SPECTRE: Spectral Conditioning Helps to Overcome the Expressivity Limits of One-shot Graph Generators

Karolis Martinkus [1] **Andreas Loukas** [*2] **Nathanaël Perraudin** [*3] **Roger Wattenhofer** [1]

Abstract

We approach the graph generation problem from a spectral perspective by first generating the dominant parts of the graph Laplacian spectrum and then building a graph matching these eigenvalues and eigenvectors. Spectral conditioning allows for direct modeling of the global and local graph structure and helps to overcome the expressivity and mode collapse issues of one-shot graph generators. Our novel GAN, called SPECTRE, enables the one-shot generation of much larger graphs than previously possible with one-shot models. SPECTRE outperforms state-of-the-art deep autoregressive generators in terms of modeling fidelity, while also avoiding expensive sequential generation and dependence on node ordering. A case in point, in sizable synthetic and real-world graphs SPECTRE achieves a 4-to-170 fold improvement over the best competitor that does not overfit and is 23-to-30 times faster than autoregressive generators.

1. Introduction

The ability to generate new samples from a distribution is a central problem in machine learning. Most of the work has focused on data with a regular structure, such as images and audio (Brock et al., 2018; Oord et al., 2016a). For such data, Generative Adversarial Networks (GANs) (Goodfellow et al., 2014; Arjovsky et al., 2017) have emerged as a powerful paradigm, managing to balance generation novelty and fidelity in a manner previously thought impossible.

The present work considers the use of GANs for graph data. The generation of novel graphs is relevant for numerous applications in molecule (Jin et al., 2018; De Cao & Kipf, 2018; Liu et al., 2018), protein (Huang et al., 2016), network, and circuit design (Mirhoseini et al., 2021). Yet, despite recent efforts, the problem remains largely unresolved: current state-of-the-art approaches are constrained to small graphs and often fail to strike a beneficial trade-off between capturing the essential properties of the training distribution and exhibiting high novelty. We argue that the one-shot generators used in current GAN models face expressivity issues that hinder them from capturing the global graph properties. We refer to a model as one-shot if it generates all edges between nodes at once. In contrast, auto-regressive models build a graph by progressively adding new nodes and edges.

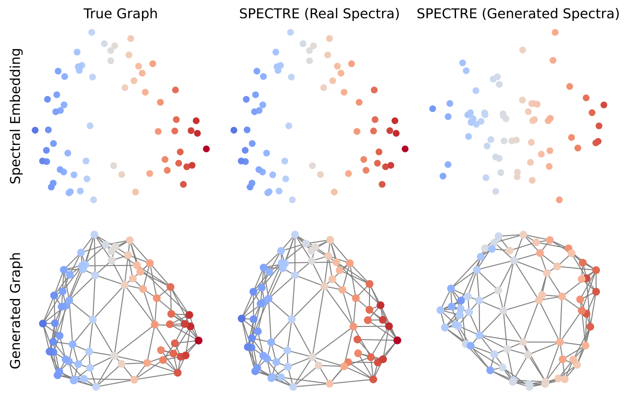


Figure 1. Generating the spectrum first allows SPECTRE to control the global graph structure prior to the local connectivity. SPECTRE is also able to correct for imperfect generated spectra. **Top**: Conditioning spectral embedding. **Bottom**: The generated graph, plotted using its final spectral embedding.

One-shot generators are typically confronted with the demanding task of controlling global graph structure emergent from a large number of local node interactions. As a thought experiment, consider the toy problem of generating a tree by sampling node embeddings and then connecting them using some similarity kernel (Krawczuk et al., 2020; Serviansky et al., 2020; Vignac & Frossard, 2021). Even if the extended neighborhood of each node is locally tree-like, the overall graph will not be valid unless *all* nodes are positioned appropriately w.r.t. each other. Misplacing even a few nodes can

[*]Equal contribution [1]ETH Zurich [2]EPFL and Prescient Design, Genentech [3]Swiss Data Science Center. Correspondence to: Karolis Martinkus <martinkus@ethz.com>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

completely alter the global properties by introducing cycles and rendering the graph disconnected. Sadly, the aforementioned expressivity issue becomes increasingly pronounced as the size of the graph grows and can manifest in terms of non-convergent GAN training.

This work puts forth SPECTRE—an equivariant generator that aims to overcome the expressivity issues of one-shot approaches. SPECTRE decomposes the graph generation problem into two parts that are learned jointly: (*i*) modeling the dominant components of the graph spectrum and (*ii*) generating a graph conditioned on a set of eigenvalues and eigenvectors. Modeling the distribution of the top-k eigenvalues and eigenvectors is a simpler problem than one-shot graph generation. Crucially, a direct inspection of the graph spectrum conveys many pertinent global graph properties (e.g., connectivity, cluster structure, diameter) and can be utilized to construct embeddings that approximate the geodesic distance between nodes (Belkin & Niyogi, 2003; Coifman et al., 2005; Mohar, 1997). As shown in Figure 1, the (normalized) graph Laplacian eigenvectors (associated with low eigenvalues) capture well the coarse graph structure, making them ideal to model non-local dependencies. Thus, by generating the spectrum first, SPECTRE can control the global properties of the generated graphs. The learned eigenvectors and eigenvalues are then used to initialize the node embeddings of a second generator (inspired by GG-GAN (Krawczuk et al., 2020)) that acts as a local refinement procedure. Both steps are permutation equivariant, differentiable, and are optimized jointly in an end-to-end fashion.

