Improving Graph Generation by Restricting Graph Bandwidth

Nathaniel Diamant [1] **Alex M. Tseng** [1] **Kangway V. Chuang** [1] **Tommaso Biancalani** [1] **Gabriele Scalia** [1]

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). Computational approaches to capture graph statistical properties have long been established (Albert & Barabási, 2002), whereas deep generative modeling has recently been proven capable of learning both global and fine-grained structural properties, along with their complex interdependencies (Guo & Zhao, 2023). These results suggest many promising applications for deep graph generative modeling, which include the biomedical and pharmaceutical domains, where essential objects such as molecules, gene networks, and cell-level tissue organization 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.

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

[1]Department of Artificial Intelligence and Machine Learning, Research and Early Development, Genentech, USA. Correspondence to: Nathaniel Diamant <diamant.nathaniel@gene.com>, Gabriele Scalia <scalia.gabriele@gene.com>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

[1]https://github.com/Genentech/bandwidth-graph-generation

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. We describe our strategy in the context of three recent models: an autoregressive model based on GraphRNN (You et al., 2018), a one-shot VAE-based model based on Graphite (Grover et al., 2019), and a one-shot score-based model based on EDP-GNN (Niu et al., 2020).

We experimentally validate BwR by evaluating the reconstruction accuracy and generation quality on both synthetic and real datasets. Additionally, we analyze memory and time improvements. We include molecular datasets—spanning both small molecules and larger peptides—to evaluate the advantages of BwR for *de novo* molecular generation. Our results show that BwR consistently achieves superior or competitive generative performance to the standard baselines, at a fraction of the time/space complexity.

Contributions. We summarize our main contributions as follows:

- We show that many real-world classes of graphs, such as molecules, have 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 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. Related Work

2.1. 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 $\mathcal{G} = \{G_1, \dots, G_S\} \sim p(G)$, where each graph $G_i = (\mathcal{V}_i, \mathcal{E}_i)$ is defined by its set of nodes $\mathcal{V}_i = (v_1, \dots, v_N)$ 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, factorizing $p_{\text{model}}(G)$ into the joint probability of its components (e.g., nodes or motifs). Node-based autoregressive methods (You et al., 2018; Shi et al., 2020) generate each node and its edges in a predefined order, conditioning each new node on the already-generated graph, with time complexity and output space in $\mathcal{O}(N^2)$. Solutions have been proposed to separately output actions corresponding to the number of new edges and their identities for each new node, reducing time complexity (Li et al., 2018; Liu et al., 2018). However, these methods increase model complexity without reducing the output space. In contrast, BwR reduces *both* the time complexity and output space of existing node-based autoregressive models to be in $\mathcal{O}(N \cdot \hat{\varphi})$. This reduction is achieved without losing expressiveness and without increasing model complexity.

One-shot models sample the whole topology of the graph from a latent distribution. Many one-shot models use permutation-invariant functions to output the graph topology. This class of generative models can be split into two main categories: adjacency-matrix-based models (Ma et al., 2018; Niu et al., 2020) directly output A^π, while node-embedding-based models (Kipf & Welling, 2016; Grover et al., 2019) sample node embeddings from the latent distribution and compute A^π based on pairwise relationships between them. In both cases, these methods have time complexity and output space in $\mathcal{O}(N^2)$, as they need to consider edges between every pair of nodes. In contrast, BwR allows reducing both the time complexity and output space to be in $\mathcal{O}(N \cdot \hat{\varphi})$, with no loss of expressiveness.

Last, we notice how 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. For the

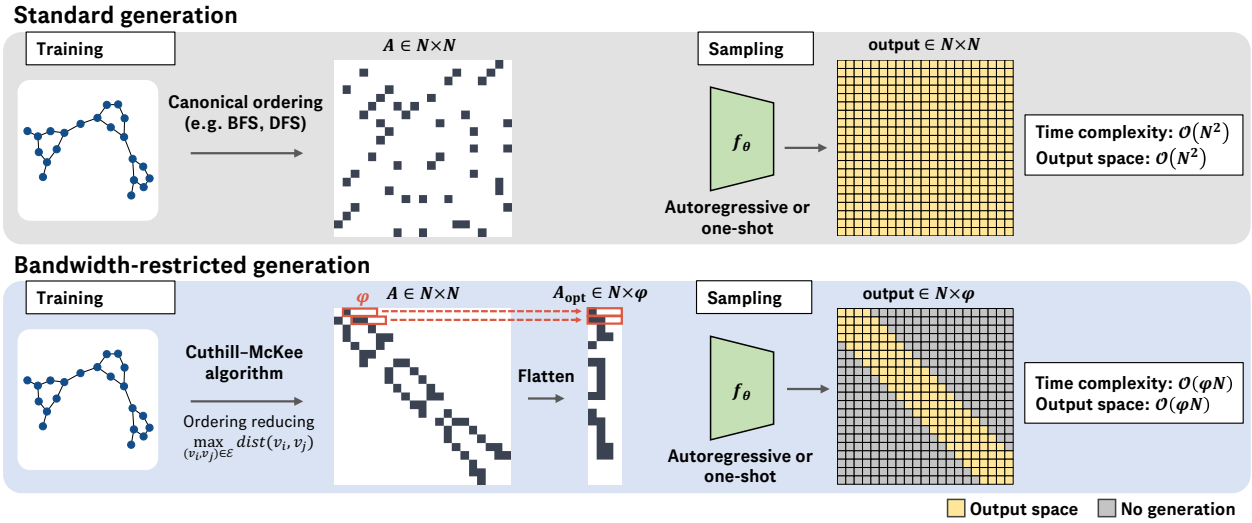

Figure 1. Overview of our strategy and comparison with standard generation methods. **(Top)** In a standard graph generative method, the model is trained on adjacency matrices A derived through a specific canonical ordering on the graph (e.g., BFS or DFS). During sampling, the model needs to predict 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 $\mathcal{O}\left(N \times \varphi(A)\right)$.

present work, we focus on node-based generation, but our approach can be extended to coarser motif-based generation (Liao et al., 2019).

2.2. Graph Ordering

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). Given that our method imposes a specific node ordering, it is related to other works that have investigated ordering in graph generation.

Ordering is crucial in autoregressive models. For any particular ordering π, if the index distance $|\pi\left(v_i\right) - \pi\left(v_j\right)|$ between two connected nodes $\left(v_i, v_j\right) \in \mathcal{E}$ is high, the model is required to handle long-term dependencies. This issue can be addressed by choosing a specific canonical ordering. For example, GraphRNN (You et al., 2018) is trained using random breadth-first-search (BFS) node orderings for each graph, such that new nodes can only be connected to existing nodes at the frontier of the BFS. GraphRNN and its comparison with our bandwidth-restricted version are further discussed in Section 4.2.

