Improving Graph Generation by Restricting Graph Bandwidth

Anonymous authors
Paper under double-blind review

Abstract

Deep graph generative modeling has proven capable of learning the distribution of complex, multi-scale structures characterizing real-world graphs. However, one of the main limitations of existing methods is their large output space, which limits generation scalability and hinders accurate modeling of the underlying distribution. To overcome these limitations, we propose a novel approach that significantly reduces the output space of existing graph generative models. Specifically, starting from the observation that many real-world graphs have low graph bandwidth, we restrict graph bandwidth during training and generation. Our strategy improves both generation scalability and quality without increasing architectural complexity or reducing expressiveness. Our approach is compatible with existing graph generative methods, and we describe its application to both autoregressive and one-shot models. We extensively validate our strategy on synthetic and real datasets, including molecular graphs. Our experiments show that, in addition to improving generation efficiency, our approach consistently improves generation quality and reconstruction accuracy. The implementation is made available[1].

1 Introduction

Learning the underlying distribution of graphs for generative purposes finds important applications in diverse fields, where objects can be naturally described through their structures (Hamilton et al., 2017). Deep graph generative modeling has recently been proven capable of learning both global and fine-grained structural properties, along with their complex interdependencies (Guo & Zhao, 2022). These results have shown promise for many applications, such as in the biomedical domain, where essential objects (molecules, gene networks, tissues, etc.) can be represented as graphs (Li et al., 2022). Despite these promises, several open challenges remain.

Applications in these domains require modeling graphs with a high number of nodes N that leads to a large output space, where the number of possible edges is in $\mathcal{O}(N^2)$. At the same time, many real-world graphs are sparse, and they are characterized by a small number of semantically rich connections which need to be accurately modeled (*e.g.*, a small subset of chemical bonds can confer radically different properties in a molecular graph).

Recent research has focused on accurately learning complex dependencies leveraging generative models such as variational autoencoders (VAEs) (Grover et al., 2019), recurrent neural networks (RNNs) (You et al., 2018), normalizing-flow models (Shi et al., 2020) and score-based models (Niu et al., 2020). A general limitation of these approaches is their high time complexity and output space $\mathcal{O}(N^2)$. This limits both their scalability and makes accurate prediction of sparse connections challenging, as the ratio of observed to possible edges can be extremely small. For this reason, more tractable methods have been proposed, which leverage different architectures (Li et al., 2018; Liu et al., 2018; Dai et al., 2020), generate coarse-grained motifs (Jin et al., 2018; Liao et al., 2019), or change the output representation, such as transforming graphs to sequences (Goyal et al., 2020) or using domain-specific encodings such as molecular SMILES (Gómez-Bombarelli et al., 2018). Although these approaches are more scalable, they trade-off efficiency with model complexity, expressiveness, or have limited applicability because of domain-specific choices.

[1]The implementation will be publicly released upon unblinding.

To overcome the limitations of existing approaches, we propose a novel strategy: BwR (*Bandwidth-Restricted*) graph generation. BwR leverages a permutation of the adjacency matrix to restrict the *graph bandwidth*, reducing both the time complexity and the output space from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \cdot \hat{\varphi})$, where $\hat{\varphi}$ is the estimated bandwidth. As will be shown, $\hat{\varphi}$ is low for many classes of real-world graphs, such as those characterizing the biomedical domain. During training, BwR leverages bandwidth-restricted adjacency matrices; during sampling, it constrains the generation within a bandwidth-restricted space, which reduces both time complexity and output space without losing expressiveness.

This strategy brings two key advantages. First, reducing time complexity improves *generation scalability* (*i.e.*, time and memory requirements). Second, reducing the output space simplifies learning the underlying data distribution, while also making the ratio of observed to possible edges less imbalanced, with a positive impact on *generation quality*. Importantly, BwR can be easily integrated into virtually all existing graph generative methods, as it is orthogonal to the generative model architecture. Therefore, it does not increase model complexity nor add domain-specific constraints.

Contributions. We summarize our main contributions as follows:

- We show that many real-world classes of graphs, such as molecules, are characterized by a low graph bandwidth.
- Building on this property, we propose *BwR* (Bandwidth-Restricted) graph generation, a novel strategy for graph generation that constrains the bandwidth, drastically reducing time complexity and output space.
- BwR can be applied to virtually all existing graph generation methods. We describe its application to an autoregressive method, GraphRNN (You et al., 2018); a one-shot VAE-based method, Graphite (Grover et al., 2019); and a one-shot score-based method, EDP-GNN (Niu et al., 2020).
- We experimentally validate BwR on both synthetic and real-world datasets, with a focus on real-world molecular datasets. Our results show that, in addition to being more efficient in terms of time and memory used, BwR consistently improves reconstruction accuracy and generative quality across datasets and methods.

2 Background

Graph Generative Models. Graph generative models seek to learn the underlying distribution of graph datasets. A model is trained on a set of observed graphs $\mathcal{G} = \{G_1, \ldots, G_S\} \sim p(G)$, where each graph $G_i = (\mathcal{V}_i, \mathcal{E}_i)$ is defined by its set of nodes $\mathcal{V}_i = (v_1, \ldots, v_N)$ and edges $\mathcal{E}_i \subseteq \mathcal{V}_i \times \mathcal{V}_i$. The model learns the distribution $p_{\text{model}}(G) \approx p(G)$ that allows sampling new graphs. Broadly, graph generative models can be categorized as *autoregressive* (You et al., 2018; Shi et al., 2020; Goyal et al., 2020; Li et al., 2018; Liu et al., 2018) or *one-shot* (Kipf & Welling, 2016; Ma et al., 2018; Grover et al., 2019; Niu et al., 2020) models. A common way to represent the graph topology is through its adjacency matrix $A^\pi \in N \times N$. The adjacency matrix depends on a specific node ordering π, defined as a bijective function $\pi : \mathcal{V} \to [1, N]$. Autoregressive models treat graph generation as a sequential decision process, conditioning each new node on the already-generated graph, with time complexity and output space in $\mathcal{O}(N^2)$. One-shot models sample the whole topology of the graph from a latent distribution, and they need to consider edges between every pair of nodes, thus also leading to a time complexity and output space in $\mathcal{O}(N^2)$. In contrast, BwR allows reducing time complexity and output space of both classes of models to be in $\mathcal{O}(N \cdot \hat{\varphi})$, with no loss of expressiveness and without increasing model complexity. Additionally, our approach is orthogonal and compatible with other methods proposed to increase GNN efficiency—such as graph partitioning (Jia et al., 2020)—as it is largely independent of the generative method.

Graph Ordering. Given that our method imposes a specific node ordering, it is related to other works that have investigated ordering in graph generation. A unique challenge of graph generative models is that the set of all possible orderings leads to up to $N!$ different adjacency matrices for the same graph (Liao et al., 2019). Choosing a specific ordering π does not rigorously correspond to maximum-likelihood estimation (MLE), and can make the reconstruction loss ambiguous (Liao et al., 2019). For this reason, several methods have been proposed to improve likelihood estimation

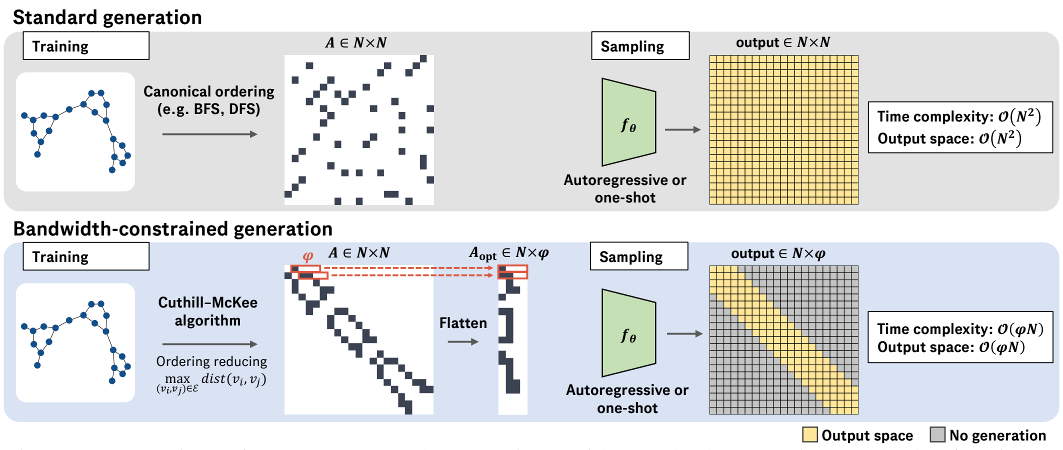
Standard generation
Training
Canonical ordering (e.g. BFS, DFS)
$A \in N \times N$
Sampling
f_θ
Autoregressive or one-shot
output $\in N \times N$
Time complexity: $\mathcal{O}(N^2)$
Output space: $\mathcal{O}(N^2)$
Bandwidth-constrained generation
Training
Cuthill–McKee algorithm
Ordering reducing $\max_{(v_i,v_j)\in\mathcal{E}} dist(v_i, v_j)$
φ
$A \in N \times N$
$A_{\text{opt}} \in N \times \varphi$
Flatten
Sampling
f_θ
Autoregressive or one-shot
output $\in N \times \varphi$
Time complexity: $\mathcal{O}(\varphi N)$
Output space: $\mathcal{O}(\varphi N)$
Output space
No generation