Our experiments with synthetic and real-world graphs provide evidence that spectral conditioning helps to overcome the limitations of one-shot generators, managing to faithfully capture the graph statistics even for graphs with hundreds of nodes. Interestingly, SPECTRE can outperform state-of-the-art by a non-negligible margin, striking a compelling trade-off between the ability to generate graphs not in the training distribution (novelty) and modeling fidelity. We also find that conditioning SPECTRE on real spectra can yield further improvement without sacrificing novelty, especially when k is large, indicating that additional gains may be attainable by using a better spectrum generator.

2. Related Work

Graph generation entails building statistical models and fitting them to distributions of graphs. In this light, the problem is intimately linked with random graph models (Erdos et al., 1960; Holland et al., 1983; Eldridge et al., 2016) studied extensively in discrete mathematics, statistical physics, and network science. These models represent useful abstractions but are generally too simplistic to fit the real graph distributions that we care about. We refer the interested reader to the surveys (Chakrabarti & Faloutsos, 2006; Goldenberg et al., 2010) for a more in-depth exposition.

Deep learning approaches, on the other hand, forego the simplicity and tractability of most random graph models to achieve greater data fidelity. Two main types of deep learning approaches have been proposed:

Autoregressive. Autoregressive models learn to build a graph by iteratively adding new nodes and predicting their edges (You et al., 2018b; Liao et al., 2019; Dai et al., 2020). By breaking the problem into smaller manageable parts, these methods are generally apt at capturing complex statistics of the data distribution. Unfortunately, training autoregressive generators hinges on imposing an order on the nodes, which renders the generator permutation sensitive and can lead to low novelty (due to memorizing the training set and generating different permutations of the same adjacency matrix). In addition, autoregressive generators are generally non fully parallelizable and thus take longer to generate larger graphs.

One-shot. One-shot models aim to learn how to generate all edges between nodes at once, bringing the promise of parallelization. Within one-shot methods we can distinguish those that build on the variational autoencoder (VAE) and generative adversarial network (GAN) paradigms: Graph VAEs (Kipf & Welling, 2016; Simonovsky & Komodakis, 2018; Mitton et al., 2021) are generally easy to train and can work well for small graphs. A key challenge with VAEs that have a vector-based latent state is that to decouple the learning process from the permutation chosen when representing the graph as an adjacency matrix, one needs to solve a graph matching problem to align the VAE's input and output. The high computational complexity of graph matching necessitates the use of rough heuristics when training models with more than a handful of nodes and can significantly deteriorate performance.

The GAN framework provides an elegant alternative to VAEs (Wang et al., 2018; De Cao & Kipf, 2018; Bojchevski et al., 2018; Yang et al., 2019; Krawczuk et al., 2020). Adversarial training is believed to encourage larger sample diversity than maximizing likelihood, and moreover, by employing a permutation invariant discriminator one sidesteps the need for graph matching. Nevertheless, current equivariant one-shot GANs may exhibit convergence issues requiring involved fixes (Yang et al., 2019) and, even after recent improvements on their generator (Krawczuk et al., 2020; Vignac & Frossard, 2021), they remain less apt at modeling complex graph statistics of larger graphs. Our work builds and improves upon this previous work: we find that GAN generators have difficulty in faithfully capturing graph statistics—a phenomenon that manifests more prominently as graphs become larger. We introduce the idea of spectrum conditioning to mitigate this challenge.

Other generation strategies include learning a score function based on annealed Langevin dynamics (Niu et al., 2020) and graph normalizing flows (Liu et al., 2019). We also mention two special instances of GANs: GraphGAN models the distribution of a node's neighborhood conditioned on the rest of the graph (Wang et al., 2018), whereas NETGAN assembles a graph from a collection of random walks generated by a GAN (Bojchevski et al., 2018)—these models were designed for (and evaluated on) graph representation learning tasks and not graph generation. VAEs have also been applied on such tasks (Shi et al., 2020).

We note that the above discussion focuses on generally applicable methods and does not review specialized methods such as those specifically tailored to molecules (Jin et al., 2018; You et al., 2018a; Liu et al., 2018; Bresson & Laurent, 2019; Jin et al., 2020; Garcia Satorras et al., 2021; Mahmood et al., 2021; Liu et al., 2021).

Finally, there exist optimization-based approaches for building co-spectral graphs (Baldesi et al., 2018; Shine & Kempe, 2019). These works do not explicitly focus on modeling graph distributions.