It has been shown that the choice of node ordering impacts graph generation for specific applications. For example, in the context of autoregressive molecular generation, BFS had clear advantages over depth-first-search (DFS), even though the latter is more often used to define a canonical order for molecular structures (Mercado et al., 2021).

A more fundamental issue raised by non-unique graph representations is that choosing a specific ordering π does not rigorously correspond to maximum-likelihood estimation (MLE), thus preventing exact likelihood evaluation (Liao et al., 2019; Chen et al., 2021). Additionally, it can make the reconstruction loss ambiguous. As discussed by Liao et al. (2019), training on random orderings in a specific canonical family (e.g., BFS) optimizes a variational lower bound of the true log-likelihood tighter than any single arbitrary ordering. For autoregressive models, a tighter lower bound is derived by Chen et al. (2021) by performing approximate posterior inference over the node ordering. For one-shot models, Winter et al. (2021) addresses reconstruction ambiguity by training a permuter model to reorder generated graphs alongside a standard encoder/decoder architecture. We notice how 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. This integration will be investigated in future work.

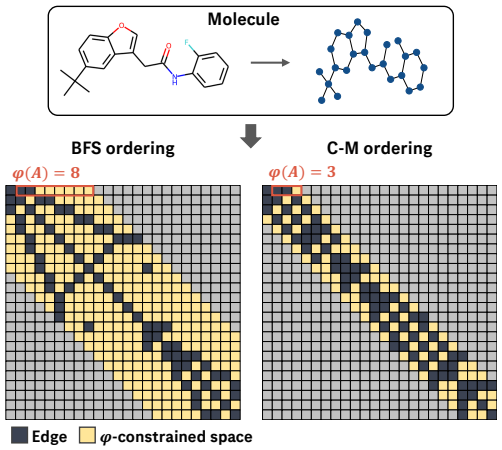

Figure 2. Bandwidth of two adjacency matrices of the same molecular graph. The left adjacency matrix is given by a BFS ordering, the right adjacency matrix is given by a Cuthill-McKee ordering.

3. Graph Bandwidth Background

We start providing 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.:

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

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), 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 super-polynomial 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.

One such heuristic is 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). Extensions of this algorithm and other heuristics have been proposed that improve efficiency and/or theoretical guarantees, though results are dataset dependent (Gonzaga de Oliveira et al., 2018). For the present work, we leverage the C-M algorithm initialized to start at a pseudo peripheral node (Gibbs et al., 1976). However, our proposed strategy is independent of the choice of the bandwidth-minimization algorithm, and other approaches—potentially even learned—will be explored in future work. Interestingly, as discussed in Section 2, BFS has been shown to be superior to DFS for autoregressive molecular generation. Given that C-M can be seen as a special case of BFS, using C-M allows us to retain the benefits of BFS, while also further reducing $\max_{(v_i,v_j)\in\mathcal{E}} \text{dist}_\pi(v_i, v_j)$. We define the bandwidth of the adjacency matrix derived with the C-M algorithm as $\hat{\varphi}$.

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 (see Appendix C for more details on the datasets). For each dataset, a *savings factor* summarizes the space reduction. The savings factor is calculated as 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 savings factor > 3 for most small-molecule datasets (e.g., 3.8 ± 1.0 for ZINC250k). The savings 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 (KKI, OHSU). The high savings factor of molecular datasets is due to the existence of an intrinsic upper bound on the bandwidth of molecule-like graphs, which we derive and further discuss in Appendix E.

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

Table 1. Number of nodes, C-M bandwidth, and savings factor of a set of chemical and biological datasets. The first section of the table consists of small molecules, the second section consists of large molecules, and the third section consists of brain networks. See Appendix C for more details on the datasets. Average and standard deviation calculated across graphs for each dataset.

Dataset	N	$\hat{\varphi}$	Savings Factor
ZINC250k	23.2 ± 4.5	3.3 ± 0.8	3.9 ± 1.0
AIDS	14.0 ± 10.4	3.1 ± 1.1	2.4 ± 1.0
Alchemy	10.1 ± 0.7	2.8 ± 0.7	2.1 ± 0.4
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
NCI-H23	25.8 ± 10.3	4.0 ± 1.3	3.5 ± 1.2
OVCAR-8	25.8 ± 10.3	4.0 ± 1.3	3.5 ± 1.2
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
SN12C	25.8 ± 10.3	4.0 ± 1.3	3.5 ± 1.2
Tox21	16.8 ± 10.1	3.0 ± 1.2	3.0 ± 1.3
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

graphs given by BFS, DFS, RDKit (Landrum, 2006), and C-M orderings are presented in Figure 4 (Appendix). 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).

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.

4. Bandwidth-Restricted Graph Generation

In this section we describe BwR, our novel approach for improving graph generation. As BwR can be combined with different existing generative methods, we first describe its general strategy and principles in Section 4.1. Then, we detail the strategies for bandwidth-restricted graph generation applied to an autoregressive model based on GraphRNN (You et al., 2018) in Section 4.2, and two distinct one-shot models based on Graphite (Grover et al., 2019) and EDP-GNN (Niu et al., 2020) in Section 4.3.

4.1. Restricting Graph Bandwidth

Starting from the observation that many real-world graphs have low bandwidth (Section 3), we propose to reduce the output space of a graph generative model from $N \times N$ to $N \times \hat{\varphi}$[2]. As the goal of graph 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^* computed through 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 using 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.

4.2. 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. This can be viewed as generating the adjacency matrix row-by-row or block-by-block. 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

[2]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.

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. In particular, we focus on the GraphRNN-S variant (additional model details are provided in Appendix B.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 across all 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 (corresponding to the savings factor, Table 1).

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 918 protein graphs, the authors set $d = M_{\text{data}} = 230$, whereas we derive $d = \hat{\varphi}_{\text{data}} = 122$, a nearly two-fold reduction.

4.3. 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; Kipf & 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:

$$\boldsymbol{\mu}, \boldsymbol{\sigma} = \text{GNN}_\phi(A, X), \tag{2}$$

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:

$$p[(v_i, v_j) \in \mathcal{E}] = \text{GNN}_\theta(\hat{A}, X, Z)_{i,j}, \tag{3}$$

where $\hat{A}$ is fully-connected and Z are samples from $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$. Further architectural details are described in Appendix B.4. In our bandwidth-restricted version of Graphite, we constrain the bandwidth of $\hat{A}$. For a graph G, we get the bandwidth $\varphi(A^\pi)$ for π found using the C-M algorithm and build a new edge set:

$$\hat{\mathcal{E}} = \{(i, j) \mid 1 \le |i - j| \le \varphi(A^\pi)\}, \tag{4}$$

where $1 \le i, j \le N$. With the new edge set, the decoder message passing steps are reduced from complexity $\mathcal{O}(N^2)$ 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:

$$\mathbf{s}_\theta(A)_{i,j} = \text{MLP}(\hat{A}_{ij}). \tag{5}$$

To constrain the bandwidth, we restrict $(i, j) \in \hat{\mathcal{E}}$ (Eq. 4) 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 B.5.