Figure 1: Overview of our strategy and comparison with standard generation methods. **(Top)** In a standard graph generative method, the model is trained on adjacency matrices A derived through a specific canonical ordering on the graph (*e.g.*, BFS or DFS). During sampling, the model needs to predict edges from a space in $\mathcal{O}\left(N^2\right)$. **(Bottom)** Our bandwidth-restricted graph generation leverages the Cuthill-Mckee (C-M) ordering (Cuthill & McKee, 1969) to reduce the bandwidth $\varphi(A)$ of each adjacency matrix. The C-M order results in an adjacency matrix that is a *band matrix*, with all-zero entries outside a $\varphi(A)$-sized band. A is re-parameterized as $A^{\text{opt}} \in N \times \varphi(A)$, which is used for training. During sampling, only edges in an $N \times \varphi(A)$ space (yellow) are considered as candidates, thus drastically reducing the output space to $\mathcal{O}\left(N \times \varphi(A)\right)$.

and reduce ambiguity in graph generative models (Liao et al., 2019; Chen et al., 2021; Winter et al., 2021). We note that these works are orthogonal with respect to our contribution. Indeed, bandwidth-optimized graphs define a canonical family of node orderings, and existing methods could be used to improve likelihood estimation.

Graph Bandwidth. We define graph bandwidth through the graph bandwidth problem (Unger, 1998). Intuitively, the graph bandwidth problem can be seen as placing the nodes of a graph on a line such that the length of the longest edge in the graph is minimized. The bandwidth of the graph is then simply the length of the longest edge. We provide a formal definition of the graph bandwidth problem in Appendix A.1.

Importantly, an ordering that minimizes φ results in an adjacency matrix where all non-zero entries lie in a narrow band along the diagonal (hence the term “bandwidth”). This enables a compact matrix representation of $N \times \varphi$ instead of N^2 (Figure 1) and drastically reduces the space required to represent the graph when $\varphi \ll N$. The maximum size of the off-diagonal band of the adjacency matrix is known as the matrix’s bandwidth, which we denote as $\varphi(A)$. Figure 2 shows the bandwidth of two adjacency matrices of a molecular graph. We note that for an ordering π, $\varphi(\pi) = \varphi(A^\pi)$. The graph bandwidth problem has been shown to be NP-hard for general graphs (Papadimitriou, 1976). However, in practice, efficient heuristic approaches work well for general graphs and are routinely leveraged in applications[2]. One such heuristic is the Cuthill-McKee (C-M) algorithm (Cuthill & McKee, 1969), which is based on a variation of BFS search, has linear time complexity $\mathcal{O}(|\mathcal{E}|)$ (Chan & George, 1980), and has been extensively studied from a theoretical perspective (Turner, 1986). We leverage the C-M algorithm for BwR, though our strategy is independent of the choice of the bandwidth minimization algorithm. We define the bandwidth of the adjacency matrix derived with the C-M algorithm as $\hat{\varphi}$.

To the best of our knowledge, the concept of graph bandwidth has been used in the context of GNNs only by Balog et al. (2019), with the explicit purpose of improving dense implementations on custom hardware. Their work does not leverage the bandwidth in the model itself and does not target graph generation.

[2]Additional details about bandwidth complexity and references are included in Appendix A.1.

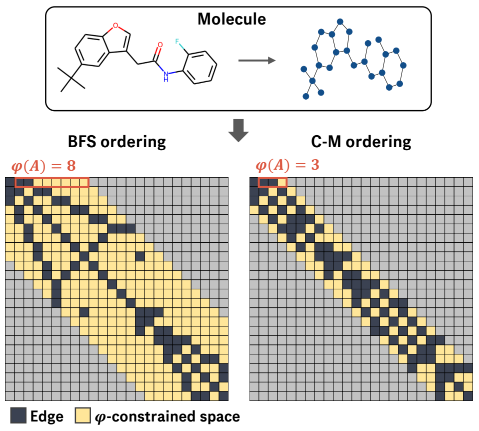

Figure 2: Bandwidth of two adjacency matrices of the same molecular graph. The left adjacency matrix is given by a BFS ordering, the right adjacency matrix is given by a Cuthill-McKee ordering.

Dataset	N	$\hat{\varphi}$	Saving Factor
ZINC250k	23.2 ± 4.5	3.3 ± 0.8	3.9 ± 1.0
MCF-7	26.1 ± 10.7	4.1 ± 1.3	3.5 ± 1.2
MOLT-4	25.8 ± 10.3	4.0 ± 1.3	3.5 ± 1.2
Mutagenicity	28.5 ± 14.1	5.5 ± 2.2	2.9 ± 1.1
NCI1	29.3 ± 13.4	4.3 ± 1.5	3.7 ± 1.3
PC-3	26.1 ± 10.6	4.1 ± 1.4	3.5 ± 1.2
QM9	18.0 ± 2.9	5.3 ± 1.5	2.0 ± 0.4
SF-295	25.8 ± 10.3	4.0 ± 1.3	3.5 ± 1.2
UACC257	25.8 ± 10.3	4.0 ± 1.3	3.5 ± 1.2
Yeast	21.1 ± 8.8	3.6 ± 1.2	3.3 ± 1.1
Peptides-func	150.9 ± 84.5	5.7 ± 2.6	13.5 ± 6.5
DD	277.7 ± 217.3	36.0 ± 20.7	4.1 ± 1.4
ENZYMES	31.7 ± 13.3	5.4 ± 2.2	3.3 ± 1.2
KKI	27.0 ± 19.5	7.2 ± 5.1	2.2 ± 0.6
OHSU	82.0 ± 43.7	20.0 ± 13.2	2.4 ± 0.7

Table 1: Number of nodes, C-M bandwidth, and saving factor of a set of chemical and biological datasets. The first section of the table consists of small molecules, the second section of large molecules, and the third of brain networks. See Appendix A.4 for more details.

3 Bandwidth-Restricted Graph Generation

In this section we describe BwR, our novel approach for improving graph generation. First, we motivate our approach by empirically showing that many classes of real-world graphs have low graph bandwidth in Section 3.1. As BwR can be combined with different existing generative methods, we first describe its general strategy and principles in Section 3.2. Then, we detail the strategies for bandwidth-restricted graph generation applied to an autoregressive model based on GraphRNN (You et al., 2018) in Section 3.3, and two distinct one-shot models based on Graphite (Grover et al., 2019) and EDP-GNN (Niu et al., 2020) in Section 3.4.

3.1 Bandwidth of Real-world Biomedical Datasets

A key observation that motivates the present work is that many real-world graphs, such as those characterizing the biomedical domain, have low $\hat{\varphi}$. Table 1 shows the empirical bandwidth computed with the C-M algorithm for a diverse set of chemical and biological datasets. For each dataset, a *saving factor* summarizes the space reduction. The saving factor is the ratio of the number of edges in the bandwidth-restricted graph to the number of edges in the complete graph (*i.e.* the size of a non-bandwidth-restricted adjacency matrix). As shown, a bandwidth reparameterization leads to a saving factor > 3 for most small-molecule datasets (*e.g.*, 3.8 ± 1.0 for ZINC250k). The saving factor is even higher for datasets including larger molecules (*e.g.*, 13.5 ± 6.5 for the Peptides-func dataset). Significant savings are also confirmed on non-molecular datasets, such as brain networks.

The C-M algorithm consistently reduces $\varphi(\pi)$ compared to the orderings routinely used in graph generation (BFS, DFS, etc.). Figure 2 compares the adjacency matrices of a molecular graph given by BFS and C-M, and their respective bandwidths. The bandwidth decreases from 8 to 3, which translates into a two-fold reduction of the output space. Additional examples of adjacency matrices for molecular graphs given by BFS, DFS, RDKit (Landrum, 2006), and C-M orderings are shown in Figure 4 (Appendix A.2). As shown, the C-M ordering consistently leads to the lowest φ. Overall, the C-M algorithm allows reducing the bandwidths of all the considered datasets. For example, 95% of the molecules in ZINC250k have $\hat{\varphi} \leq 4$ (Figure 5, Appendix).