3. Background

In this contribution, we consider unweighted undirected graphs $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is a set of n nodes connected (or not) by a set of edges $\mathcal{E}$. We index each node $v_i \in \mathcal{V}$ with $i = 1, \ldots, n$ and define the adjacency matrix $\boldsymbol{A}$ as $\boldsymbol{A}_{i,j} = 1$ if v_i and v_j are connected and $\boldsymbol{A}_{i,j} = 0$, otherwise.

3.1. Spectral Graph Theory

Spectral graph theory studies the connections between the spectrum of the graph Laplacian and the general properties of the graph. The normalized graph Laplacian[1] is defined as $\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{D}^{-\frac{1}{2}} \boldsymbol{A} \boldsymbol{D}^{-\frac{1}{2}}$, where $\boldsymbol{D} = \mathrm{diag}(d_1, \cdots, d_n)$ is the diagonal degree matrix defined as $d_i = \sum_{j=1}^{n} \boldsymbol{A}_{i,j}$. Being a symmetric positive semi-definite matrix, the graph Laplacian can always be diagonalized as $\boldsymbol{L} = \boldsymbol{U} \boldsymbol{\Lambda} \boldsymbol{U}^T$, where the orthogonal matrix $\boldsymbol{U} = [\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n]$ and the diagonal matrix $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \cdots, \lambda_n)$ contain the graph Laplacian eigenvectors and eigenvalues, respectively. We follow the convention of sorting eigenvalues in non-decreasing order as $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n \leq 2$.

Our approach is motivated by the well known fact that the first few eigenvalues $\boldsymbol{\lambda}_k = (\lambda_1, \cdots, \lambda_k)$ and eigenvectors $\boldsymbol{U}_k = [\boldsymbol{u}_1, \cdots, \boldsymbol{u}_k]$ of the graph Laplacian describe global properties of the graph structure such as its connectivity, clusterability, diameter, and node distance (see Appendix E).

[1]Though, in the following, by graph Laplacian we refer to the normalized Laplacian, our ideas can be trivially extended to the combinatorial and random-walk Laplacian matrices.

We assume the graph to be connected. Therefore the first eigenvector (associated with $\lambda_1 = 0$) only contains information about the node degree (a local feature) and we start conditioning graph generation with the *second* eigenvector.

3.2. Orthogonal Matrices

In the following, we describe some useful facts about the geometric, algebraic, and group structure of orthogonal matrices as eigenvectors of undirected graphs are such matrices.

The special orthogonal group $\mathbb{SO}(n)$. Being orthogonal matrices ($\boldsymbol{U}\boldsymbol{U}^\top = \boldsymbol{U}^\top \boldsymbol{U} = \boldsymbol{I}$), eigenvectors belong to the orthogonal matrix group $\mathbb{O}(n)$. The latter contains two connected components: one comprising of all of the matrices with a determinant of -1 and another comprising of all of the matrices with a determinant of $+1$ (rotation matrices), also known as the special orthogonal group $\mathbb{SO}(n)$. Graph eigenvectors are phase-independent, meaning that $-\boldsymbol{U}$ corresponds to the same eigenbasis as $\boldsymbol{U}$ and there always exists a $\mathbb{SO}(n)$ matrix that corresponds to a given graph's eigenbasis. We can thus generate all possible graph eigenspace while being restricted to $\mathbb{SO}(n)$, as we do in the following.

The Lie algebra of $\mathbb{SO}(n)$ is formed by skew-symmetric matrices ($\boldsymbol{S}^T = -\boldsymbol{S}$). The matrix exponential $\boldsymbol{U} = \exp(\boldsymbol{S})$ can then be used as a surjective map from skew-symmetric matrices onto $\mathbb{SO}(n)$ (Shepard et al., 2015). A simple counting argument reveals that orthogonal matrices can be defined using $n(n-1)/2$ parameters.

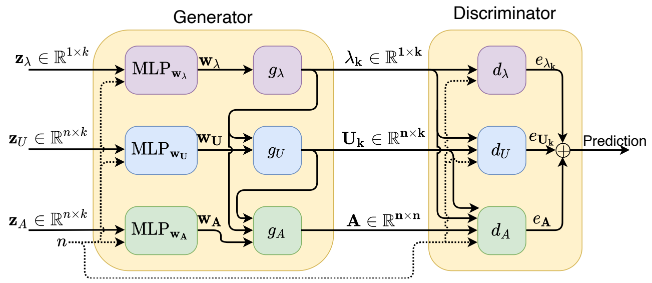
The Stiefel manifold $\mathbb{V}_k(n)$. In this work, we mostly focus on the first k eigenvectors $\boldsymbol{U}_k = [\boldsymbol{u}_1, \cdots, \boldsymbol{u}_k] \in \mathbb{R}^{n \times k}$, which form a Stiefel manifold $V_k(n)$ and abide to: $\boldsymbol{U}_k^\top \boldsymbol{U}_k = \boldsymbol{I}_k$, where $\boldsymbol{I}_k$ is the $k \times k$ identity matrix. Note that $\boldsymbol{U}_k \boldsymbol{U}_k^\top = \boldsymbol{I}_n$ iff $n = k$. One point in the Stiefel manifold can be transformed into any other with a rotation, i.e. , using multiplication with orthogonal matrices $\boldsymbol{R}_L \in \mathbb{SO}(n)$, $\boldsymbol{R}_R \in \mathbb{SO}(k)$:

$$\boldsymbol{U} \in \mathbb{V}_k(n) \Rightarrow \boldsymbol{R}_L \boldsymbol{U} \boldsymbol{R}_R \in \mathbb{V}_k(n). \quad (1)$$

