1] marked a significant advancement in addressing the instability issues inherent in standard GANs. WGAN-GP [6] further improved upon this by enforcing the Lipschitz constraint through gradient penalty rather than weight clipping, enabling more effective training and better sample diversity. Recent work has successfully adapted the WGAN framework to graph generation by combining it with Graph Neural Networks (GNNs) as critics. In [9], [10], a generator produces both node features and connectivity patterns, while a GNN-based critic evaluates graph realism and class consistency. These methods demonstrated superior performance in generating class-conditional graphs with improved training stability. However, despite these advances, these works show that these models still exhibit noticeable deviations such as in the degree spectral distribution when compared to real graphs. This limitation results in graphs with unrealistic connectivity patterns that fail to capture the complex structural dependencies between nodes and do not reflect the learned node feature distributions.

We address these core limitations by extending existing graph generation approaches with a Genetic Algorithm (GA)–based refinement stage. After the graph generator is fully trained, we apply a GA to refine the generated graphs through crossover and mutation operations that directly alter graph structure, such as edge presence and local connectivity patterns. The refinement process is guided by fitness measures derived from structural statistics of real graphs, with a particular focus on correcting deviations in degree, clustering coefficient and spectral distributions. Central to our refinement strategy is the concept of evolutionary edge editing, where graph topologies are iteratively optimized through a sequence of discrete structural modifications. This approach builds upon

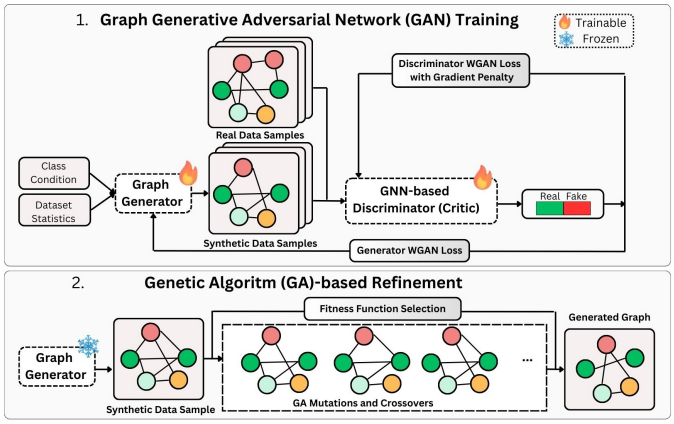

Fig. 1. Model overview. Phase 1: GAN training (based on [9], [10]). The generator learns to place nodes in a latent space where closer nodes are more likely to be connected. A GNN-based critic processes graphs using several convolution layers, pools node features, and combines them with class embeddings to compute Wasserstein scores. Phase 2: Genetic Algorithm (GA) refinement. After training, the generator produces synthetic graphs that are then refined using a GA that produces graphs closely resembling real samples. The fitness function encourages the generated graphs to match the real data distribution and target class while remaining distinct and novel.

established generative representations that utilize elementary operations, such as adding, removing, or toggling edges, to evolve networks toward specific structural or functional targets [2], [4]. By leveraging these direct edge manipulations, the evolutionary stage can enforce precise local constraints and correct statistical deviations that are often smoothed over by the continuous approximations inherent in deep generative models.

Experiments were conducted on three bioinformatics graph benchmark datasets from [8]. The results demonstrate that the proposed refinement strategy improves the alignment between generated graphs and the statistical properties of the training data. The refined graphs show more coherent and realistic structural patterns, with connectivity that better reflects the relationships encoded in node features. Overall, the GA refinement reduces structural mismatches while maintaining variability, leading to higher-quality synthetic graphs suitable for downstream graph learning tasks.

Data and Code: These are both available at: `github.com/shorinbonsai/WGAN-GA-Refine`.

II. Methodology: Evolutionary Refinement

Our framework uses the output of a deep generative model as input to evolutionary refinement to produce precise graph structures. These stages are shown in Figure 1: (1) a coarse generation stage using a Wasserstein Generative Adversarial Network (WGAN) (based on [9], [10]) and (2) a refinement stage using a Genetic Algorithm (GA) library for edge editing developed in Rust. We note that the focus of this paper is the evolutionary refinement stage, and that the coarse generation stage could use any method that produces graph structures. Although this method can be applied to extend existing GAN-based graph generation models, we use WGAN as a representative example due to its strong reported performance.