3.2 Restricting Graph Bandwidth

Starting from the observation that many real-world graphs have low bandwidth, we propose to reduce the output space of a graph generative model from $N \times N$ to[3] $N \times \hat{\varphi}$. As the goal of graph

[3]Technically, as the adjacency matrix is symmetric, only a triangular matrix is generated both in the standard formulation and in our bandwidth-restricted re-parameterization.

generation is to learn a distribution of the data $p\,(G)$, we assume that, given $\hat{\varphi}_{\text{data}}$ the maximum empirical bandwidth on the training set $\mathcal{G}_{\text{train}}$, we can reduce the output space of the generative model to $N \times \hat{\varphi}_{\text{data}}$ without losing expressiveness (*i.e.*, without losing the ability to generate in-distribution graphs). In general, we achieve this reduction through two complementary mechanisms: (1) imposing a bandwidth-reducing ordering and (2) restricting the output space to a dataset-specific bandwidth $\hat{\varphi}_{\text{data}}$ or a graph-specific bandwidth $\hat{\varphi}$ (Figure 1, bottom).

During training, for each graph we use the precomputed adjacency matrix A^{π^*} with the ordering π^* given by the previously introduced C-M algorithm. We remark that, given the linear time complexity of the C-M algorithm, our preprocessing does not introduce any additional overhead compared to standard orderings such as BFS/DFS. Notably, just restricting training examples to a specific canonical ordering does not guarantee that such an ordering will be respected during generation (Chen et al., 2021), and does not provide any advantage in time complexity or output space reduction, since the complete adjacency matrix needs to be generated. Therefore, we also re-parameterize the adjacency matrix as $A^{\pi^*}_{\text{opt}} \in N \times \hat{\varphi}_{\text{data}}$ (or $N \times \hat{\varphi}$ for each graph), dropping the zeros outside the bandwidth. During sampling, only edges belonging to the reduced matrix are considered as candidates, thus constraining the output space and reducing the time complexity, without losing expressiveness.

Below, we discuss the details of our strategy applied to different models and highlight model-specific choices and advantages.

3.3 Autoregressive Graph Generation

Autoregressive graph generation approaches recursively generate the edges of a single node (You et al., 2018) or a group of nodes, (Liao et al., 2019) conditioned on the previously generated subgraph. We will focus on GraphRNN (You et al., 2018), although a similar approach could be applied to virtually any autoregressive graph generative model.

In GraphRNN, the probability of node v_i being connected to node v_j, with $\pi(v_j) < \pi(v_i)$, is parameterized by an output function f_θ applied to the hidden state of an RNN over the previous rows of the adjacency matrix:

$$p[(v_i, v_j) \in \mathcal{E}] = f_\theta(\text{RNN}_\phi(A^{\pi}_{1:i-1}))_j$$

where θ and ϕ are parameters learned to maximize the likelihood of the data. Additional model details are provided in Appendix A.3.3.

We note that a d-unit f_θ can generate graphs with at most bandwidth d. Potentially, we could set d to be the maximum number of nodes N of any graph we want to generate, which would ensure maximum expressiveness. Instead, we set d equal to the maximum bandwidth of any A^{π} we would want to generate, greatly increasing efficiency and reducing training signal sparsity for low bandwidth graphs. We find the order π for each graph G by using the C-M algorithm and set $d = \hat{\varphi}_{\text{data}}$ as the maximum bandwidth of all of the A^{π} in the training data. Compared to generating N rows of length $d = N$, we generate $\mathcal{O}(N/\hat{\varphi}_{\text{data}})$-times fewer edges.

In the original GraphRNN model, You et al. (2018) used a random BFS ordering during training, and set $d = M_{\text{data}} < N$, with M_{data} defined as the maximum number of nodes in the BFS queue at any time in the training data. M_{data} is estimated empirically by sampling 100,000 BFS orderings per dataset and setting M_{data} to be roughly the 99.9 percentile of the empirical distribution of maximum queue sizes. Critically, we observe that M_{data} derived in You et al. (2018) approximates the *maximum bandwidth across all possible BFS orderings*. In contrast, $\hat{\varphi}_{\text{data}}$ is derived by explicitly reducing the bandwidth. Notably, our approach allows significantly shrinking d, thus directly reducing the output space and the time complexity. For example, on the DD dataset (Dobson & Doig, 2003) of protein graphs, the authors set $d = M_{\text{data}} = 230$, whereas we derive $d = \hat{\varphi}_{\text{data}} = 122$, a nearly two-fold reduction.

3.4 One-shot Graph Generation

One-shot graph generative models sample the entire graph topology simultaneously. Typically the topology is represented by putting edge probabilities on each pair of nodes, resulting in a complete graph with a probability placed on each edge by the generative model (Simonovsky & Komodakis, 2018; Grover et al., 2019; Thomas & Welling, 2016). We will call this graph the *edge-probability* *graph.* Our key insight is that the complete edge probability graph can be replaced with a bandwidth-restricted edge-probability graph (Figure 1, bottom).

One-shot models can be divided into two main categories: (1) *Node-embedding-based* models sample node embeddings from the latent distribution and compute the edge-probability graph based on pairwise relationships, and (2) *Adjacency-matrix-based* models directly output the edge-probability graph. We focus on both categories, considering a model based on Graphite (Grover et al., 2019) and a model based on EDP-GNN (Niu et al., 2020).

Node-embedding-based one-shot generation. Graphite (Grover et al., 2019) is a VAE-based approach with one latent vector $\mathbf{z_i} \in \mathbb{R}^d$ per node with standard Gaussian prior. The encoder network uses graph convolution layers (Kipf & Welling, 2017) on the input adjacency matrix $A \in \mathbb{R}^{N \times N}$ and node features $X \in \mathbb{R}^{N \times k}$ to derive the mean and standard deviation of the variational marginals, $\boldsymbol{\mu}, \boldsymbol{\sigma} = \text{GNN}_\phi(A, X)$, where $\boldsymbol{\mu}, \boldsymbol{\sigma} \in \mathbb{R}^{N \times d}$. We will focus on the case without additional node features, so X can be a positional encoding dependent on the node ordering. The decoder network outputs edge probabilities:

$$p[(v_i, v_j) \in \mathcal{E}] = \text{GNN}_\theta(\hat{A}, X, Z)_{i,j} \tag{1}$$

where $\hat{A}$ is fully-connected and Z are samples from $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$. Further architectural details are described in Appendix A.3.4. In our bandwidth-restricted version of Graphite, we constrain the bandwidth of $\hat{A}$. For graph G, we get the bandwidth $\varphi(A^\pi)$ for π found using the C-M algorithm and build a new edge set:

$$\hat{\mathcal{E}} = \{(i, j) \mid 1 \leq |i - j| \leq \varphi(A^\pi)\} \tag{2}$$

where $1 \leq i, j \leq N$. With the new edge set, the decoder message passing steps are reduced from complexity in $\mathcal{O}(N^2)$ to $\mathcal{O}(N \cdot \hat{\varphi})$. During generation, the standard Graphite model samples the number of nodes from the empirical distribution of the training data. In our bandwidth-restricted version, both the number of nodes and the bandwidth φ_{data} are sampled from the empirical distribution of the training data, thus further reducing the output space.

Adjacency-matrix-based one-shot generation. EDP-GNN (Niu et al., 2020) is a score-based model in which a GNN is trained to match the score function of the data distribution. Intuitively, EDP-GNN learns to denoise the upper-right triangle of the adjacency matrix. Our modification denoises the bandwidth-restricted upper-right triangle, thus reducing the modeled output space (Figure 6, Appendix). In EDP-GNN, a multi-layer perceptron (MLP) predicts the final edge features from intermediate edge features $\hat{A}$ built by message passing layers and edge-update layers:

$$\mathbf{s}_\theta(A)_{i,j} = \text{MLP}(\hat{A}_{ij}). \tag{3}$$

To constrain the bandwidth, we restrict $(i, j) \in \hat{\mathcal{E}}$ (Eq. 2) in the final MLP and in all of the message passing and edge-update layers using the same approach described for Graphite. This drastically reduces the time complexity and output space. Further details are included in Appendix A.3.5.

4 Experiments

We experimentally validate our method on both synthetic and real graphs, comparing bandwidth-constrained architectures and non-constrained baselines.

4.1 Metrics and Experimental Setup

We measure two goals of the models: closeness of the sample distribution to the true distribution of graphs, and reconstruction quality. In order to measure the quality of the sample distribution consistently with the recent literature, we use the Maximum Mean Discrepancy (MMD) between sampled and test graph statistics (You et al., 2018). The graph statistics we compare are degree, clustering coefficient, node orbit counts following You et al. (2018), and spectrum (the eigenvalues of the normalized graph Laplacian) following Liao et al. (2019). To track overall sample quality, we compute the mean MMD^2 across all four MMD metrics[4]. Reconstruction quality is measured by comparing the reconstructed graph and the original input graph using the Area Under the Precision

[4]We note that previous works usually report MMD^2 but they indicate MMD in the results.