We use this property to build the layers of the eigenvector generator (see Sec. 4.2). To generate a random initial point on the Stiefel manifold, one can create a random skew-symmetric matrix with $nk - k(k+1)/2$ non-zero parameters, compute its exponential and select the first k columns.

4. Spectrum and Graph Generation

SPECTRE aims to overcome a key difficulty that one-shot graph generators face: controlling the global graph structure by manipulating local interactions. To this end, SPECTRE generates graphs by first modeling the distribution of the top-k eigenvalues and eigenvectors $\boldsymbol{\lambda}_k \in \mathbb{A}_k$ and $\boldsymbol{U}_k \in \mathbb{V}_k(n)$,


Generator
Discriminator
$z_\lambda \in \mathbb{R}^{1\times k}$
$z_U \in \mathbb{R}^{n\times k}$
$z_A \in \mathbb{R}^{n\times k}$
n
MLP_{w_λ}
MLP_{w_U}
MLP_{w_A}
w_λ
w_U
w_A
g_λ
g_U
g_A
$\lambda_k \in \mathbb{R}^{1\times k}$
$U_k \in \mathbb{R}^{n\times k}$
$A \in \mathbb{R}^{n\times n}$
d_λ
d_U
d_A
e_{λ_k}
e_{U_k}
e_A
Prediction

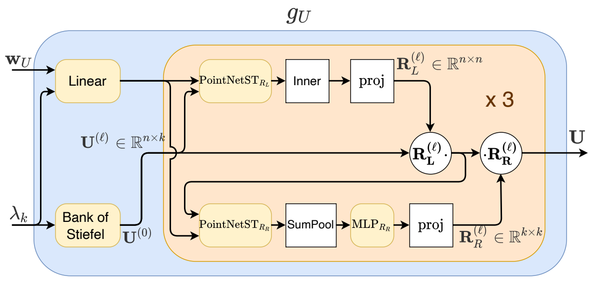

g_U
w_U
Linear
$U^{(\ell)} \in \mathbb{R}^{n\times k}$
λ_k
Bank of Stiefel
$U^{(0)}$
$\mathrm{PointNetST}_{R_L}$
Inner
proj
$R_L^{(\ell)} \in \mathbb{R}^{n\times n}$
x 3
$R_L^{(\ell)}\cdot$
$\cdot R_R^{(\ell)}$
U
$\mathrm{PointNetST}_{R_R}$
SumPool
MLP_{R_R}
proj
$R_R^{(\ell)} \in \mathbb{R}^{k\times k}$

Figure 2. **Left:** General architecture. Generation is performed sequentially with 3 GANs generating the eigenvalues (purple), the eigenvectors (blue) and finally the graph (green). Each generation step is conditioned on the previous generated variable. The input latent variable $\boldsymbol{w}$ of each sub-generator is obtained using an MLP. **Right:** Eigenvector generator. The initial eigenvectors $\boldsymbol{U}^{(0)}$ are selected from a learned Bank of Stiefel Manifolds. They are then transformed with 3 rotation layers (orange box) that each perform one left and one right rotation.

where $\mathbb{A}_k$ is the set of k strictly positive non-decreasing eigenvalues of the graph Laplacian

$$\mathbb{A}_k := \{\boldsymbol{\lambda}_k \in (0,2]^k \text{ s.t. } \lambda_1 \leq \cdots \leq \lambda_k\},$$

whereas $\mathbb{V}_k(n)$ is the Stiefel manifold.

The graph is then generated conditioned on $(\boldsymbol{U}_k, \boldsymbol{\lambda}_k)$. As explained in the background section, the dominant Laplacian spectra succinctly summarize the global structural properties of a graph and provide a rough embedding for the nodes. The latter can be used to bootstrap the graph generator module, simplifying its job.

We cover the architecture of SPECTRE (Figure 2, left) by first presenting the conditional graph generator and then showing how the top-k eigenvalue and eigenvector generation works. Each latent variable $\boldsymbol{z}_\lambda \in \mathbb{R}^{1\times k}$, $\boldsymbol{z}_U \in \mathbb{R}^{n\times k}$, $\boldsymbol{z}_A \in \mathbb{R}^{n\times k}$ is obtained by sampling a uniform point $\boldsymbol{z}_U$ from a hypersphere. Then, they are transformed using four-layer Multi-Layer Perceptrons ($\mathrm{MLP}_{\boldsymbol{w}_\lambda}$, $\mathrm{MLP}_{\boldsymbol{w}_U}$, $\mathrm{MLP}_{\boldsymbol{w}_A}$) to obtain $\boldsymbol{w}_\lambda \in \mathbb{R}^{1\times k}$, $\boldsymbol{w}_U \in \mathbb{R}^{n\times k}$, $\boldsymbol{w}_A \in \mathbb{R}^{n\times k}$. Further implementation details of the architecture can be found in Appendix A.