A. Coarse Generation: WGAN

WGAN generates coarse graph candidates that capture the global structural characteristics of the target graph distribution. Graphs are represented as adjacency matrices, with the generator mapping latent samples to continuous adjacency estimates. The critic is trained using the Wasserstein distance with a Lipschitz constraint, which provides a smoother and more stable training signal than standard GANs.

The primary role of the WGAN in our system is not to produce perfectly valid graphs, but to capture high-level statistical regularities observed in the training data, such as edge density, degree patterns, and broad spectral or clustering properties. These coarse outputs serve as informed initializations for the evolutionary stage. By starting the GA-based refinement from WGAN-generated candidates instead of random graphs, the search space explored during refinement is biased toward structurally plausible regions, improving convergence speed and solution quality. As a result, the WGAN provides a global initialization of graph structure, while detailed refinement and structural optimization are handled by the evolutionary stage. The WGAN implementation is based on the code from [10].

B. Fine-Tuning: Evolutionary Refinement

While GAN-based models capture global structural trends, they often struggle to satisfy precise local constraints, such as exact degree distributions or triangle counts. To address this limitation, we refine the generated graphs using a GA.

In this work, we provide a Rust-based GA library that integrates WGAN-based graph generation with a GA refinement module. The Python environment interfaces with our library to pass graphs generated by the WGAN directly to the GA initialization stage. The GA then iteratively refines graph structures by optimizing a fitness function defined over selected structural metrics (e.g. clustering coefficient, spectral gap), that are also reflected in the WGAN training objective.

C. Representation and Initialization

To effectively bridge the continuous latent space of the WGAN and the discrete search space of the graph refiner, we utilize a dual-representation strategy. This approach distinguishes between the *phenotype* (the actual graph structure) and the *genotype* (the evolutionary instruction set).

1) Graph Phenotype: The phenotype represents the physical graph structure utilized for fitness evaluation. It is implemented in the GA library as an adjacency list.

2) Command String Genotype: Our system employs a linear command-based genotype. Each individual in the population is defined by a genome g, a sequence of genes that each encode a specific operation. The final graph is produced by an expression function Φ:

$$G_{final} = \Phi(G_{base}, g) = op_n(\dots op_2(op_1(G_{base}))\dots) \quad (1)$$

where G_{base} is the coarse graph generated by the WGAN. This ensures that the GA explores the local neighbourhood of the coarse graph, rather than the entire search space. The evolutionary search progresses through the manipulation of the linear command genotype.

D. Genetic Operators and Dynamics

Crucially, our operators do not directly manipulate edges on the adjacency matrix; instead, they manipulate the *instruction sequence* that constructs the graph. In particular, the chromosome is a sequence of edge-editing operations that is applied sequentially; it is thus deterministic and specifies a particular network. The length of the chromosome is twice the number of vertices, to stay relatively consistent with [4] while scaling for graphs with different cardinalities.

1) Edge Editing Operations: The operations are applied to a graph $G(V, E)$ with nodes u, v, w, and x from V, as follows.

- **Toggle(u, v):** If edge $\{u, v\}$ is in E then remove it from E, else add it to E. ***Local* Toggle**(u, w, v) requires that edges $\{u, w\}$ and $\{w, v\}$ exist then calls **Toggle(u, v)**.
- **Hop(u, v, w):** If edges $\{u, v\}$ and $\{v, w\}$ are in E and edge $\{u, w\}$ is not, then remove $\{u, v\}$ from E and add $\{u, w\}$ to E.
- **Add(u, v):** If $\{u, v\}$ is not in E then add it to E, else do nothing. ***Local* Add(u, w, v)** requires that edges $\{u, w\}$ and $\{w, v\}$ exist then calls **Add**(u, v).
- **Delete(u, v):** If $\{u, v\}$ is in E then remove it from E, else do nothing. ***Local* Delete(u, w, v)** requires that edges $\{u, w\}$ and $\{w, v\}$ exist then calls **Delete**(u, v).
- **Swap(u, v, w, x):** If $\{u, v\}$ and $\{w, x\}$ are the only edges in E between nodes u, v, w and x then remove them from E and add $\{u, x\}$ and $\{v, w\}$ to E.
- **Null():** Do nothing.