		Community2		Planar		grid2d		DD		ENZYMES		PROTEINS	
		MMD^2 ↓	AUPRC ↑	MMD^2 ↓	AUPRC ↑	MMD^2 ↓	AUPRC ↑	MMD^2 ↓	AUPRC ↑	MMD^2 ↓	AUPRC ↑	MMD^2 ↓	AUPRC ↑
GraphRNN	Standard	**0.028**	0.376	0.265	0.545	0.323	0.642	**0.174**	**0.299**	0.023	0.445	**0.017**	0.533
	BwR [ours]	**0.024**	**0.421**	**0.233**	**0.652**	**0.240**	**0.999**	**0.234**	**0.318**	**0.016**	**0.602**	**0.020**	**0.716**
Graphite	Standard	0.055	0.706	**0.361**	0.975	0.649	0.453	**0.368**	0.804	0.107	0.925	0.153	**0.95**
	BwR [ours]	**0.047**	**0.747**	**0.468**	**0.990**	**0.528**	**0.915**	**0.273**	**0.840**	**0.038**	**0.950**	**0.037**	0.933
EDP-GNN	Standard	**0.030**	–	**0.459**	–	**0.645**	–	OOM	–	**0.092**	–	0.077	–
	BwR [ours]	**0.040**	–	**0.474**	–	**0.590**	–	**0.299**	–	**0.027**	–	**0.024**	–

Table 2: Graph generation results on generic datasets. **Bold** indicates best results compared to the other model of the same type and dataset. Significance was determined by Welch's t-test with five replicates per model. Models are considered comparable when $p \geq 0.05$. MMD^2 denotes average MMD^2 across four metrics (degree, cluster, orbit, spectra). Due to space limitations, we provide all the individual metrics in Table 4. OOM denotes out-of-memory issues. Hyphen (–) denotes not applicable metric/model.

		ZINC250k						Peptides-func					
		Deg.	Cluster	Orbit	Spectra	Mean ↓	AUPRC ↑	Deg.	Cluster	Orbit	Spectra	Mean ↓	AUPRC ↑
GraphRNN	Standard	0.025	**0.045**	0.012	**0.071**	**0.038**	0.668	**0.009**	**0.004**	**0.000**	**0.108**	**0.030**	0.809
	BwR [ours]	**0.011**	**0.044**	**0.005**	**0.057**	**0.029**	**0.783**	**0.008**	**0.001**	**0.001**	**0.123**	**0.033**	**0.903**
Graphite	Standard	0.049	0.516	0.005	0.043	0.153	**0.999**	0.169	**0.235**	**0.030**	**0.293**	0.182	**0.987**
	BwR [ours]	**0.009**	**0.307**	**0.002**	**0.019**	**0.084**	0.995	**0.056**	**0.216**	**0.011**	**0.198**	**0.120**	**0.993**
EDP-GNN	Standard	0.174	**0.055**	0.024	0.170	**0.106**	–	0.159	**0.041**	0.047	**0.213**	**0.115**	–
	BwR [ours]	**0.015**	0.528	**0.004**	**0.023**	**0.143**	–	**0.050**	0.371	**0.007**	**0.144**	**0.143**	–

Table 3: Graph generation results on molecular datasets. **Bold** indicates best results compared to the other model of the same type and dataset. Significance was determined by Welch's t-test with five replicates per model. Models are considered comparable when $p \geq 0.05$. Graph statistics (degree, cluster, orbit, spectra) are reported as MMD^2. Mean computed across individual statistics for each model/dataset. Hyphen (–) denotes not applicable metric/model.

Recall Curve (AUPRC). For GraphRNN, we compare the reconstructed row and the original row; for Graphite, we compare the reconstucted graph and the original graph[5].

We compare our bandwidth-restricted versions (+BwR) of the models based on GraphRNN (You et al., 2018), Graphite (Grover et al., 2019), and EDP-GNN (Niu et al., 2020) to their standard baselines (i.e. without BwR). Our models are described in Section 3 and additional details are provided in Appendix A.3. Each model architecture is individually hyperoptimized (details in Appendix A.3.2). All the experiments are repeated five times and significance is determined by Welch's t-test (models are considered comparable when p-value ≥ 0.05).

4.2 Generic Graph Generation

Datasets. We evaluated our models on six standard graph generation datasets, including both synthetic and real-world graphs: (1) *Community2*, 1500 two-community graphs generated using an Erdös–Renyi model with 60–160 nodes; (2) *Planar*, 1500 random planar graphs with 64 nodes made using Delaunay triangulation; (3) *Grid2d*, 66 distinct two-dimensional grids with side lengths between 10 and 20; (4) *DD*, 918 protein graphs with amino acids as nodes (Dobson & Doig, 2003); (5) *Enzymes*, 556 protein graphs of enzyme tertiary structures from the BRENDA database (Schomburg et al., 2004); and (6) *Proteins*, 904 protein graphs from the Protein Data Bank (Dobson & Doig, 2003). Additional details on the datasets are provided in Appendix A.4.2.

Results. Table 2 summarizes the results for generic graph generation. Extended results are shown in Table 4 (Appendix). As shown, our approach consistently achieves superior or competitive performance across datasets and methods. GraphRNN+BwR and Graphite+BwR outperform their standard counterparts in five out of six datasets, with comparable performance on the sixth. BwR improves EDP-GNN generation quality in two out of six datasets, with comparable performance on

[5]Reconstruction quality is less easily definable for score-based models, therefore we omit AUPRC in EDP-GNN evaluation.

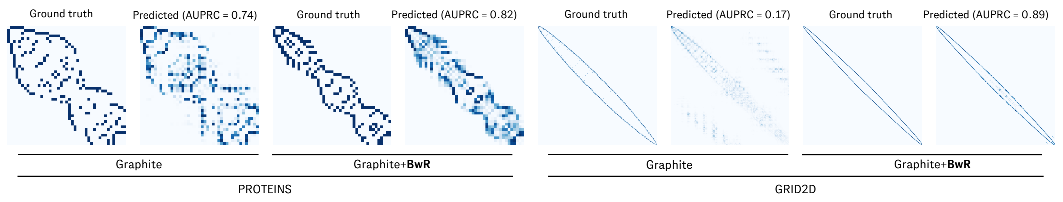

Figure 3: Comparison of Graphite reconstructions with and without BwR on samples from PROTEINS (left) and Grid2D (right). The addition of BwR improves reconstruction quality and AUPRC of generated graphs by constraining the bandwidth.

the others. Notably, the dataset with the largest graphs, DD (mean 277.7 nodes per graph), could not be trained using standard EDP-GNN due to out-of-memory (OOM) issues. In contrast, we are able to train our EDP-GNN+BwR, thus highlighting its lower memory complexity. We observe that, in several GraphRNN and Graphite experiments, the mean MMD is comparable between the standard and the BwR versions, but the AUPRC is significantly better in our version (*e.g.*, on ENZYMES). This highlights how BwR improves the accuracy of the reconstructed graphs even when bulk statistical distributions are comparable. We also show examples of reconstructed adjacency matrices in Figure 3. In the PROTEINS example (left), the standard model must predict edges in a much larger and imbalanced output space. In the Grid2d example (right), the standard model predicts edges far outside the bandwidth, which is inherently impossible with BwR. Overall, results show that BwR consistently improves or matches sampling quality at a fraction of the time/memory cost.

4.3 Molecular Generation

Datasets. We evaluate our models on real-world molecular datasets to show the benefits of BwR for *de novo* molecular generation. We consider two datasets. *ZINC250k* (Irwin et al., 2012) includes 249,455 drug-like small molecules extracted from the ZINC database, averaging 23.14 heavy atoms (nodes) each. *Peptides-func* (Dwivedi et al., 2022) is a recently published dataset of peptide structures that includes 15,535 molecules, averaging 150.94 heavy atoms (nodes) each. Compared to common small-molecule benchmarks, this dataset includes significantly larger molecular graphs and functional motifs (amino acids). Therefore, it allows us to test the advantages of a reduced time complexity and output space given by our bandwidth-constrained generation. Additional datasets detailes are provided in Appendix A.4.3.

Results. Table 3 shows the results on molecular graph generation. BwR achieves superior or competitive performance on both datasets across all methods. GraphRNN+BwR significantly improves reconstruction accuracy with a comparable generation quality with respect to the standard baseline. Graphite+BwR leads to a significantly better generation quality with comparable reconstruction accuracy with respect to the standard baseline. Finally, EDP-GNN+BwR, achieves comparable results with respect to its standard counterpart. However, we remark that, even when generation results are comparable, our approach still significantly reduces time/memory complexity.