4.1. Conditional Graph Generation

The SPECTRE graph generator g_L aims to construct graphs with a given dominant spectra $(\boldsymbol{\lambda}_k, \boldsymbol{U}_k)$:

$$\boldsymbol{A} = g_A(\boldsymbol{\lambda}_k, \boldsymbol{U}_k, \boldsymbol{w}_A).$$

First, we build $\boldsymbol{L}^{(0)} \in \mathbb{R}^{n\times n}$, a rough approximation of the Laplacian matrix

$$\boldsymbol{L}^{(0)} = \boldsymbol{U}_k \operatorname{diag}(\boldsymbol{\lambda}_k)\, \boldsymbol{U}_k^\top \ .$$

Though $\boldsymbol{L}^{(0)}$ does not look like a graph at this stage as it lacks the local connectivity information, it internally encodes the global graph structure. This matrix is passed in place of an adjacency matrix to a l-layer Provably Powerful Graph Network (PPGN) (Maron et al., 2019) using $\boldsymbol{\lambda}_k$, $\boldsymbol{U}_k$ and $\boldsymbol{w}_A$ as node features. The PPGN then produces the final adjacency matrix. We can interpret this as g_A taking the initial approximation of the Laplacian matrix $\boldsymbol{L}^{(0)}$ and progressively refining it:

$$\boldsymbol{L}^{(l)} = \mathrm{PPGN}_l(\boldsymbol{L}^{(l-1)}) \quad \text{for layer} \quad l = 1, \cdots, L-1.$$

The final layer then produces the adjacency matrix directly to avoid a complicated manual conversion from a Laplacian to an adjacency:

$$\boldsymbol{A} = \sigma(\mathrm{PPGN}_L(\boldsymbol{L}^{(L-1)})),$$

where σ is the sigmoid activation function. Since we actually use high-dimensional representations between the PPGN layers, in this interpretation, this high-dimensional representation encodes $\boldsymbol{L}^{(l)}$. PPGN's expressive power matches that of a 3-WL test. The network is thus better suited to modeling and distinguishing graphs than typical Graph Neural Networks (GNNs) that typically are as discriminative as a 2-WL test (Xu et al., 2019; Morris et al., 2019). More recent GNNs are either less expressive than PPGN (Zhang & Li, 2021; Sandfelder et al., 2021; Papp et al., 2021; Bevilacqua et al., 2021), or are computationally slower in the non-asymptotic regime (Vignac et al., 2020; Morris et al., 2020), or hinge on the computation of predefined features (Bouritsas et al., 2020). We stress that the list of mentioned references provides an incomplete sample of all known GNN architectures whose expressive power surpasses the 2-WL test.

We should note that the set $\mathbb{S}_k(n) \subset \mathbb{A}_k \times \mathbb{V}_k(n)$ of *valid graph Laplacian spectra*, i.e., those $(\boldsymbol{\lambda}_k, \boldsymbol{U}_k)$ that lead to a valid graph G, can be much smaller than those that are modeled by SPECTRE. Unfortunately, the problem of determining whether $(\boldsymbol{\lambda}_k, \boldsymbol{U}_k) \in \mathbb{S}_k(n)$ appears to be computationally hard when $k \ll n$, meaning that we cannot expect that the spectrum generator always returns a valid sample. We thus do not enforce exact conditioning on $(\boldsymbol{\lambda}_k, \boldsymbol{U}_k)$ but instead motivate the graph generator to generate graphs whose first k eigenvalues and eigenvectors are close to those sampled during the first stages.

The graph discriminator takes the adjacency matrix and corresponding spectral features as input:

$$e_{\boldsymbol{A}} = d_A(\boldsymbol{A}, \boldsymbol{\lambda}_k, \boldsymbol{U}_k, n).$$

Due to its conditioning, the discriminator helps to ensure that the generated graph and eigenvectors are consistent. To further encourage the discriminator to focus on the relation between the graph and the eigenvectors, we sometimes pass true (perturbed) eigenvalues and eigenvectors to the generator. The discriminator architecture is analogous to the generator, with an additional global pooling layer before the output. Note that having spectral features strengthens the discriminator. Consider, for instance, the task of determining whether a graph is connected—a task that a good discriminator needs to solve. Whereas in normal architectures the GNN depth needs to exceed the graph diameter to determine connectivity, when spectral features are available, connectivity can be checked locally. To guarantee global consistency, we then just need to ensure that the spectral features we condition on are well-posed.

Note that while in this paper we mainly focus on the generation of graph structure alone, we can also use the PPGN to directly produce node and edge features for the generated graph (see Appendix A.1).