These are visualized in a sequential manner in Figure 2. The probabilities of the operations are given in Table II. In initial tuning, the probabilities of Toggle, Add, Delete, and Local toggle were all set to half those of the other operations because otherwise the degree distribution was negatively affected.

2) Crossover: We employ Two-Point Crossover to recombine successful edit strategies from the population. This method is preferred over Uniform Crossover for this application as it preserves contiguous subsequences of graph operations. Since the phenotype is constructed by applying commands sequentially, preserving these functional blocks allows offspring to inherit effective macro-actions (e.g., a specific sequence of edits that creates a valid substructure) without disrupting the logic of the generative process.

3) Mutation: Mutation randomly selects 1–4 genes and replaces those with new random genes. This slightly disruptive operation promotes a deeper exploration of the search space.

E. Fitness Function

The fitness function quantifies the structural deviation between a candidate graph G and the distribution of the target graphs. Since the goal is to generate graphs that statistically resemble a class of real-world networks rather than reproducing a single instance, we employ MMD.

To compute the final score, we treat the features of the generated graph as a sample and compare them to the distribution of features in the training set. The total fitness F (where lower is better) is calculated as a weighted sum of the MMD scores combined with a penalty for deviation in graph density:

$$F = (w_d \cdot \text{MMD}_d) + (w_c \cdot \text{MMD}_c) + (w_s \cdot \text{MMD}_s) + P_{\text{edge}} \quad (2)$$

where w_d, w_c, and w_s are configurable weights for the degree, clustering, and spectral components respectively, chosen to be consistent with [10]. The edge penalty term P_{edge} ensures the graph maintains a realistic density and is defined as $P_{\text{edge}} = w_e \cdot ||E_{gen}| - E_{target}|$, where $|E_{gen}|$ is the current edge count, E_{target} is the desired edge count, and w_e is the penalty weight.

The GA minimizes this aggregate distance, driving the population toward graphs that are statistically authentic in both local detail and global organization. The components of the fitness landscape are as follows.

1) Degree Distribution (MMD_d): To ensure the generated graphs reproduce realistic connectivity patterns, we compute the degree distribution histogram for the candidate graph. Let $H_d(G)$ be the normalized histogram of node degrees for graph G, computed using adaptive binning with a static count of ten bins to accommodate varying graph sizes. MMD_d measures the distance between $H_d(G)$ and the set of degree histograms. This penalizes graphs that fail to capture the density and distribution of the real data.

2) Clustering Coefficient (MMD_c): Local substructures are evaluated using the distribution of local clustering coefficients. For each node v, the clustering coefficient C_v measures the density of triangles in its neighbourhood. MMD_c measures

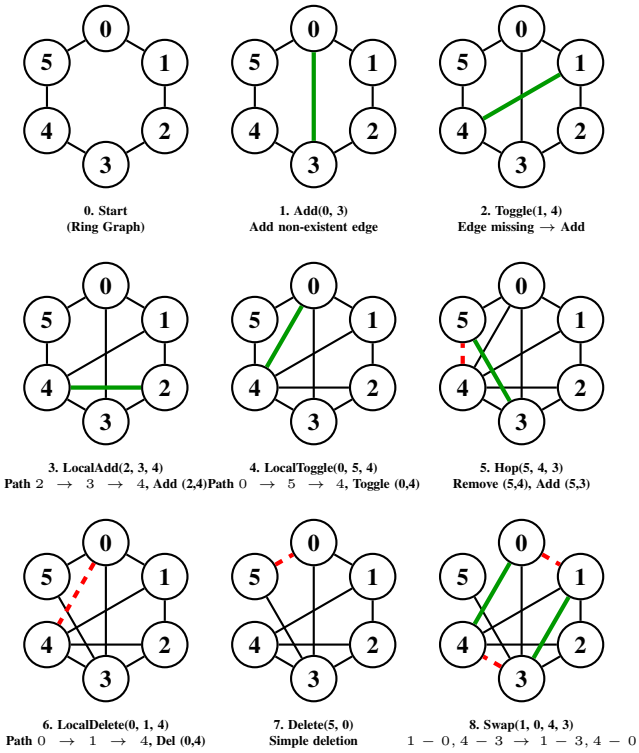

Fig. 2. Sequential visualization of the edge editing operations. Green edges are added; red dashed edges are removed. The chromosome producing the final graph (bottom right) from the start graph (top left) thus consists of these 8 operations in sequence.