5. Experiments

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

5.1. Metrics

We measure two goals of the models: closeness of the sample distribution to the true distribution of graphs, and reconstruction quality.

To measure the quality of the sample distribution, we use two metrics. First, 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 following Liao et al. (2019). To track overall sample quality, we compute the mean MMD[2] across all four MMD metrics[3]. Additionally, as recommended by recent work on evaluation metrics for graph generative modeling (Thompson et al., 2022), we use a precision–recall metric (Kynkäänniemi et al., 2019) which accounts for both sample quality and variety. We report the harmonic mean of precision and recall as F1-PR.

Reconstruction quality is measured by comparing the reconstructed graph and the original test graph using the Area Under the Precision–Recall Curve (AUPRC). For GraphRNN, we compare the reconstructed row and the original row; for Graphite, we compare the reconstructed graph and the original graph[4]. We chose to use AUPRC because it does not depend on true negatives and because it is well suited to class imbalance (Davis & Goadrich, 2006). Additionally, we report the estimated log-likelihood for test graphs. Although log-likelihood has known limitations for evaluating (graph) generative models (Thompson et al., 2022), it is well suited (in combination with the other metrics) to demonstrating the benefits of BwR 's reduced output space.

5.2. Experimental Setup

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 4 and additional details are provided in Appendix B. Each model architecture is individually hyperoptimized (details in Appendix B.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).

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

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

5.3. 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 C.2.

Results. Table 2 summarizes the results for generic graph generation. As shown, our approach consistently achieves superior or competitive performance across datasets and methods as measured by mean MMD, F1-PR and/or AUPRC. GraphRNN+BwR and Graphite+BwR outperform their standard counterparts in five out of six datasets, with comparable performance (i.e., not statistically significant or mixed) on the sixth. BwR improves EDP-GNN generation quality in four out of six datasets, with comparable performance on 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 issues. In contrast, we are able to train our EDP-GNN+BwR, thus highlighting its lower memory complexity. We observe a statistically significant improvement in AUPRC given by BwR in almost all the experiments, thus showing how our approach improves the accuracy of the reconstructed graphs even when bulk statistical distributions are comparable. Still, BwR results in an improvement in the quality of the sample distribution (MMD and/or F1-PR) in the majority of the experiments, with comparable performance in the others. We observe a significant improvement in log-likelihood compared to standard models across all the experiments.

We 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 generation quality at a fraction of the time/memory cost.

5.4. 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:

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 5 (Appendix). OOM denotes out-of-memory issues. Hyphen (–) denotes not applicable metric/model.

		COMMUNITY2				PLANAR				GRID2D			
		↓ MMD^2	↑ F1-PR	↑ $\ell\ell$	↑ AUPRC	↓ MMD^2	↑ F1-PR	↑ $\ell\ell$	↑ AUPRC	↓ MMD^2	↑ F1-PR	↑ $\ell\ell$	↑ AUPRC
GraphRNN	Standard	**0.028**	**0.648**	-1990	0.376	0.265	**0.029**	-400.0	0.545	0.323	0.250	-540.0	0.642
	BwR [ours]	**0.024**	**0.729**	**-1940**	**0.421**	**0.233**	**0.139**	**-309.0**	**0.652**	**0.240**	**0.909**	**-36.4**	**0.999**
Graphite	Standard	0.055	**0.507**	-3560	0.706	**0.361**	**0.010**	**-595.0**	0.975	0.649	0.051	-2320	0.453
	BwR [ours]	**0.047**	**0.423**	**-3450**	**0.747**	**0.468**	**0.023**	**-554.0**	**0.990**	**0.528**	**0.666**	**-358.0**	**0.915**
EDP-GNN	Standard	**0.030**	**0.621**	-211000	-	**0.459**	0.172	-57800	-	**0.645**	**0.548**	-622000	-
	BwR [ours]	**0.040**	**0.581**	**-168000**	-	**0.474**	**0.450**	**-36400**	-	**0.590**	**0.528**	**-97900**	-
		DD				ENZYMES				PROTEINS			
		↓ MMD^2	↑ F1-PR	↑ $\ell\ell$	↑ AUPRC	↓ MMD^2	↑ F1-PR	↑ $\ell\ell$	↑ AUPRC	↓ MMD^2	↑ F1-PR	↑ $\ell\ell$	↑ AUPRC
GraphRNN	Standard	**0.174**	**0.426**	-1460	**0.299**	0.023	**0.948**	-199.0	0.445	**0.017**	**0.971**	-173.0	0.533
	BwR [ours]	**0.234**	**0.579**	**-1400**	**0.318**	**0.016**	**0.963**	**-177.0**	**0.602**	**0.020**	**0.964**	**-117.0**	**0.716**
Graphite	Standard	**0.368**	**0.003**	-4360	0.804	0.107	0.459	-204.0	0.925	0.153	0.523	-220.0	**0.950**
	BwR [ours]	**0.273**	**0.008**	**-3430**	**0.840**	**0.038**	**0.916**	**-164.0**	**0.950**	**0.037**	**0.889**	**-166.0**	0.933
EDP-GNN	Standard	OOM	OOM	OOM	OOM	**0.092**	0.726	-18000	-	0.077	0.782	-23400	-
	BwR [ours]	**0.299**	**0.106**	**-269000**	-	**0.027**	**0.914**	**-7320**	-	**0.024**	**0.944**	**-7590**	-

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 $p \geq 0.05$. 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 5 (Appendix). Hyphen (–) denotes not applicable metric/model.

		ZINC250K				PEPTIDES-FUNC			
		↓ MMD^2	↑ F1-PR	↑ $\ell\ell$	↑ AUPRC	↓ MMD^2	↑ F1-PR	↑ $\ell\ell$	↑ AUPRC
GraphRNN	Standard	**0.038**	**0.803**	-38.9	0.668	**0.031**	**0.477**	-168.0	0.809
	BwR [ours]	**0.029**	**0.900**	**-31.0**	**0.783**	**0.033**	**0.526**	**-112.0**	**0.903**
Graphite	Standard	0.153	0.178	-57.0	**0.999**	0.182	**0.031**	-742.0	**0.987**
	BwR [ours]	**0.084**	**0.511**	**-39.4**	0.995	**0.120**	**0.186**	**-362.0**	**0.993**
EDP-GNN	Standard	**0.106**	0.172	-7510	-	**0.115**	**0.041**	-634000	-
	BwR [ours]	**0.143**	**0.750**	**-3180**	-	**0.143**	**0.109**	**-32200**	-

(1) *ZINC250k* (Irwin et al., 2012) includes 249,455 drug-like small molecules extracted from the ZINC database, averaging 23.14 heavy atoms (nodes) each.
(2) *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 C.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 or better reconstruction accuracy with respect to the standard baseline. Finally, EDP-GNN+BwR, achieves significantly improved generation quality on ZINC250k, and comparable results on Peptides-func. We observe a significant improvement in log-likelihood compared to standard models across all the experiments. We remark that, even when generation quality is comparable, our approach still significantly reduces time/memory complexity.