4.2. Conditional Eigenvector Generation

The eigenvector generator (Figure 2 right), aims to construct eigenvectors $\boldsymbol{U}_k$ matching given eigenvalues $\boldsymbol{\lambda}_k$:

$$\boldsymbol{U}_k = g_U(\boldsymbol{\lambda}_k, \boldsymbol{w}_U),$$

It operates by iteratively refining a starting eigenvector matrix. First, a starting Stiefel matrix $\boldsymbol{U}_k^{(0)}$ is selected from a bank of learned matrices $\{\boldsymbol{B}_0, ..., \boldsymbol{B}_m\}$ with $\boldsymbol{B}_i \in \mathbb{V}_k(n)$ for all $i = 1, \cdots, m$. Using a bank helps to make the generation easier, as orthogonal matrices which correspond to valid graph Laplacian eigenvectors form a potentially small subset of all orthogonal matrices. However, it is possible to instead use only one fixed learnable rotation matrix as input, which would cause a slight decrease in generation quality. The selection is done by generating a query matrix $\boldsymbol{Q} = \text{MLP}(\boldsymbol{\lambda}_k)$ of size $n \times k$ that is compared to matrices in the bank using the canonical Stiefel manifold metric (Edelman et al., 1998):

$$m(\boldsymbol{Q}, \boldsymbol{B}_i) := \text{tr}\big(\boldsymbol{Q}^T\big(\boldsymbol{I} - \frac{1}{2}\boldsymbol{B}_i\boldsymbol{B}_i^\top\big)\boldsymbol{B}_i\big),$$

which we normalize such that the distance from $\boldsymbol{B}_i$ to itself is equal to one. Here tr(X) computes the trace of the matrix. The starting matrix $\boldsymbol{U}_k^{(0)}$ is then sampled using Gumbel-softmax (Jang et al., 2016; Maddison et al., 2017).

The generator proceeds to refine $\boldsymbol{U}_k^{(0)}$ by repeatedly multiplying it with left and right orthogonal matrices:

$$\boldsymbol{U}_k^{(\ell)} = \boldsymbol{R}_L^{(\ell)}\, \boldsymbol{U}_k^{(\ell-1)} \boldsymbol{R}_R^{(\ell)} \quad \text{for layer} \quad \ell = 1, \cdots, L.$$

As described in (1) this transformation ensures that the matrix stays on the Stiefel manifold, meaning that it is a valid eigenvector matrix with orthogonal unit-length column vectors. The left refinement matrix is constructed by processing inputs with a PointNetST layer (Segol & Lipman, 2019) and projecting the result onto $\mathbb{SO}(n)$ by constructing a skew-symmetric matrix and finally using the matrix exponential:

$$\boldsymbol{R}_L^{(\ell)} = \text{proj}\big(\text{outer}\big(\text{PointNetST}(\boldsymbol{U}_k^{(\ell-1)}, \boldsymbol{\lambda}_k, \boldsymbol{z}_U)\big)\big),$$

where we define outer$(\boldsymbol{X}) := \boldsymbol{X}\boldsymbol{X}^\top$ and proj$(\boldsymbol{X}) :=$ $\exp(\text{tril}(\boldsymbol{X}) - \text{tril}(\boldsymbol{X})^\top)$, where tril$(X)$ gives the lower triangular of the matrix. The right rotation matrix is constructed similarly by using a second PointNetST layer (no parameter-sharing) and mean pooling over the set of nodes:

$$\boldsymbol{R}_R^{(\ell)} = \text{proj}\big(\text{MLP}_{\boldsymbol{R}_R}\big(\frac{1}{n}\sum_{i=1}^{n} \text{PointNetST}(\boldsymbol{U}_k^{(\ell-1)}, \boldsymbol{\lambda}_k, \boldsymbol{z}_U)\big)\big).$$

The eigenvector discriminator takes the eigenvectors and the corresponding eigenvalues as input:

$$e_{\boldsymbol{U}_k} = d_U(\boldsymbol{U}_k, \boldsymbol{\lambda}_k, n).$$

Since spectral node embeddings induce a clustering, for the discriminator we use an architecture based on PointNet (Qi et al., 2017), which achieves good results on point cloud segmentation and classification. This architecture comprises of a right rotation, a point-wise transformation with an MLP, another right rotation, and a PointNetST layer followed by mean pooling and an MLP. The right rotations are constructed in the same way as done in the generator.

We use PointNetST layers as they are efficient and can approximate any equivariant set function (Segol & Lipman, 2019). Model weights are initialized such, that all of the rotation matrices are initially close to identity.

To avoid any sign ambiguity in the eigenvector representation we transform both the generated and the true eigenvectors such that the maximum absolute value for each eigenvector is positive.