TABLE I
SUMMARY OF DATASET CHARACTERISTICS

Property	MUTAG	ENZYMES	PROTEINS
Domain	Chemistry	Biochemistry	Biochemistry
Number of Graphs	188	600	1,113
Number of Classes	2	6	2
Avg. Nodes	17.93	32.63	39.06
Avg. Edges	19.79	62.14	72.82

TABLE II
SETTINGS FOR THE EVOLUTIONARY ALGORITHM

Parameter	Settings
Crossover/Mutation	50/80
Tournament Size	5
Population Size	500
Generations	300
Elitism	2
Operation Probability	Toggle: 1/14 Delete : 1/14 LocalAdd: 1/7 Hop : 1/7 Swap : 1/7 LocalDel : 1/7 Add : 1/14 LocalTog: 1/14 Null : 1/7

the distance between the graph and the data distribution. This metric is crucial for enforcing transitivity and community structure, properties often missed by simpler models.

3) Spectral Features (MMD_s): Global structural properties are captured using the graph spectrum. We compute the eigenvalues of the combinatorial Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{A}$ is the adjacency matrix and $\mathbf{D}$ is the degree matrix. The eigenvalues are computed and sorted. This spectrum encodes vital information regarding graph cuts, connectivity, and diffusion properties. By minimizing the MMD between the spectral densities, the GA ensures that the generated graphs share the same global topology as the target class.

III. EXPERIMENTAL DESIGN

A. Datasets

We assess our graph refinement methods using three benchmark graph classification datasets [8] that cover diverse domains and structural complexities, detailed in Table III-A below. The MUTAG dataset consists of chemical compound graphs for mutagenicity prediction and is characterized by relatively small graphs with an average of 18 nodes. ENZYMES represents protein structures distributed across six classes, consisting of moderate-sized graphs averaging 33 nodes. Finally, PROTEINS serves as the largest and most structurally varied dataset for distinguishing enzymes from non-enzymes, with graph sizes spanning from 4 to 620 nodes. The feature representations differ significantly among these benchmarks: while MUTAG incorporates both 7-dimensional node and 4-dimensional edge features, ENZYMES and PROTEINS rely exclusively on 3-dimensional node features. Collectively, these datasets provide a rigorous environment for testing graph generation performance across varying scales and complexities.

B. GA Setup

The hyperparameters used for the evolutionary refinement stage are detailed in Table II. The evolutionary process is managed by a Rust backend, which handles population management, fitness evaluation, and genetic operators in parallel.

1) Population Initialization and the Identity Genome: A critical component of our hybrid architecture is the initialization strategy. Standard evolutionary approaches often initialize populations with random noise; however, our goal is to *refine* the structural priors learned by the WGAN, not to generate graphs from scratch. Consequently, the initialization process in the GA is biased toward the WGAN output. The population of size P is initialized as follows. First, the *Identity Genome*, the first individual in the population, is explicitly constructed as a sequence consisting entirely of **Null**() operations; hence no changes are made to the base graph provided by the WGAN. This guarantees that the evolutionary search begins with a solution that is at least as fit as the original WGAN output, providing a stable baseline for improvement. The remaining $P - 1$ individuals are each initialized as a random sequence of operations, chosen according to the specified probabilities. This injects immediate diversity into the population, exploring the local structural neighbourhood of the base graph.

2) Selection and Variation: We employ tournament selection with a tournament size of 5. This balances exploration with exploitation of high fitness structural edits. To ensure that high quality refinements are never lost due to stochastic effects, we utilize elitism, preserving the top 2 individuals unchanged into the next generation.

We utilize high crossover (0.5) and mutation (0.8) probabilities. This aggressive variation strategy is viable because the operations are applied sequentially to a robust base structure, allowing the algorithm to rapidly test various combinations of edge additions, deletions, and swaps to correct specific spectral deviations and better exploit the search space.