5 Conclusion

We presented BwR, a novel approach to reduce the time/space complexity and output space of graph generative models. Leveraging the observation that many real-world graphs have low graph bandwidth, our method restricts the bandwidth of the adjacency matrices during training and generation. Our method is compatible with virtually all existing graph generative models, and we described its application to autoregressive, VAE-based, and score-based models. Our extensive results on both synthetic, biological, and chemical datasets showed that our strategy consistently achieves superior or comparable generation quality compared to the standard methods, while significantly reducing time/space complexity. Currently, our strategy leverages random Cuthill-McKee orderings to reduce the bandwidth. Future work will explore other—potentially even learned—bandwidth-minimizing orderings, while further theoretically studying the distribution of orderings induced by our approach. Future work will also extend our strategy to additional settings, such as conditional generation, larger graphs, and new state-of-the-art models.

REFERENCES

Matej Balog, Bart van Merriënboer, Subhodeep Moitra, Yujia Li, and Daniel Tarlow. Fast training of sparse graph neural networks on dense hardware. *arXiv preprint arXiv:1906.11786*, 2019.

Lukas Biewald. Experiment tracking with weights and biases, 2020. URL `https://www.wandb.com/`. Software available from wandb.com.

Wing-Man Chan and Alan George. A linear time implementation of the reverse cuthill-mckee algorithm. *BIT Numerical Mathematics*, 20(1):8–14, 1980.

Xiaohui Chen, Xu Han, Jiajing Hu, Francisco Ruiz, and Liping Liu. Order matters: Probabilistic modeling of node sequence for graph generation. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 1630–1639. PMLR, 18–24 Jul 2021.

Elizabeth Cuthill and James McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference*, pp. 157–172, 1969.

Marek Cygan and Marcin Pilipczuk. Exact and approximate bandwidth. *Theoretical Computer Science*, 411(40-42):3701–3713, 2010.

Hanjun Dai, Azade Nazi, Yujia Li, Bo Dai, and Dale Schuurmans. Scalable deep generative modeling for sparse graphs. In *International Conference on Machine Learning*, pp. 2302–2312. PMLR, 2020.

Paul D. Dobson and Andrew J. Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771–783, 2003. ISSN 0022-2836. doi: https://doi.org/10.1016/S0022-2836(03)00628-4.

Vijay Prakash Dwivedi, Ladislav Rampášek, Mikhail Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. *arXiv:2206.08164*, 2022.

Uriel Feige. Coping with the np-hardness of the graph bandwidth problem. In *Scandinavian Workshop on Algorithm Theory*, pp. 10–19. Springer, 2000.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

Nikhil Goyal, Harsh Vardhan Jain, and Sayan Ranu. Graphgen: a scalable approach to domain-agnostic labeled graph generation. In *Proceedings of The Web Conference 2020*, pp. 1253–1263, 2020.

Aditya Grover, Aaron Zweig, and Stefano Ermon. Graphite: Iterative generative modeling of graphs. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2434–2444. PMLR, 09–15 Jun 2019.

Xiaojie Guo and Liang Zhao. A systematic survey on deep generative models for graph generation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Eng. Bull.*, 40(3):52–74, 2017.

Dan Hendrycks and Kevin Gimpel. Gaussian Error Linear Units (GELUs), July 2020. arXiv:1606.08415 [cs].

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 6840–6851. Curran Associates, Inc., 2020.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020.

John J Irwin, Teague Sterling, Michael M Mysinger, Erin S Bolstad, and Ryan G Coleman. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52 (7):1757–1768, 2012.

Zhihao Jia, Sina Lin, Mingyu Gao, Matei Zaharia, and Alex Aiken. Improving the accuracy, scalability, and performance of graph neural networks with roc. *Proceedings of Machine Learning and Systems*, 2:187–198, 2020.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *International conference on machine learning*, pp. 2323–2332. PMLR, 2018.

Thomas N. Kipf and Max Welling. Variational graph auto-encoders, 2016.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Greg Landrum. RDKit: Open-source cheminformatics, 2006. URL `https://rdkit.org`.

Michelle M Li, Kexin Huang, and Marinka Zitnik. Graph representation learning in biomedicine and healthcare. *Nature Biomedical Engineering*, pp. 1–17, 2022.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs, 2018.

Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. *Advances in neural information processing systems*, 32, 2019.

Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, and Alexander Gaunt. Constrained graph variational autoencoders for molecule design. *Advances in neural information processing systems*, 31, 2018.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Tengfei Ma, Jie Chen, and Cao Xiao. Constrained generation of semantically valid graphs via regularizing variational autoencoders. *Advances in Neural Information Processing Systems*, 31, 2018.

Burkhard Monien. The bandwidth minimization problem for caterpillars with hair length 3 is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 7(4):505–512, 1986.

Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020.

Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models, 2021.

Chenhao Niu, Yang Song, Jiaming Song, Shengjia Zhao, Aditya Grover, and Stefano Ermon. Permutation invariant graph generation via score-based generative modeling. In Silvia Chiappa and Roberto Calandra (eds.), *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pp. 4474–4484. PMLR, 26–28 Aug 2020.

Ch H Papadimitriou. The np-completeness of the bandwidth minimization problem. *Computing*, 16 (3):263–270, 1976.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Ida Schomburg, Antje Chang, Christian Ebeling, Marion Gremse, Christian Heldt, Gregor Huhn, and Dietmar Schomburg. Brenda, the enzyme database: updates and major new developments. *Nucleic acids research*, 32(suppl_1):D431–D433, 2004.

Chence Shi, Minkai Xu, Zhaocheng Zhu, Weinan Zhang, Ming Zhang, and Jian Tang. Graphaf: a flow-based autoregressive model for molecular graph generation. In *International Conference on Learning Representations*, 2020.

Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In Věra Kůrková, Yannis Manolopoulos, Barbara Hammer, Lazaros Iliadis, and Ilias Maglogiannis (eds.), *Artificial Neural Networks and Machine Learning – ICANN 2018*, pp. 412–422, Cham, 2018. Springer International Publishing. ISBN 978-3-030-01418-6.

N Kipf Thomas and Max Welling. Variational graph auto-encoders.(2016). In *Neural Information Processing Systems Workshop on Bayesian Deep Learning*, 2016.

Jonathan S Turner. On the probable performance of heuristics for bandwidth minimization. *SIAM journal on computing*, 15(2):561–580, 1986.

Walter Unger. The complexity of the approximation of the bandwidth problem. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280)*, pp. 82–91. IEEE, 1998.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Robin Winter, Frank Noé, and Djork-Arné Clevert. Permutation-invariant variational autoencoder for graph-level representation learning. *Advances in Neural Information Processing Systems*, 34: 9559–9573, 2021.

Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. GraphRNN: Generating realistic graphs with deep auto-regressive models. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5708–5717. PMLR, 10–15 Jul 2018.

A Appendix

A.1 Bandwidth Details

We provide a definition of bandwidth through the graph bandwidth problem (Unger, 1998). Intuitively, the graph bandwidth problem can be seen as placing the nodes of a graph on a line such that the length of the longest edge in the graph is minimized. The bandwidth of the graph is then simply the length of the longest edge. Given a graph $G = (\mathcal{V}, \mathcal{E})$ on N vertices, each ordering $\pi : \mathcal{V} \to [1, N]$ defines a graph linearization. We define the *distance* between nodes v_i and v_j in the ordering π as $\text{dist}_\pi(v_i, v_j) = |\pi(v_i) - \pi(v_j)|$. The bandwidth of the ordering π is defined as the maximum stretch of any edge on the linearization, *i.e.* $\varphi(\pi) = \max_{(v_i,v_j)\in\mathcal{E}} \text{dist}_\pi(v_i, v_j)$. The bandwidth of a graph G is the minimum bandwidth across all possible orderings, *i.e.*:

$$\varphi(G) = \min_{\pi:\mathcal{V}\to[1,N]} \varphi(\pi) \tag{4}$$

The graph bandwidth problem has been shown to be NP-hard for general graphs (Papadimitriou, 1976), and also for simpler classes of graphs such as trees and even caterpillar trees (Monien, 1986).

Exact polynomial solutions exist for very restricted classes of graphs, and approximate superpolynomial solutions have been proposed for general graphs (Feige, 2000; Cygan & Pilipczuk, 2010). However, in practice, efficient heuristic approaches work well for general graphs and are routinely leveraged in applications. The Cuthill-McKee (C-M) algorithm (1969), which is based on a variation of BFS search, has linear time complexity $\mathcal{O}(|\mathcal{E}|)$ (Chan & George, 1980), and has been extensively studied from a theoretical perspective (Turner, 1986).