4.3. Eigenvalue Generation

The final piece of the puzzle entails generating

$$\boldsymbol{\lambda}_k = g_\lambda(\boldsymbol{w}_\lambda).$$

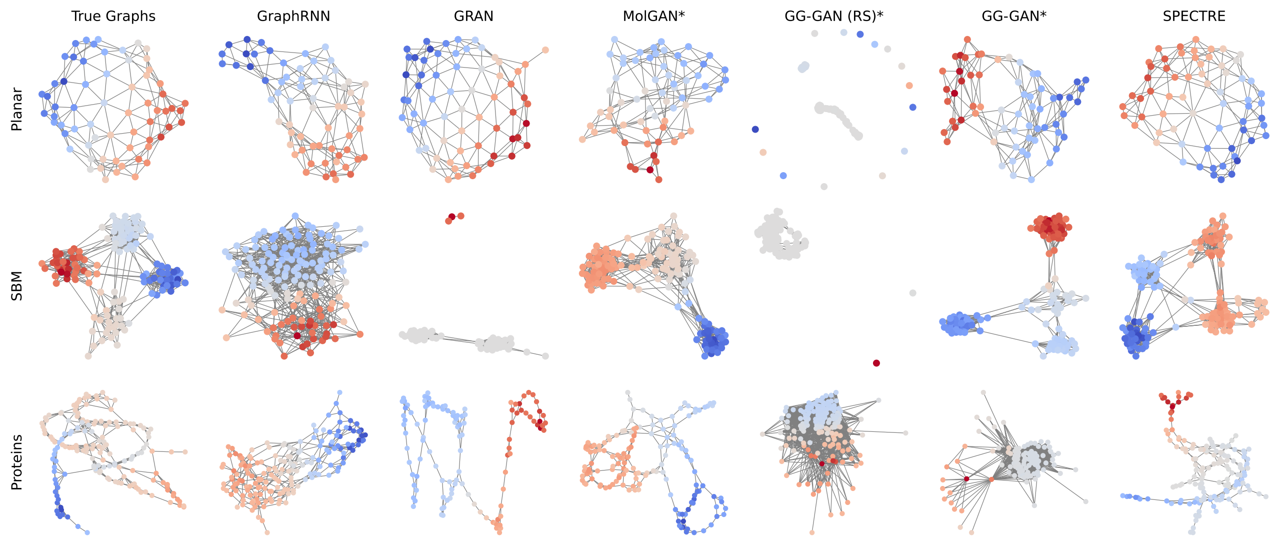


Figure 3. A set of sample graphs produced by the models. Each row is conditioned on the same number of nodes.

As eigenvalues are just an increasing sequence, the eigenvalue generator is a simple 4-layer 1D CNN with upsampling (Donahue et al., 2018).

Likewise, the discriminator:

$$e_{\boldsymbol{\lambda}_k} = d_\lambda(\boldsymbol{\lambda}_k, n),$$

is a strided 4-layer 1D CNN with a linear final read-out layer. Both networks employ gated activation units $z = \tanh(W_1 X) \cdot \sigma(W_2 X)$ as used in WaveNet (Oord et al., 2016a) and PixelCNN (Oord et al., 2016b).

4.4. Training

To train our model we use the WGAN-LP loss ($\lambda_{\text{LP}} = 5$) (Petzka et al., 2018), ADAM optimizer ($\beta_1 = 0.5$, $\beta_2 = 0.9$) (Kingma & Ba, 2015), and learning rate of $1e-4$ for both the generator and the discriminator.

During training we utilize a form of teacher forcing, where initially each model is trained separately for 26k training steps, using true slightly perturbed spectral features from the training set for conditioning. We then gradually anneal the mixing temperature τ, which defines how many real inputs each model receives, over the next 26k steps from 1.0 to 0.8 using a cosine schedule. This teacher forcing serves to "teach" the model how to construct graphs with given spectral features and correct small errors. All of our models are trained for 150k steps in total. The code is publicly available[2].

The selection of k (the number of eigenvalues and eigenvectors) is done as follows. First we train only the graph generator g_A conditioned on true spectra using different values of $k \in [2, 4, 8, 16, 32]$ for 26k steps. Then we select the lowest k which resulted in the generation of good quality graph samples (low MMD measures).

[2]https://github.com/KarolisMart/SPECTRE

5. Experimental Evaluation

Our experimental setup largely follows You et al. (2018b) and Liao et al. (2019) with some important extensions that are discussed below.

Datasets. We consider five real and synthetic datasets of varying size and connectivity: Community-small (12-20 nodes) (You et al., 2018b), QM9 (9 nodes) (Ramakrishnan et al., 2014), Planar graphs (64 nodes), a Stochastic Block Model (20-40 nodes per community, 2-5 communities), Proteins (100-500 nodes) (Dobson & Doig, 2003). All datasets are described in Appendix C. We split all datasets into test (20%) and training (80%) sets. We use 20% of the training set for validation. We generate the same number of samples as there are in the test split of each dataset.

MMD measures. Generated graph quality is commonly evaluated by comparing the distributions of graph statistics between the generated and test graphs (Li et al., 2018; You et al., 2018b; Liao et al., 2019; Krawczuk et al., 2020). In particular, we adopt the Maximum Mean Discrepancy (MMD) measures used by Liao et al. (2019): node degree (Deg.), clustering coefficient (Clus.), orbit count (Orbit), eigenvalues of the normalized graph Laplacian (Spec.). We further introduce an eigenspace-based MMD (Wavelet) that evaluates the similarity of graph eigenspaces using statistics from a graph wavelet transform (Hammond et al., 2011).