IV. RESULTS AND DISCUSSION

A. Performance on Benchmark Datasets

Table III presents a comprehensive study comparing the WGAN generator output against the final GA refined graphs. The results demonstrate that the evolutionary refinement of the base graphs significantly enhances the structural character, particularly capturing global topology and clustering patterns.

For the MUTAG dataset, the refinement yielded a dramatic improvement in the Clustering MMD, dropping from values greater than 1.0 in the base WGAN to near zero for both classes. This indicates that the GA successfully recovered the specific structures of the generally small graphs in the dataset that the WGAN struggled to capture. While the Spectral MMD increased slightly, the gains in clustering result and improved Degree MMD suggest more plausible synthetic graphs.

With the ENZYMES dataset, we see consistent and sometimes substantial improvements in the Spectral MMD, notably in Class 0 (0.198 to 0.034) and Class 3 (0.098 to 0.021). This suggests that the edge editing operations were successful in

TABLE III
COMPARISON OF WGAN VS. GA-REFINED GRAPHS. LOWER MMD SCORES INDICATE BETTER DISTRIBUTIONAL MATCHING.

Dataset	Class	Nodes	Avg Edges		MMD Degree		MMD Clustering		MMD Spectral	
		(Real→WGAN)	WGAN	**Refined**	WGAN	**Refined**	WGAN	**Refined**	WGAN	**Refined**
MUTAG	Class 0	13.4→13.8	14.7	14.0	0.259	**0.247**	1.070	**0.000**	**0.081**	0.112
	Class 1	20.7→18.6	19.7	21.5	0.295	**0.270**	1.043	**0.000**	**0.113**	0.130
ENZYMES	Class 0	31.6→38.4	76.4	66.6	**0.210**	0.213	0.204	**0.200**	0.198	**0.034**
	Class 1	32.1→29.1	59.3	56.7	**0.137**	0.175	**0.127**	0.131	0.114	**0.099**
	Class 2	30.1→30.5	60.3	56.0	**0.156**	0.165	0.139	**0.135**	0.053	**0.034**
	Class 3	37.4→37.1	76.5	74.9	**0.210**	0.219	**0.168**	**0.168**	0.098	**0.021**
	Class 4	32.1→28.4	54.4	58.6	**0.147**	0.152	**0.115**	0.116	0.053	**0.048**
	Class 5	38.8→27.7	51.5	52.5	**0.146**	0.165	0.124	**0.122**	**0.056**	0.063
PROTEINS	Class 0	54.5→47.2	111.2	95.0	**0.059**	0.074	0.047	**0.046**	0.176	**0.026**
	Class 1	19.6→24.0	52.0	40.1	**0.091**	0.110	**0.057**	0.062	0.076	**0.071**

TABLE IV
COMPARISON WITH STATE-OF-THE-ART METHODS USING MMD METRICS (LOWER IS BETTER) AS PRESENTED IN [5], [9], [10]

Model	PROTEINS			ENZYMES		
	Deg.	Clust.	Spect.	Deg.	Clust.	Spect.
DeepGMG [7]	0.96	0.63	–	0.43	0.38	–
GraphRNN [14]	0.04	0.18	–	0.06	0.20	–
LGGAN [5]	0.18	0.15	–	0.09	0.17	–
WPGAN [9]	**0.03**	0.31	–	**0.02**	0.28	–
Edge-WGAN [10]	0.09	0.07	0.07	0.10	0.08	0.05
(Ours) Refined	0.07	**0.03**	**0.01**	0.07	**0.04**	**0.02**

adjusting the graph eigenvalues to match the real distribution. In some of these cases we see a slight trade off, particularly with regards to Degree MMD, as the algorithm prioritized global connectivity over the degree counts.

The PROTEINS dataset further highlights the strength of the refinement in fixing topological properties. Spectral MMD improved significantly for both classes (e.g., Class 0: 0.176 to 0.026), and Clustering MMD remained competitive. The average edge counts in the refined graphs (95.0 for Class 0) dropped compared to the WGAN output (111.2), with the real data distribution for Class 0 being 103.