5.5. Memory and Time Efficiency

We evaluate whether the lower time complexity and output space reduction actually translate into reduced time and memory footprint. For this analysis, we consider all the datasets and models previously introduced, for a total of 24 experiments. We measure the following metrics: (1) *Mem-*

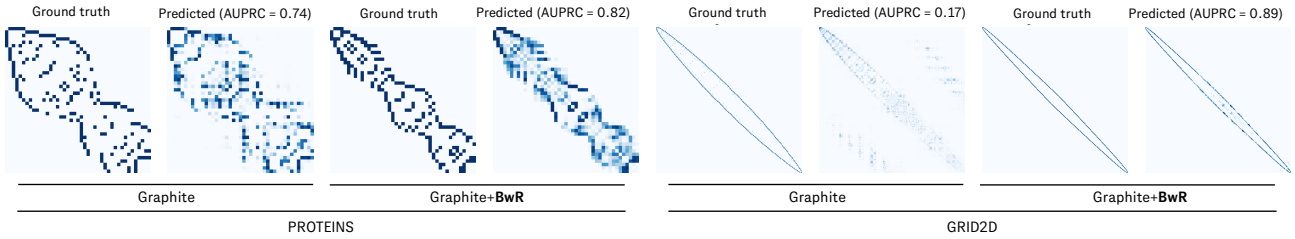

Figure 3. Comparison of Graphite reconstructions with and without BwR on samples from PROTEINS (left) and Grid2D (right).

Table 4. Computational cost results. **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 $p \geq 0.05$. OOM denotes out-of-memory issues.

		COMMUNITY2		PEPTIDES-FUNC		GRID2D		DD	
		↓ sample (s)	↓ mem. (GB)	↓ sample (s)	↓ mem. (GB)	↓ sample (s)	↓ mem. (GB)	↓ sample (s)	↓ mem. (GB)
GraphRNN	Standard	**0.673**	**0.059**	**7.690**	**0.082**	**0.490**	**0.115**	**1.470**	**0.142**
	BwR [ours]	**0.538**	**0.061**	**7.680**	**0.080**	**0.520**	**0.112**	**1.500**	**0.149**
Graphite	Standard	1.460	0.805	3.520	1.740	6.400	3.270	10.90	5.870
	BwR [ours]	**1.050**	**0.553**	**0.268**	**0.170**	**0.842**	**0.447**	**2.380**	**1.510**
EDP-GNN	Standard	12.50	1.600	69.80	6.280	50.80	6.730	OOM	OOM
	BwR [ours]	**8.410**	**1.080**	**4.880**	**0.547**	**6.550**	**0.837**	**19.90**	**2.620**

ory usage, as the maximum GPU utilization during a training batch; (2) *Training time*, as the average wall time to train on one batch; (3) *Sample time*, as the average wall time to sample 256 graphs from the generative model.

Table 4 shows memory usage and sample time for four datasets. The remaining results are included in Table 6 (Appendix). As shown, BwR significantly reduces memory usage across all datasets and all methods besides GraphRNN[5], up to a factor of 11x for larger graphs. Additionally, it improves sample time (up to a factor of 13x-14x) in 14 out of 24 experiments (with comparable results in the others) and training time in 8 out of 24 experiments (with comparable results in the others). Overall, BwR achieves a consistent reduction in computational costs.

5.6. Impact of Output Space Reduction to Performance

We analyze the relationship between the savings factor, which summarizes the space reduction given by the C-M bandwidth reparameterization (Table 1), and the performance/computational improvement. Results are included in Appendix D.3. Results suggest that we can estimate the empirical improvement given by BwR for a specific dataset in advance, without actually training a model, by computing simple statistical properties of the dataset.

[5]We believe that GraphRNN has close to the same time/memory usage with and without BwR because most of the computation happens in the hidden layers of the GRU as opposed to the single linear readout layer which predicts the next row.

6. 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.

Acknowledgements

We thank the anonymous reviewers for their suggestions which have further strengthened the conclusions of the paper. We thank members of the AI/ML department in Genentech for helpful feedback and discussions. All authors are employees of Genentech, Inc. and shareholders of Roche.

References

Albert, R. and Barabási, A.-L. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

Balog, M., van Merriënboer, B., Moitra, S., Li, Y., and Tarlow, D. Fast training of sparse graph neural networks on dense hardware. *arXiv preprint arXiv:1906.11786*, 2019.

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

Böttcher, J., Pruessmann, K. P., Taraz, A., and Würfl, A. Bandwidth, expansion, treewidth, separators and universality for bounded-degree graphs. *European Journal of Combinatorics*, 31(5):1217–1227, 2010.

Chan, W.-M. and George, A. A linear time implementation of the reverse Cuthill-McKee algorithm. *BIT Numerical Mathematics*, 20(1):8–14, 1980.

Chen, X., Han, X., Hu, J., Ruiz, F., and Liu, L. Order matters: probabilistic modeling of node sequence for graph generation. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 1630–1639. PMLR, 2021.

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

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

Dai, H., Nazi, A., Li, Y., Dai, B., and Schuurmans, D. Scalable deep generative modeling for sparse graphs. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 2302–2312. PMLR, 2020.

Davis, J. and Goadrich, M. The relationship between Precision-Recall and ROC curves. In *International Conference on Machine Learning*, pp. 233–240. Association for Computing Machinery, 2006.

Dobson, P. D. and Doig, A. J. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771–783, 2003.

Dwivedi, V. P., Rampášek, L., Galkin, M., Parviz, A., Wolf, G., Luu, A. T., and Beaini, D. Long range graph benchmark. In *Advances in Neural Information Processing Systems*, volume 35, pp. 22326–22340. Curran Associates, Inc., 2022.

Eppstein, D. Isometric diamond subgraphs. In *Graph Drawing: 16th International Symposium, GD 2008, Heraklion, Crete, Greece, September 21-24, 2008. Revised Papers 16*, pp. 384–389. Springer, 2009.

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

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

Gibbs, N. E., Poole, W. G., and Stockmeyer, P. K. An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM Journal on Numerical Analysis*, 13 (2):236–250, 1976.

Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-Guzik, A. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

Gonzaga de Oliveira, S. L., Bernardes, J. A., and Chagas, G. O. An evaluation of low-cost heuristics for matrix bandwidth and profile reductions. *Computational and Applied Mathematics*, 37(2):1412–1471, 2018.

Goyal, N., Jain, H. V., and Ranu, S. GraphGen: a scalable approach to domain-agnostic labeled graph generation. In *Proceedings of The Web Conference 2020*, pp. 1253–1263, 2020.

Grover, A., Zweig, A., and Ermon, S. Graphite: iterative generative modeling of graphs. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2434–2444. PMLR, 2019.

Guo, X. and Zhao, L. A systematic survey on deep generative models for graph generation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(5):5370–5390, 2023.

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

Hendrycks, D. and Gimpel, K. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, volume 33, pp. 6840–6851. Curran Associates, Inc., 2020.

Hu, W., Liu, B., Gomes, J., Zitnik, M., Liang, P., Pande, V., and Leskovec, J. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020.

Irwin, J. J., Sterling, T., Mysinger, M. M., Bolstad, E. S., and Coleman, R. G. ZINC: a free tool to discover chemistry for biology. *Journal of Chemical Information and Modeling*, 52(7):1757–1768, 2012.

Jia, Z., Lin, S., Gao, M., Zaharia, M., and Aiken, A. Improving the accuracy, scalability, and performance of graph neural networks with ROC. *Proceedings of Machine Learning and Systems*, 2:187–198, 2020.

Jin, W., Barzilay, R., and Jaakkola, T. Junction tree variational autoencoder for molecular graph generation. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2323–2332. PMLR, 2018.

Kipf, T. N. and Welling, M. Variational graph auto-encoders. In *Neural Information Processing Systems Workshop on Bayesian Deep Learning*, 2016.

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

Kynkäänniemi, T., Karras, T., Laine, S., Lehtinen, J., and Aila, T. Improved precision and recall metric for assessing generative models. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

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

Li, M. M., Huang, K., and Zitnik, M. Graph representation learning in biomedicine and healthcare. *Nature biomedical engineering*, 6(12):1353–1369, 2022.

Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018.

Liao, R., Li, Y., Song, Y., Wang, S., Hamilton, W., Duvenaud, D. K., Urtasun, R., and Zemel, R. Efficient graph generation with graph recurrent attention networks. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Liu, Q., Allamanis, M., Brockschmidt, M., and Gaunt, A. Constrained graph variational autoencoders for molecule design. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

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

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

Ma, T., Chen, J., and Xiao, C. Constrained generation of semantically valid graphs via regularizing variational autoencoders. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Mercado, R., Bjerrum, E. J., and Engkvist, O. Exploring graph traversal algorithms in graph-based molecular generation. *Journal of Chemical Information and Modeling*, 62(9):2093–2100, 2021.

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

Morris, C., Kriege, N. M., Bause, F., Kersting, K., Mutzel, P., and Neumann, M. TUDataset: a collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop Graph Representation Learning and Beyond*, 2020.

Nichol, A. Q. and Dhariwal, P. Improved denoising diffusion probabilistic models. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8162–8171. PMLR, 2021.

Niu, C., Song, Y., Song, J., Zhao, S., Grover, A., and Ermon, S. Permutation invariant graph generation via score-based generative modeling. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pp. 4474–4484. PMLR, 2020.

Otachi, Y. and Suda, R. Bandwidth and pathwidth of three-dimensional grids. *Discrete Mathematics*, 311(10-11): 881–887, 2011.

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

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: an imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Schomburg, I., Chang, A., Ebeling, C., Gremse, M., Heldt, C., Huhn, G., and Schomburg, D. BRENDA, the enzyme database: updates and major new developments. *Nucleic acids research*, 32(suppl_1):D431–D433, 2004.

Shi, C., Xu, M., Zhu, Z., Zhang, W., Zhang, M., and Tang, J. GraphAF: a flow-based autoregressive model for molecular graph generation. In *International Conference on Learning Representations*, 2020.

Simonovsky, M. and Komodakis, N. Graphvae: Towards generation of small graphs using variational autoencoders. In Kůrková, V., Manolopoulos, Y., Hammer, B., Iliadis, L., and Maglogiannis, I. (eds.), *Artificial Neural Networks and Machine Learning – ICANN 2018*, pp. 412–422. Springer International Publishing, 2018.

Thompson, R., Knyazev, B., Ghalebi, E., Kim, J., and Taylor, G. W. On evaluation metrics for graph generative models. In *International Conference on Learning Representations*, 2022.

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

Unger, W. 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.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Wester, M. J., Pollock, S. N., Coutsias, E. A., Allu, T. K., Muresan, S., and Oprea, T. I. Scaffold topologies. 2. analysis of chemical databases. *Journal of Chemical Information and Modeling*, 48(7):1311–1324, 2008.

Winter, R., Noe, F., and Clevert, D.-A. Permutation-invariant variational autoencoder for graph-level representation learning. In *Advances in Neural Information Processing Systems*, volume 34, pp. 9559–9573. Curran Associates, Inc., 2021.

You, J., Ying, R., Ren, X., Hamilton, W., and Leskovec, J. GraphRNN: generating realistic graphs with deep auto-regressive models. In *International Conference on Machine Learning*, pp. 5708–5717. PMLR, 2018.

A. 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.

B. Model Details

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

B.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 B.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.

B.2. Hyperoptimization

Hyperparameters were separately optimized for each combination of node order, model, and dataset. The hyperparameters for MMD and AUPRC results were chosen to minimize mean validation MMD^2 in the case of GraphRNN and EDP-GNN, and $\text{MMD}^2 - \text{AUPRC}$ in the case of Graphite. The hyperparameters for log-likelihood and F1-PR results were chosen to maximize F1-PR. 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.

B.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:

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & A^{\text{opt}}_{0,0} & \cdots & A^{\text{opt}}_{0,\hat{\varphi}_{\text{data}}} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & A^{\text{opt}}_{N,0} & \cdots & A^{\text{opt}}_{N,\hat{\varphi}_{\text{data}}} \\ 1 & 0 & \cdots & 0 \end{bmatrix}. \tag{6}$$

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.

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`($128 \mapsto 128$)
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. 6 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:

$$p[(v_i, v_j) \in \mathcal{E}] \sim \text{Bernoulli}\left(\frac{1}{\tau}\ell_{i,j+1}\right). \tag{7}$$

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

B.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. 4.