A.2 Bandwidth Visualization

In Figure 4, we show the adjacency matrix A and the bandwidth $\varphi(A)$ for a random set of 10 molecules from ZINC250k. The RDKit ordering is computed using the canonical atom ranking provided by the RDKit library. For the BFS, DFS and C-M, we randomly sample 100 orderings and plot the one with the highest φ (this approximates the output space needed to correctly model the graph for each ordering).

In Figure 5, we show the distribution of bandwidth of ZINC250k adjacency matrices using the C-M algorithm and the canonical SMILES order.

A.3 Model Details

Each model was re-implemented and somewhat modified to facilitate the pairwise comparison with and without the BwR modification.

A.3.1 Optimization

All models were trained for 100 epochs of 30 training batches and nine validation batches. The batch size was fixed at 32. The AdamW optimizer (Loshchilov & Hutter, 2019) was used with a cosine annealed learning rate (Loshchilov & Hutter, 2017). The initial learning rate was hyperoptimized (Appendix A.3.2), and the weight decay parameter was either set to zero or hyperoptimized. GraphRNN and Graphite were trained using binary cross entropy to measure reconstruction accuracy. EDP-GNN was trained using mean squared error loss.

A.3.2 Hyperoptimization

Hyperparameters were separately optimized for each combination of node order and dataset. The hyperparameters were chosen to minimize mean validation MMD^2 in the case of GraphRNN and EDP-GNN, and MMD^2 $-$ AUPRC in the case of Graphite. All hyperoptimizations used 20 outer-loop steps of the Weights and Biases (Biewald, 2020) Bayesian hyperoptimizer. For the specific hyperparameter ranges, see the model details below.

A.3.3 GraphRNN

GraphRNN (You et al., 2018) is an autoregressive model for generating adjacency matrices. We used the GraphRNN-S variant which uses an MLP to predict a whole row at once of the adjacency matrix from the RNN's hidden state.

GraphRNN data pre-processing. GraphRNN was trained using teacher forcing to autogressively predict the next row of the re-parameterized adjacency matrices $A^{\text{opt}} \in N \times \hat{\varphi}_{\text{data}}$ (Figure 1, bottom). In order to prepare the data, a row of zeros was prepended and appended to each A^{opt} to serve as the initial inputs and final outputs for the model. Next, a column with an indicator for whether the row was the first or last was prepended. This resulted in training data of the form:

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & A^{\text{opt}}_{0,0} & \cdots & A^{\text{opt}}_{0,\hat{\varphi}_{\text{data}}} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & A^{\text{opt}}_{N,0} & \cdots & A^{\text{opt}}_{N,\hat{\varphi}_{\text{data}}} \\ 1 & 0 & \cdots & 0 \end{bmatrix}. \tag{5}$$

In order to train the model, the data were placed into `PackedSequence` objects in PyTorch (Paszke et al., 2019), enabling batched training with variable sequence lengths.

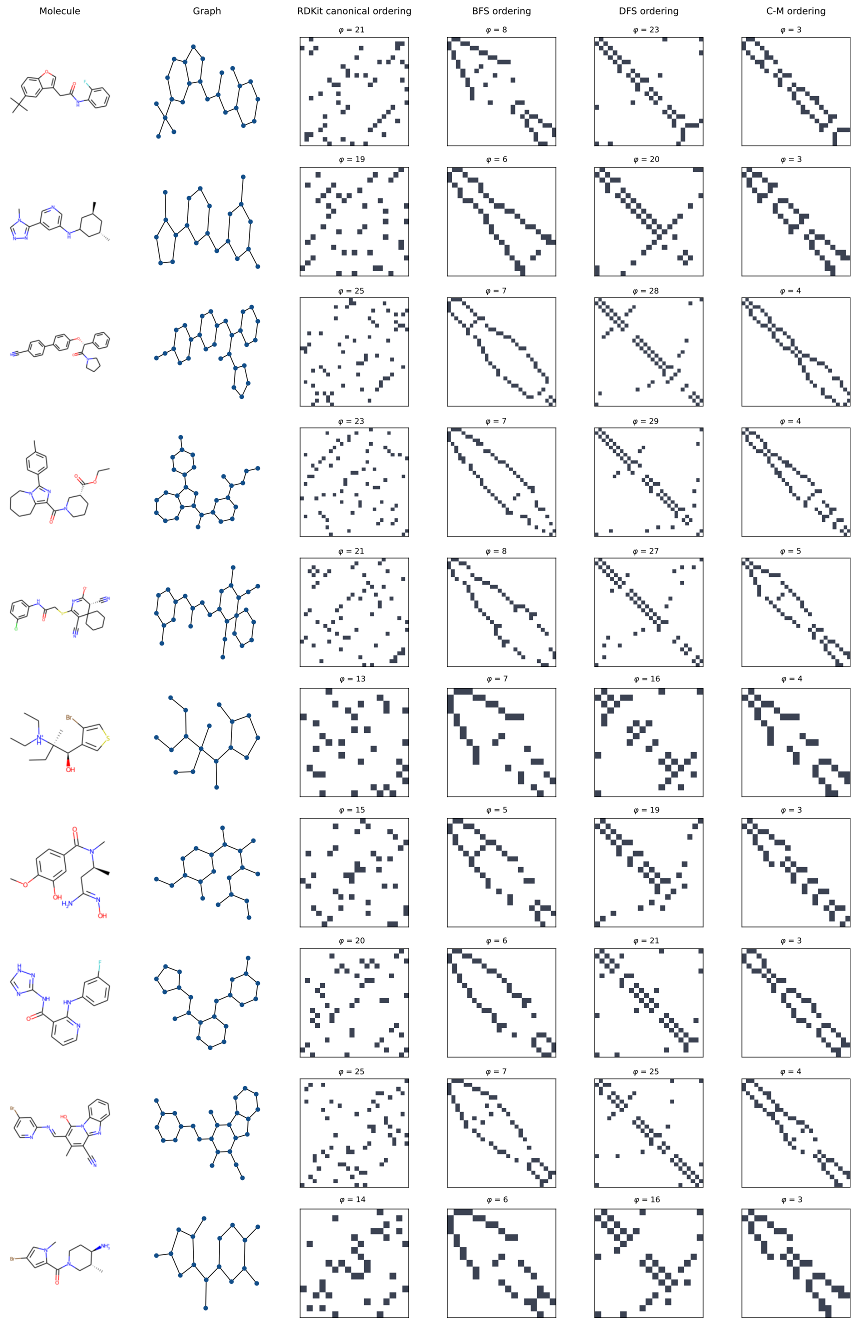

Figure 4: Adjacency matrix and bandwidth φ for different orderings (canonical RDKit, BFS, DFS and Cuthill-McKee) for a random set of 10 molecules from ZINC250k.

GraphRNN architecture. The architecture is a two layer MLP followed by a four layer GRU followed by two layer MLP:

GraphRNN layers

1. `Linear`($\hat{\varphi}_{\text{data}} + 1 \mapsto 128$)
2. `BatchNorm1D`
3. `ReLU`
4. `Linear`($128 \mapsto 128$)

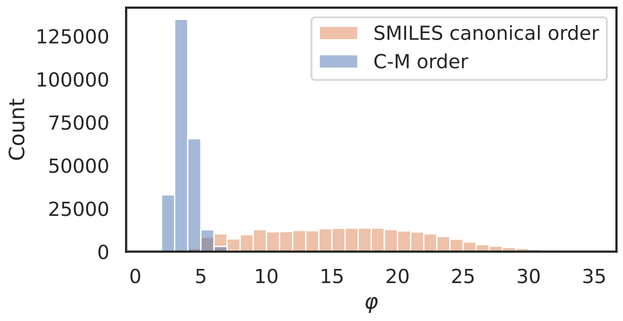

Figure 5: Distribution of bandwidth of ZINC250k adjacency matrices using the C-M algorithm and the canonical SMILES order.

5. `GRU`(4 layers, 128 $\mapsto$ 128)
6. `Linear`(128 $\mapsto$ 128)
7. `BatchNorm1D`
8. `ReLU`
9. `Linear`(128 $\mapsto \hat{\varphi}_{\text{data}} + 1$)

GraphRNN hyperoptimization. The GraphRNN hyperparameter ranges were:

- Learning rate $\sim$ Log Uniform $[10^{-4}, 10^{-2}]$
- Weight decay $\sim$ Log Uniform $[10^{-5}, 10^{-1}]$.