B. Comparison to State of the Art

Table IV compares the results of our refinement process against DeepGMG, GraphRNN, LGGA, and the base WGAN. The refined graphs outperform the baseline WGAN in Spectral MMD for both PROTEINS (0.01) and ENZYMES (0.02). In terms of Clustering MMD, the refined graphs achieve the best performance on PROTEINS (0.03) and remains highly competitive on ENZYMES (0.04), outperforming GraphRNN and LGGAN. While the degree distribution MMD is slightly higher than the best baseline in some cases, the overall balance of metrics indicates that the hybrid WGAN-GA approach generated graphs that are structurally more robust and representative of the target domain.

C. Visual Analysis

Figure 3 provides a qualitative comparison of the generated graphs. These two graphs were chosen as representative as they are both close to the mean cardinality of other graphs of their class. The refined graphs exhibit more coherent structures with fewer disconnected components compared to the raw WGAN outputs. This visual inspection correlates with the quantitative improvements in Spectral MMD, as the GA successfully connects isolated nodes and breaks unrealistic hubs not found in the source data. This connectivity also occurs while simultaneously decreasing the edge count in some cases to better match the data.

Figure 4 displays the histograms for degree, clustering coefficient, and spectral distributions respectively. The refined distributions (orange) generally track the real data (black lines) more closely than the WGAN outputs (blue). Notably, the spectral distribution in Figure 4 shows that the refinement process effectively shifts the eigenvalue spectrum to align much closer to the ground truth, correcting the skew in distribution seen in the coarser WGAN generation.

V. CONCLUSIONS AND FUTURE WORK

In this work, we presented a framework for graph generation and refinement that combines global modelling of Wasserstein GANs with local optimization of GAs. By using the state of the art WGAN outputs as high quality initial seed populations, our GA is able to effectively refine the graph structure, correcting deviations in spectral and clustering that the WGAN on its own could not capture.

Our results on multiple benchmark datasets demonstrate that this two-stage approach consistently improves structural fidelity of the graphs that were already optimized using a GAN in [10]. Specifically, the evolutionary refinement significantly lowers MMD scores for spectral and clustering distributions, giving us graphs that are topologically more realistic. Offloading this computationally heavy GA process to an optimized library ensures that the refinement is as efficient as possible.

Future work could focus on expanding the ways in which the GA can be used in the graph generation, perhaps as an

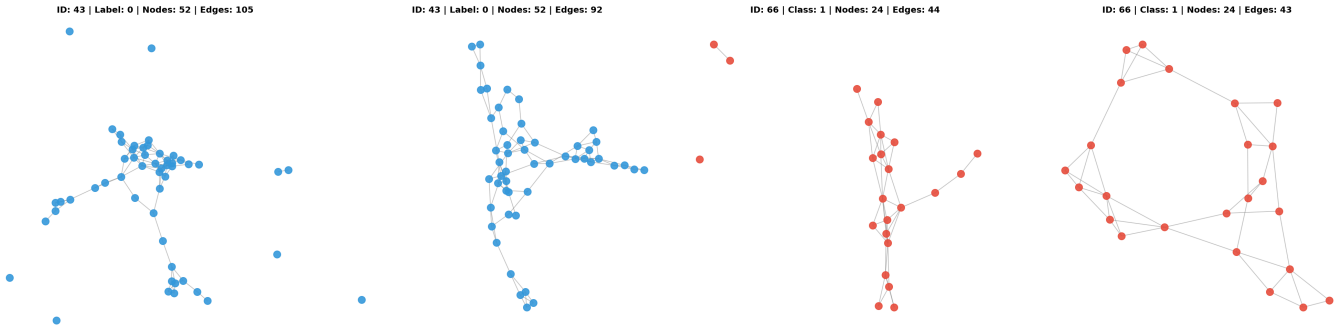

Fig. 3. Visual comparison on PROTEINS dataset (left: generated and refined graphs from Class 0; right: Class 1).