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, 2016) 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 $\boldsymbol{\mu}, \boldsymbol{\sigma}$ (Eq. 2) 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:

$$\mathcal{L}(\mathcal{E}, \ell, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \frac{\beta}{|\mathcal{E}|} \sum_{i=1}^{N} \left(\boldsymbol{\mu}^2 + \boldsymbol{\sigma}^2 - \log[\boldsymbol{\sigma}] - \frac{1}{2} \right)_i + \frac{1}{|\mathcal{E}|} \texttt{BCE}(\ell, \mathcal{E}), \tag{8}$$

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. 3, 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.

B.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. 4), 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).

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.

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
$\boldsymbol{\epsilon}_{\theta,i,j} = \epsilon^1_{i,j} + \epsilon^2_{i,j}$

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).

B.6. Likelihood Evaluation

To compute the likelihood of test set graphs as an evaluation metric, we use the following strategies. For GraphRNN, we use the auto-regressive log-likelihood. For Graphite, we use the variational ELBO as a lower bound. For EDP-GNN, we use the ELBO for diffusion models, as shown in (Ho et al., 2020).

C. Datasets

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

C.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).

C.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.

C.3. Molecular Datasets

ZINC250k. No filtering was required.

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

D. Additional Experimental Results

D.1. MMD Metrics

We include the individual MMDs results (summarized by the mean MMD in the main text) in Table 5.

D.2. Computational Metrics

We include computational metrics for all datasets in Table 6.

D.3. Impact of Output Space Reduction to Performance Improvements

We study whether the theoretical improvement in space/time complexity given by BwR translates well into an empirical improvement in generation quality and computational complexity. To do this, we analyze the relationship between the savings factor, which summarizes the space reduction given by the C-M bandwidth reparameterization (Table 1), and the performance improvement, measured as the ratio between standard models' metrics and their +BwR extensions. Interestingly, we observe a high correlation (Spearman-r of 0.90 for Graphite, 0.89 for EDP-GNN, and 0.62 for GraphRNN) between the savings ratio and the log-likelihood improvement across the 8 datasets. Additionally, we observe a high correlation (Spearman-r of 0.97 for Graphite and 0.96 for EDP-GNN) between the savings ratio and the improvement in GPU memory usage (Figure 7) across the 8 datasets. This analysis suggests that (1) the theoretical improvement correlates with the empirical advantage, and (2) we can get an estimate of the expected empirical improvement for a specific dataset in advance, without actually training a model.

E. Why Molecular Graphs Have Low Bandwidth?

In this section, we discuss the bandwidth of molecular graphs, i.e. graphs whose nodes and edges describe atoms and bonds, respectively. As shown in Table 1, all the considered molecular datasets have low average empirical bandwidth (in particular, the first section of Table 1 and Peptides-func correspond to molecular graphs). Additionally, the average bandwidth increases slowly as the average number of nodes N in the datasets increases (for the considered datasets, the average bandwidth $\hat{\varphi}$ varies between 2.8 ± 0.7 and 5.7 ± 2.6, while the average number of nodes N varies between 10.1 ± 0.7 and 150.9 ± 84.5). Furthermore, given that the empirical bandwidth is estimated through a heuristic algorithm, part of the increase in $\hat{\varphi}$ for higher N can be explained by a slightly reduced algorithm efficacy for larger graphs.

All these empirical observations motivate more theoretical research on *why* all molecular graphs seem to have restricted bandwidth. To investigate this question, we have derived several upper bounds on the bandwidth of molecular graphs. These bounds show that, indeed, the inherent properties of molecular graphs confer low bandwidth. These results further strengthen the universal validity of BwR, beyond the datasets considered in this paper.

Several bounds to the graph bandwidth for molecular graphs are discussed in the following:

Molecules have planar graphs and bounded max degree. As highlighted in the cheminformatics literature, molecules with non-planar graphs are extremely rare (Wester et al., 2008). In practice, all molecules included in drug-like libraries (e.g., ZINC250k dataset) have planar graphs. Additionally, because of chemical bonding rules, all molecular graphs have a bounded maximum degree (4-6, depending on the atom types in the molecule). The combination of these two properties (bounded degree and planarity) guarantees sub-linear bandwidth, in $\mathcal{O}\left(\frac{N}{\log_\Delta(N)}\right)$, with Δ being the maximum degree, as proved by Böttcher et al. (2010).

Molecules as combination of motifs. A less rigorous and more intuitive explanation comes from the fact that synthesizable molecules tend to consist of small components connected in a few (typically, one to three) places. Under the (simplified) assumption that a molecular graph is a string of connected components, the graph's bandwidth is upper-bounded by the size of the largest component. This can be seen by considering the resulting block-diagonal adjacency matrix: each component corresponds to a block in the adjacency matrix, and the bandwidth of the graph corresponds to the bandwidth of the largest block/component. For example, in a linear molecule (like a long alkane chain), the bandwidth is one (independent of the length of the chain) because each carbon (individual node) is only directly connected to its neighbors.

Crystal structure as molecular upper bound. Another chemistry-inspired upper bound can be derived by considering regular chemical graphs with degree four. If we allow all 4-regular graphs, this is insufficient to constrain the bandwidth since random 4-regular graphs are expander graphs and have bandwidth in $\mathcal{O}\left(\frac{N}{\log(N)}\right)$. Instead, we consider the most densely packed form of carbon, diamond, which is formed by a 3D lattice. The graph of diamond's structure is a subgraph of the 3D grid graph (Eppstein, 2009). Interestingly, the 3D grid graph has bandwidth that scales with the square root of the number of nodes (Otachi & Suda, 2011), providing a tighter upper bound in $\mathcal{O}\left(\sqrt{N}\right)$.

Further research on theoretical upper bounds on the bandwidth of graphs occurring in common domains, such as molecules and other biological objects, will be the subject of future work.

Table 5. Graph generation results with individual MMD values. **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.