GraphRNN sampling. Rows of the data matrix constructed in Eq. 5 were sampled according to the logits ℓ output at the last layer of the model adjusted by a temperature parameter, τ. That yielded row i edge probabilities:

$$p[(v_i, v_j) \in \mathcal{E}] \sim \text{Bernoulli}\left(\frac{1}{\tau}\ell_{i,j+1}\right). \tag{6}$$

τ was selected for each model to minimize mean MMD on the validation data. The sampling process was halted when an indicator was sampled.

A.3.4 Graphite

Graphite (Grover et al., 2019) is a VAE adapted for graph data with one set of latent variables per node. Graphite predicts edge probabilities using a pairwise kernel between node representations at the end of the decoder. We kept the general design while making a few architectural changes for simplicity and performance.

Graphite data pre-processing. For every graph $G = (\mathcal{V}, \mathcal{E})$ the node order was selected using either BFS (standard variation) or C-M (BwR variation). In the C-M case, we also found the bandwidth resulting from the order $\hat{\varphi}$. We constructed node features $X \in \mathbb{R}^{N \times 16}$ using transformer-style positional encodings (Vaswani et al., 2017). In the original work, Grover et al. (2019) used one-hot positional encodings when there were no node features. For each graph, an edge set was constructed for the decoder model. In the BFS case, the edge set corresponding to a fully connected graph was used. In the C-M case, the edge set for a graph with bandwidth $\hat{\varphi}$ was defined as in Eq. 2.

Graphite architecture. The architecture was implemented using PyTorch Geometric (Fey & Lenssen, 2019). With respect to the original implementation, we used `GINE` layers (Hu et al., 2020) rather than graph convolutions, and GELU activation (Hendrycks & Gimpel, 2020) rather than ReLU. Each `GINE` layer contained a two-layer MLP with the following architecture:

`GINE MLP` layers with hidden dimension h

1. `Linear`$(h \mapsto 2h)$
2. `BatchNorm1D`
3. `GELU`
4. `Linear`$(2h \mapsto h)$
5. `BatchNorm1D`
6. `GELU`.

We made use of a stack of `GINE` layers, which we refer to as `GINEStack` (Algorithm 1). The Graphite encoder was a `GINEStack` with $h = 32$. The variational marginals $\boldsymbol{\mu}, \boldsymbol{\sigma}$ were 32-dimensional and computed using a single linear layer each from the output of the encoder. The decoder also employed a `GINEStack` with $h = 32$ with a few modifications (Algorithm 2). The most consequential change in our experiments was replacing the final edge probability layer. Rather than a dot product as in the original implementation, we observed better performance and lower variance with a two-layer MLP that takes as input concatenated pairs of node embeddings (Algorithm 2, final three lines).

Algorithm 1: `GINEStack`

```
Inputs: node features $X \in \mathbb{R}^{N\times d}$, edge list $\mathcal{E}$, edge features $E \in \mathbb{R}^{|\mathcal{E}|\times k}$ hidden dimension $h$
Outputs: updated node features $\hat{X}$
$X = \texttt{Linear}(d \mapsto h)(X)$
$X = \texttt{BatchNorm1D}(X)$
$X = \texttt{GELU}(X)$
$X_0 = \texttt{GINE}(X, \mathcal{E}, E)$
$X_1 = \texttt{GINE}(X_0, \mathcal{E}, E)$
$X_2 = \texttt{GINE}(X_1, \mathcal{E}, E)$
$\hat{X} = [X_0|X_1|X_2]$
```

Algorithm 2: Graphite decoder. $\circ$ denotes function composition.

```
Inputs: node features $X \in \mathbb{R}^{N\times 16}$, embeddings $Z \in \mathbb{R}^{N\times 32}$, edge list $\mathcal{E}$, edge features $E \in \mathbb{R}^{|\mathcal{E}|\times k}$
Outputs: edge probability logits $\ell \in \mathbb{R}^{|\mathcal{E}|}$
$P = \texttt{Linear}(16 \mapsto 32)(X)$
$P = \texttt{GELU}(\texttt{BatchNorm1D})(P))$
$X_0 = Z + P$
$X_1 = \texttt{GINEStack}(X_0, \mathcal{E}, E)$
$X_2 = [P|X_1]$
$X_3 = \texttt{GELU} \circ \texttt{BatchNorm1D} \circ \texttt{Linear}(112 \mapsto 32)(X_2)$
$K = [X|X_3]$
$\ell^1_{i,j} = \texttt{Linear}(32 \mapsto 1) \circ \texttt{GELU} \circ \texttt{Linear}(96 \mapsto 32)[K_i|K_j]$
$\ell^2_{i,j} = \texttt{Linear}(32 \mapsto 1) \circ \texttt{GELU} \circ \texttt{Linear}(96 \mapsto 32)[K_j|K_i]$    # $j, i$ order to preserve symmetry
$\ell_{i,j} = \ell^1_{i,j} + \ell^2_{i,j}$
```

Graphite loss function. The loss was the standard VAE loss function with a hyperoptimized weight β on the KL divergence term:

$$\mathcal{L}(\mathcal{E}, \ell, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \frac{\beta}{|\mathcal{E}|} \sum_{i=1}^{N} \left(\boldsymbol{\mu}^2 + \boldsymbol{\sigma}^2 - \log[\boldsymbol{\sigma}] - \frac{1}{2} \right)_i + \frac{1}{|\mathcal{E}|} \texttt{BCE}(\ell, \mathcal{E}) \tag{7}$$

where ℓ denotes the model predicted edge logits and `BCE` is the binary cross entropy.

Graphite hyperoptimization. The Graphite hyperparameter ranges were:

- Learning rate $\sim$ Log Uniform $[10^{-4}, 10^{-2}]$
- KL-divergence weight (β) $\sim$ Log Uniform $[1, 10^{-5}]$.

Graphite sampling. The latent variables Z were sampled independently from the standard normal distribution. Edge probabilities were then sampled from the decoder's edge probabilities (Eq. 1, Algorithm 2). The number of nodes and bandwidths used to construct the decoder's message passing graph were sampled from the empirical distribution of the training data.

A.3.5 EDP-GNN

EDP-GNN (Niu et al., 2020) is a permutation invariant score-based generative model for graphs. Niu et al. (2020) used annealed Langevin dynamics to sample from their model and used a variance schedule with six time steps. We switched to the DDPM (Ho et al., 2020) framework, which we found led to more reliable results and faster sampling in preliminary experiments.

EDP-GNN data pre-processing. For every graph $G = (\mathcal{V}, \mathcal{E})$ a node order was selected using either BFS (standard variation) or C-M (BwR variation). In the C-M case, we also found the bandwidth $\hat{\varphi}$. We then constructed an edge set $\mathcal{E}'$ of edges not in the original graph, which the model was trained to distinguish from the real edges. In the BFS case, these were all of the edges not in $\mathcal{E}$, i.e. $\mathcal{E}' = \{(i,j)\forall i \neq j\} - \mathcal{E}$. In the C-M case, these were the edges included in a graph with $\varphi(G) = \hat{\varphi}$ (Eq. 2), and not in $\mathcal{E}$, that is, $\mathcal{E}' = \hat{\mathcal{E}} - \mathcal{E}$. Edge features $E \in \mathbb{R}^{|\mathcal{E}|+|\mathcal{E}'|}$ were constructed to encode whether each edge is in $\mathcal{E}$ or $\mathcal{E}'$ with 1 to indicate $\in \mathcal{E}$ and -1 to indicate $\in \mathcal{E}'$. Node features $X \in \mathbb{R}^{N\times 16}$ were constructed using transformer-style positional encodings (Vaswani et al., 2017). Time embedding features T used for time conditioning were constructed using 128-dimensional positional encodings.

EDP-GNN diffusion hyperparameters. We used a cosine variance schedule (Nichol & Dhariwal, 2021) with 200 steps. The effect of this schedule on a restricted adjacency matrix of a planar graph is shown in Figure 6. We used the noise predicting parameterization $\boldsymbol{\epsilon}_\theta$ introduced by Ho et al. (2020).

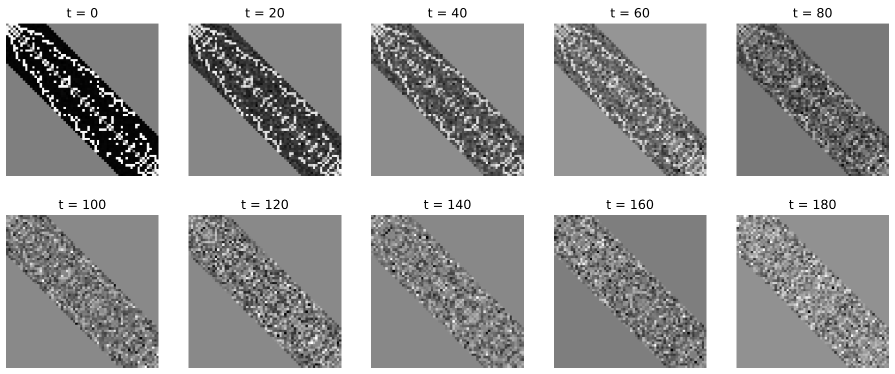

Figure 6: Visualization of cosine variance schedule forward diffusion with 200 steps on a planar graph with restricted bandwidth.