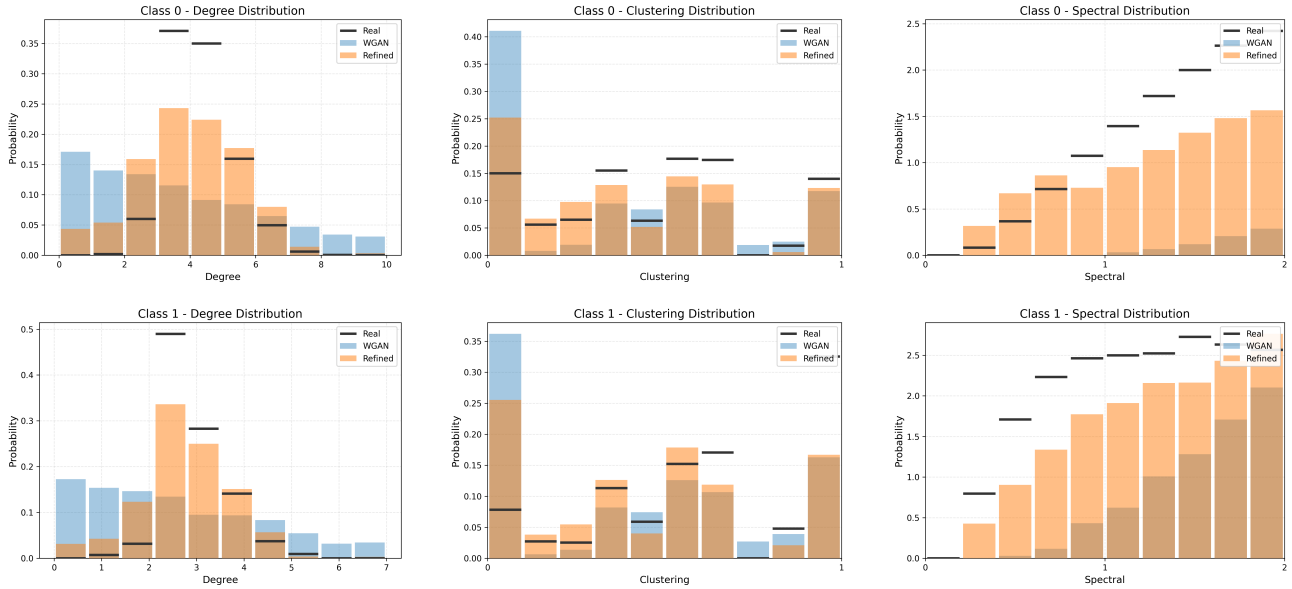

Fig. 4. Distribution comparison between real and generated graphs on the PROTEINS dataset.

integral part of the generator process in graph based GANs. Researchers could also use the graph generation capability of the GA to replace the edge prediction component of the GAN, without it being stacked at the end of the training cycle.

Acknowledgements

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), and used the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET) and Compute Canada.

References

[1] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *Proc. Intl. Conf. on Machine Learning*, pages 214–223, 2017.

[2] A. Barry, J. Griffith, and C. O'Riordan. An evolutionary and graph-rewriting based approach to graph generation. In *Intl. Joint Conf. on Computational Intelligence (IJCCI)*, volume 1, pages 237–243, 2015.

[3] N. De Cao and T. Kipf. MolGAN: An implicit generative model for small molecular graphs. *ICML 2018 workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018.

[4] M. Dubé, S. Houghten, and D. Ashlock. Representation for evolution of epidemic models. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pages 2370–2377, Piscataway NJ, 06 2019. IEEE Press.

[5] S. Fan and B. Huang. Labeled graph generative adversarial networks. *arXiv preprint arXiv:1906.03220*, 2019.

[6] I. Gulrajani et al. Improved training of Wasserstein GANs. In *Advances in Neural Information Processing Systems*, 2017.

[7] Y. Li et al. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018.

[8] C. Morris et al. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020.

[9] S.A. Razi Razavi et al. Generating labeled graphs using conditional Wasserstein GANs. In *35th International Conference on Collaborative Advances in Software and Computing (CASCON)*. IEEE Computer Society, 2025.

[10] S.A. Razi Razavi et al. Adaptive edge learning for density-aware graph generation. *arXiv preprint arXiv:2601.23052*, 2026.

[11] M. Simonovsky and N. Komodakis. GraphVAE: Towards generation of small graphs using variational autoencoders. In *Intl. Conf. on Artificial Neural Networks*, 2018.

[12] H. Wang et al. GraphGAN: Graph representation learning with generative adversarial nets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[13] P. Wang et al. Graph generative adversarial networks with evolutionary algorithm. *Applied Soft Computing*, 164:111981, 2024.

[14] J. You et al. Graph convolutional policy network for goal-directed molecular graph generation. In *Advances in Neural Information Processing Systems*, 2018.