		COMMUNITY2				PLANAR				GRID2D			
		↓ Deg.	↓ Cluster	↓ Orbit	↓ Spectra	↓ Deg.	↓ Cluster	↓ Orbit	↓ Spectra	↓ Deg.	↓ Cluster	↓ Orbit	↓ Spectra
GraphRNN	Standard	**0.016**	**0.046**	**0.024**	0.024	**0.056**	**0.309**	0.617	**0.080**	0.403	**0.000**	0.714	0.174
	BwR [ours]	**0.034**	**0.041**	**0.017**	**0.006**	**0.060**	**0.311**	**0.481**	**0.079**	**0.037**	0.797	**0.066**	**0.061**
Graphite	Standard	0.146	**0.047**	**0.015**	**0.011**	**0.289**	**0.304**	**0.749**	**0.104**	0.500	**1.300**	0.601	0.198
	BwR [ours]	**0.114**	**0.047**	**0.015**	**0.013**	**0.311**	0.323	1.110	**0.128**	0.069	**1.970**	0.038	0.035
EDP-GNN	Standard	**0.037**	**0.056**	**0.020**	**0.008**	**0.229**	**0.400**	**1.120**	**0.086**	**0.428**	**1.380**	**0.661**	**0.113**
	BwR [ours]	**0.066**	**0.048**	**0.032**	**0.014**	**0.179**	**0.377**	**1.250**	**0.092**	**0.415**	**1.450**	**0.392**	**0.101**
		DD				ENZYMES				PROTEINS			
		↓ Deg.	↓ Cluster	↓ Orbit	↓ Spectra	↓ Deg.	↓ Cluster	↓ Orbit	↓ Spectra	↓ Deg.	↓ Cluster	↓ Orbit	↓ Spectra
GraphRNN	Standard	**0.066**	**0.155**	**0.410**	**0.065**	0.011	0.045	**0.021**	**0.018**	**0.004**	**0.040**	**0.015**	**0.010**
	BwR [ours]	**0.092**	**0.229**	**0.489**	**0.125**	**0.003**	**0.039**	**0.010**	**0.014**	0.012	0.045	**0.011**	0.013
Graphite	Standard	0.316	**0.316**	**0.656**	**0.186**	0.204	**0.046**	0.099	0.078	0.293	**0.096**	0.111	0.114
	BwR [ours]	**0.239**	**0.245**	**0.492**	**0.118**	**0.042**	**0.039**	**0.052**	**0.018**	**0.037**	**0.043**	**0.052**	**0.016**
EDP-GNN	Standard	OOM	OOM	OOM	OOM	**0.098**	**0.069**	0.159	**0.042**	0.119	0.064	0.082	0.045
	BwR [ours]	**0.184**	**0.208**	**0.738**	**0.065**	**0.027**	**0.033**	**0.036**	**0.013**	**0.027**	**0.038**	**0.021**	**0.012**
		ZINC250K				PEPTIDES-FUNC							
		↓ Deg.	↓ Cluster	↓ Orbit	↓ Spectra	↓ Deg.	↓ Cluster	↓ Orbit	↓ Spectra				
GraphRNN	Standard	0.025	**0.045**	0.012	**0.071**	**0.009**	**0.004**	**0.000**	**0.108**				
	BwR [ours]	**0.011**	**0.044**	**0.005**	**0.057**	**0.008**	**0.001**	**0.001**	**0.123**				
Graphite	Standard	0.049	0.516	0.005	0.044	0.169	**0.235**	**0.030**	**0.293**				
	BwR [ours]	**0.009**	**0.307**	**0.002**	**0.019**	**0.056**	**0.216**	**0.011**	**0.198**				
EDP-GNN	Standard	0.174	**0.055**	0.024	0.170	0.159	**0.041**	0.047	0.213				
	BwR [ours]	**0.015**	0.528	**0.004**	**0.023**	**0.050**	0.371	**0.007**	**0.144**				

Table 6. Computational cost results. **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 $p \geq 0.05$.

		COMMUNITY2			PLANAR			GRID2D			DD		
		↓ sample (s)	↓ mem. (GB)	↓ batch (s)	↓ sample (s)	↓ mem. (GB)	↓ batch (s)	↓ sample (s)	↓ mem. (GB)	↓ batch (s)	↓ sample (s)	↓ mem. (GB)	↓ batch (s)
GraphRNN	Standard	**0.673**	**0.059**	**0.008**	**0.095**	**0.036**	**0.007**	**0.490**	**0.115**	**0.016**	**1.470**	**0.142**	**0.015**
	BwR [ours]	**0.538**	**0.061**	**0.009**	**0.094**	**0.036**	**0.007**	**0.520**	**0.112**	**0.018**	**1.500**	**0.149**	**0.016**
Graphite	Standard	1.460	0.805	0.012	0.513	0.266	**0.010**	6.400	3.270	0.022	10.90	5.870	0.028
	BwR [ours]	**1.050**	**0.553**	**0.010**	**0.242**	**0.135**	**0.009**	**0.842**	**0.447**	**0.012**	**2.380**	**1.510**	**0.013**
EDP-GNN	Standard	12.50	1.600	0.006	4.300	0.526	**0.004**	50.80	6.730	0.016	OOM	OOM	OOM
	BwR [ours]	**8.410**	**1.080**	**0.005**	**2.260**	**0.255**	**0.003**	**6.550**	**0.837**	**0.004**	**19.90**	**2.620**	**0.007**
		ENZYMES			PROTEINS			ZINC250K			PEPTIDES-FUNC		
		↓ sample (s)	↓ mem. (GB)	↓ batch (s)	↓ sample (s)	↓ mem. (GB)	↓ batch (s)	↓ sample (s)	↓ mem. (GB)	↓ batch (s)	↓ sample (s)	↓ mem. (GB)	↓ batch (s)
GraphRNN	Standard	**0.129**	**0.019**	**0.011**	**0.333**	**0.023**	**0.007**	**0.080**	**0.014**	**0.004**	**7.690**	**0.082**	**0.014**
	BwR [ours]	**0.239**	**0.019**	**0.014**	**0.533**	**0.021**	**0.009**	**0.080**	**0.014**	**0.004**	**7.680**	**0.080**	**0.013**
Graphite	Standard	**0.450**	0.085	**0.014**	0.227	0.105	**0.008**	**0.071**	0.040	**0.008**	3.520	1.740	0.015
	BwR [ours]	**0.050**	**0.035**	**0.010**	**0.056**	**0.031**	**0.010**	**0.083**	**0.016**	**0.010**	**0.268**	**0.170**	**0.008**
EDP-GNN	Standard	1.590	0.178	**0.004**	2.040	0.211	**0.003**	1.620	0.145	**0.003**	69.80	6.280	0.024
	BwR [ours]	**0.981**	**0.066**	**0.003**	**1.040**	**0.056**	**0.003**	**0.809**	**0.050**	**0.004**	**4.880**	**0.547**	**0.004**