EDP-GNN architecture. Our implementation of EDP-GNN was built using the previously introduced `GINEStack` (Algorithm 1) followed by a two layer MLP operating on edge features pairs of node representations to predict the sampled noise $\boldsymbol{\epsilon}$. The resultant architecture for the molecular datasets is shown in Algorithm 3. We used a node embedding size of 64 for the generic datasets instead of 128.

EDP-GNN hyperoptimization. The learning rate was hyperoptimized with a distribution $\sim$ Log Uniform $[10^{-4}, 10^{-2}]$.

EDP-GNN sampling. We used the DDPM sampling algorithm (Ho et al., 2020).

A.4 Datasets

All datasets were filtered so that there was one connected component per example.

Algorithm 3: Modified EDP-GNN architecture. $\circ$ denotes function composition.

```
**Inputs**: node features $X \in \mathbb{R}^{N\times 16}$, edge list $\mathcal{E}_\cup = \mathcal{E} \cup \mathcal{E}'$, edge features $E \in \mathbb{R}^{|\mathcal{E}_\cup|}$, time embeddings $T \in \mathbb{R}^{128}$
**Outputs**: noise predictions $\boldsymbol{\epsilon}_\theta \in \mathbb{R}^{|\mathcal{E}\cup\mathcal{E}'|}$
$P = \texttt{GELU} \circ \texttt{Linear}(16 \mapsto 128)(X)$
$T_0 = \texttt{GELU} \circ \texttt{Linear}(128 \mapsto 128)(T)$ $X_0 = T_0 + P$
$X_1 = \texttt{GINEStack}(X_0, \mathcal{E}_\cup, E)$
$X_2 = [P|T_0|X_1]$
$X_3 = \texttt{GELU} \circ \texttt{BatchNorm1D} \circ \texttt{Linear}(768 \mapsto 128)(X_2)$
$K = [X|X_3]$
$E^0 = \texttt{GELU} \circ \texttt{Linear}(1 \mapsto 128)(E)$
$\epsilon^1_{i,j} = \texttt{Linear}(128 \mapsto 1) \circ \texttt{GELU} \circ \texttt{Linear}(384 \mapsto 128)[K_i|K_j|E^0_{i,j}]$
$\epsilon^2_{i,j} = \texttt{Linear}(128 \mapsto 1) \circ \texttt{GELU} \circ \texttt{Linear}(384 \mapsto 128)[K_j|K_i|E^0_{j,i}]$    # $j, i$ order to preserve symmetry
$\boldsymbol{\epsilon}_{\theta,i,j} = \epsilon^1_{i,j} + \epsilon^2_{i,j}$
```

A.4.1 Example Datasets for Table 1

All datasets, except Peptides-func, are available through the TUDataset collection (Morris et al., 2020). Peptides-func is available in the Long Range Graph Benchmark (Dwivedi et al., 2022).

A.4.2 Generic Datasets

Community2. For each graph the number of nodes N was sampled uniformly between 60 and 160. Each community was then generated using an Erdos-Renyi model with edge probability 0.3. Then edges between the two communities were sampled with probability 0.05. Finally, the largest connected component of the resultant graph was selected.

Planar. For each graph 64 2D node coordinates were sampled uniformly between zero and one. A Delaunay triangulation was performed on these coordinates. Two nodes were considered adjacency if they shared a vertex in the triangulation.

Grid2d. All unique pairs of side lengths between 10 and 20 were enumerated. For each side length pair, a 2D grid graph was generated. Since this yielded only 66 graphs, each graph in the training and validation sets were included five times with different random BFS and C-M orders each time.

DD. The DD graphs were filtered so that each had between 100 and 500 nodes as in (You et al., 2018), going from 1178 to 918 graphs.

Enzymes. The enzymes graphs were filtered so that each had $10 \leq N \leq 125$ going from 600 to 556 graphs.

Proteins. The proteins graphs were filtered so that each had $10 \leq N \leq 125$ going from 1113 to 904 graphs.

A.4.3 Molecular Datasets

ZINC250k. No filtering was required.

Peptides-func. Removing graphs with more than one connected component filtered 15535 graphs down to 15375.

		COMMUNITY2						PLANAR						GRID2D					
		Deg.	Cluster	Orbit	Spectra	Mean ↓	AUPRC ↑	Deg.	Cluster	Orbit	Spectra	Mean ↓	AUPRC ↑	Deg.	Cluster	Orbit	Spectra	Mean ↓	AUPRC ↑
GraphRNN	Standard	**0.016**	**0.046**	**0.024**	0.024	**0.028**	0.376	**0.056**	**0.309**	0.617	**0.08**	0.265	0.545	0.403	0.0	0.714	0.174	0.323	0.642
	BwR [ours]	**0.034**	**0.041**	**0.017**	**0.006**	**0.024**	**0.421**	**0.06**	**0.311**	**0.481**	**0.079**	**0.233**	**0.652**	**0.037**	0.797	**0.066**	**0.061**	**0.24**	**0.999**
Graphite	Standard	0.146	**0.047**	**0.015**	**0.011**	0.055	0.706	**0.289**	**0.304**	**0.749**	**0.104**	**0.361**	0.975	0.5	**1.299**	0.601	0.198	0.649	0.453
	BwR [ours]	**0.114**	**0.047**	**0.015**	**0.013**	**0.047**	**0.747**	**0.311**	0.323	1.108	**0.128**	**0.468**	**0.99**	**0.069**	1.969	**0.038**	**0.035**	**0.528**	**0.915**
EDP-GNN	Standard	**0.037**	**0.056**	**0.02**	**0.008**	**0.03**	-	**0.229**	**0.4**	**1.121**	**0.086**	**0.459**	-	**0.428**	**1.378**	**0.661**	**0.113**	**0.645**	-
	BwR [ours]	**0.066**	**0.048**	**0.032**	**0.014**	**0.04**	-	**0.179**	**0.377**	**1.249**	**0.092**	**0.474**	-	**0.415**	**1.452**	**0.392**	**0.101**	**0.59**	-

		DD						ENZYMES						PROTEINS					
		Deg.	Cluster	Orbit	Spectra	Mean ↓	AUPRC ↑	Deg.	Cluster	Orbit	Spectra	Mean ↓	AUPRC ↑	Deg.	Cluster	Orbit	Spectra	Mean ↓	AUPRC ↑
GraphRNN	Standard	**0.066**	**0.155**	**0.41**	**0.065**	**0.174**	**0.299**	0.011	0.045	**0.021**	**0.018**	0.023	0.445	**0.004**	**0.04**	**0.015**	**0.01**	**0.017**	0.533
	BwR [ours]	**0.092**	**0.229**	**0.489**	**0.125**	**0.234**	**0.318**	**0.003**	**0.039**	**0.01**	**0.014**	**0.016**	**0.602**	0.012	0.045	**0.011**	0.013	**0.02**	**0.716**
Graphite	Standard	0.316	**0.316**	**0.656**	**0.186**	**0.368**	0.804	0.204	**0.046**	0.099	0.078	0.107	0.925	0.293	**0.096**	0.111	0.114	0.153	**0.95**
	BwR [ours]	**0.239**	**0.245**	**0.492**	**0.118**	**0.273**	**0.84**	**0.042**	**0.038**	**0.052**	**0.018**	**0.038**	**0.95**	**0.037**	**0.043**	**0.052**	**0.016**	**0.037**	0.933
EDP-GNN	Standard	OOM	OOM	OOM	OOM	OOM	OOM	**0.098**	**0.069**	0.159	**0.042**	**0.092**	-	0.119	0.064	0.082	0.045	0.077	-
	BwR [ours]	**0.184**	**0.208**	**0.738**	**0.065**	**0.299**	-	**0.027**	**0.033**	**0.036**	**0.013**	**0.027**	-	**0.027**	**0.038**	**0.021**	**0.012**	**0.024**	-

Table 4: Graph generation results on generic datasets, extended. **Bold** indicates best results compared to the other model of the same type and dataset. Significance was determined by Welch's t-test with five replicates per model. Models are considered comparable when $p \geq 0.05$. Graph statistics (degree, cluster, orbit, spectra) are reported as MMD^2. Mean computed across individual statistics for each model/dataset. OOM denotes out-of-memory issues. Hyphen (–) denotes not applicable metric/model.

A.5 Additional Experimental Results

A.5.1 Generic Graph Generation

Extended results with individual metrics are shown in Table 4.