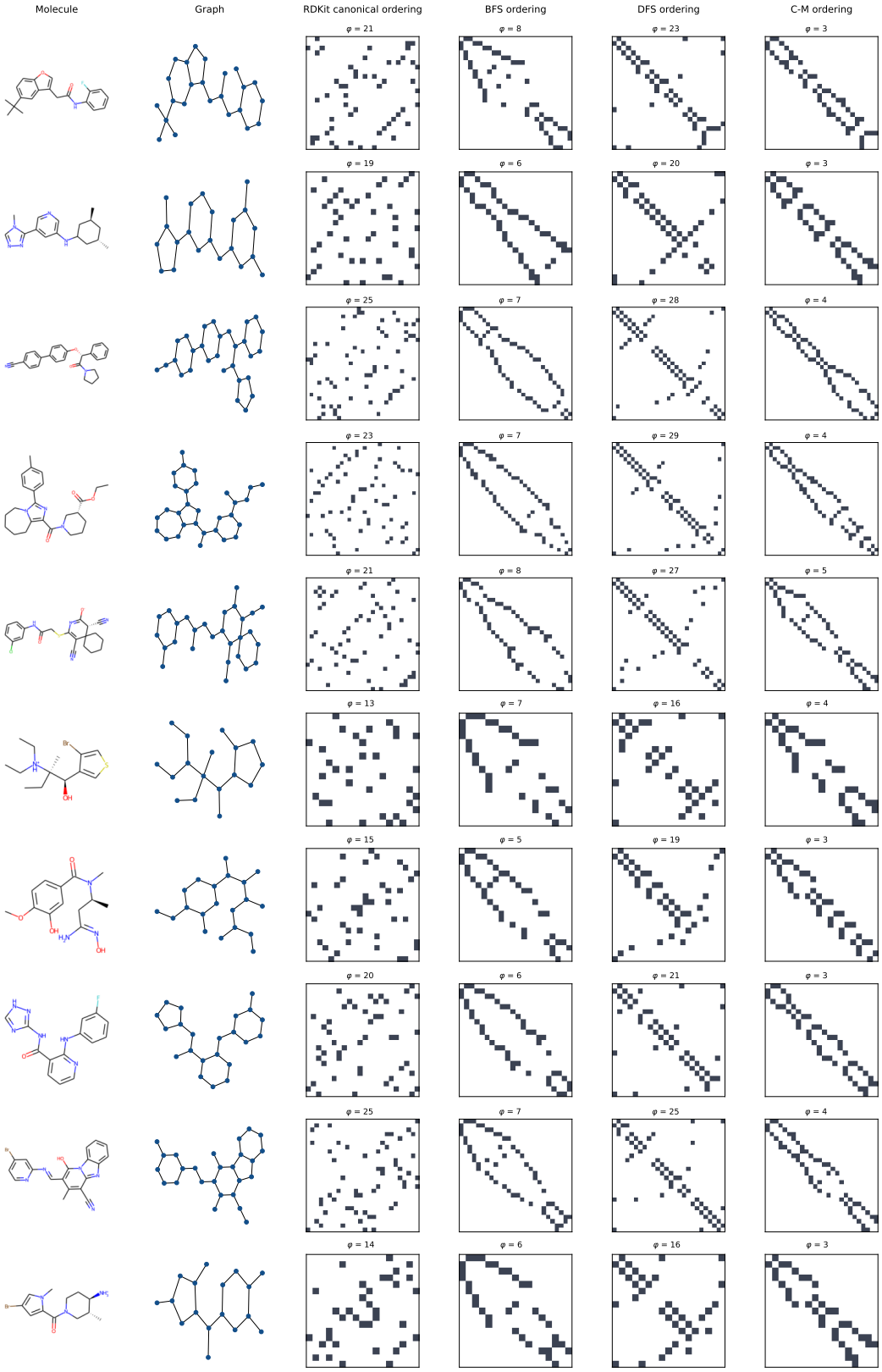

Figure 4. Adjacency matrix and bandwidth φ for different orderings (canonical RDKit, BFS, DFS and Cuthill-McKee) for a random set of 10 molecules from ZINC250k.

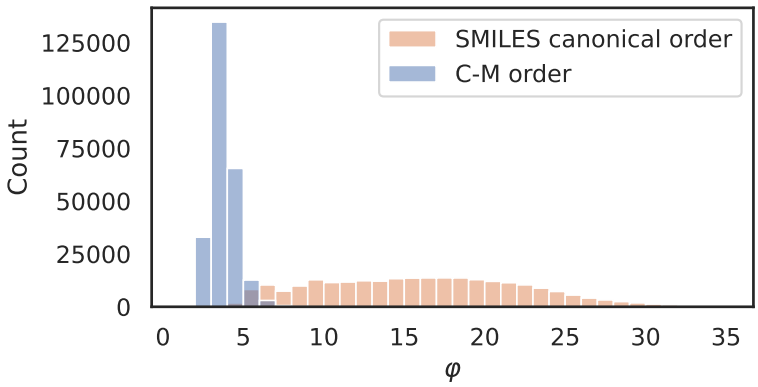

Figure 5. Distribution of bandwidth of ZINC250k adjacency matrices using the C-M algorithm and the canonical SMILES order.

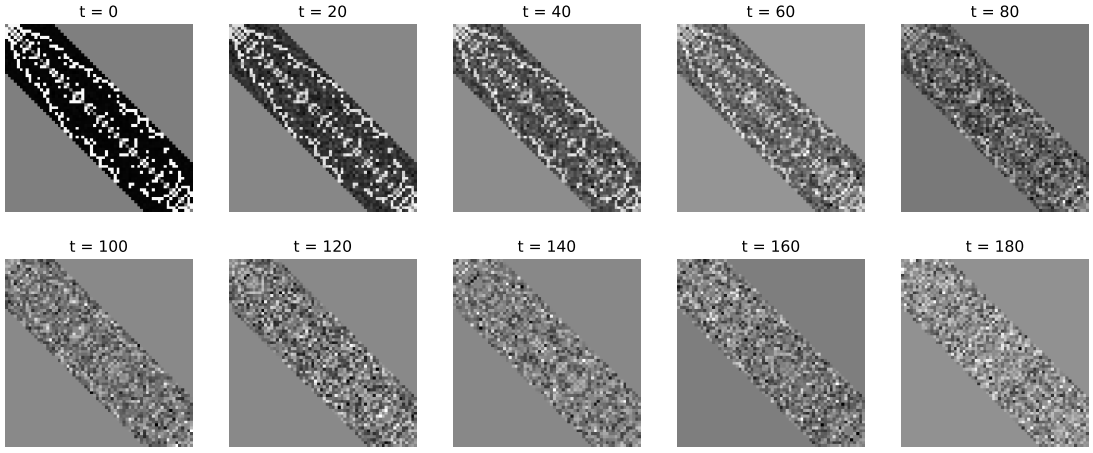

Figure 6. Visualization of cosine variance schedule forward diffusion with 200 steps on a planar graph with restricted bandwidth.

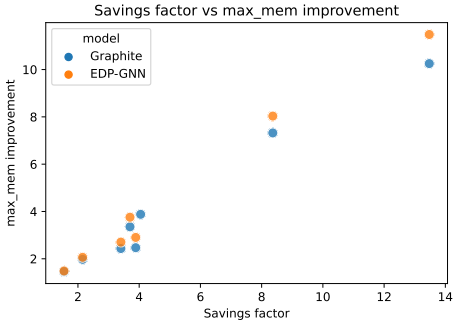

Figure 7. Savings factor versus improvement in memory usage with and without BwR for different datasets, for Graphite and EDP-GNN models.