	Planar graphs										
Model	Deg. ↓	Clus. ↓	Orbit ↓	Spec. ↓	Wavelet ↓	Ratio ↓	Valid ↑	Unique ↑	Novel ↑	Val., Uniq. & Nov. ↑	t (s) ↓
Training set	0.0002	0.0310	0.0005	0.0052	0.0012	1.0	100.0	100.0	—	—	—
GraphRNN	0.0049	0.2779	1.2543	0.0459	0.1034	527.4	0.0	100.0	100.0	0.0	0.774
GRAN	0.0007	0.0426	0.0009	0.0075	0.0019	1.9	97.5	85.0	2.5	0.0	0.920
MolGAN*	0.0009	0.3164	1.1730	0.1989	0.0729	491.9	0.0	25.0	100.0	0.0	0.002
GG-GAN (RS)*	0.1005	0.2571	1.0313	0.2040	0.3829	586.3	0.0	100.0	100.0	0.0	0.011
GG-GAN*	0.0630	1.1820	1.2280	0.1990	0.1890	601.0	0.0	10.0	100.0	0.0	0.011
SPECTRE ($k = 2$)	0.0005	0.0785	0.0012	0.0112	0.0059	2.9	25.0	100.0	100.0	25.0	0.026
SPECTRE ($k = 2$, real spectra)	0.0010	0.0668	0.0010	0.0095	0.0056	3.1	47.5	100.0[‡]	100.0[‡]	47.5[‡]	0.011

	Stochastic Block Model										
Model	Deg. ↓	Clus. ↓	Orbit ↓	Spec. ↓	Wavelet ↓	Ratio ↓	Valid ↑	Unique ↑	Novel ↑	Val., Uniq. & Nov. ↑	t (s) ↓
Training set	0.0008	0.0332	0.0255	0.0063	0.0007	1.0	100.0	100.0	—	—	—
GraphRNN	0.0055	0.0584	0.0785	0.0065	0.0431	14.9	5.0	100.0	100.0	5.0	5.108
GRAN	0.0113	0.0553	0.0540	0.0054	0.0212	9.8	25.0	100.0	100.0	25.0	1.887
MolGAN*	0.0235	0.1161	0.0712	0.0117	0.0292	15.8	10.0	95.0	100.0	9.5	0.002
GG-GAN (RS)*	0.0338	0.0581	0.1019	0.0613	0.1749	61.5	0.0	100.0	100.0	0.0	0.056
GG-GAN*	0.0035	0.0699	0.0587	0.0094	0.0202	7.8	25.0	100.0	100.0	25.0	0.057
SPECTRE ($k = 4$)	0.0015	0.0521	0.0412	0.0056	0.0028	2.0	52.5	100.0	100.0	52.5	0.074
SPECTRE ($k = 4$, real spectra)	0.0079	0.0528	0.0643	0.0074	0.0112	6.2	60.0	100.0[‡]	100.0[‡]	60.0[‡]	0.057

Table 1. Sample quality on synthetic datasets. We also provide SPECTRE results when the graph generator is conditioned on real spectra $(\boldsymbol{\lambda}_k, \boldsymbol{U}_k)$ from the test set. [‡] novelty is compared to the test set graphs, from which $(\boldsymbol{\lambda}_k, \boldsymbol{U}_k)$ were taken. Methods marked with * are similar models implemented with building blocks from our architecture.

Similar features have been used to identify the role of each node on the graph (Donnat et al., 2018) — see Appendix B for definition. The distance between the samples is computed using the total variational Gaussian kernel, which is consistent with the Gaussian earth mover's distance kernel while being considerably faster (Liao et al., 2019). To stay consistent with published results for the Community dataset for it we use a Gaussian EMD kernel (Liu et al., 2019; Niu et al., 2020). We also summarize these measures by reporting the average ratio (Ratio) between a generator's and the training set's MMD values. As the training set MMD values are the best we can hope to achieve, this ratio indicates how far away we are from the optimal statistics.

Validity. While the measures based on graph statistics measure how similar certain properties are between two distributions of graphs, they cannot guarantee that the generated graphs come from the same distribution. To this end, when the true distribution is known we also report a validity measure counting the percentage of generated graphs that belong to the ground-truth class. Specifically, we assert that planar graphs must be planar and connected and that SBM graphs are statistically indistinguishable from those generated by the ground-truth model (see Appendix C).

Novelty and uniqueness. While generating correct and statistically similar graphs is important, ultimately, a generator is useless unless the generated graphs are sufficiently diverse. To ensure that our model generates sufficiently diverse graphs, we follow Krawczuk et al. (2020) and introduce two measures based on graph isomorphism classes (Cordella et al., 2001): *Uniqueness.* We count the fraction of the generated graphs belonging to a unique isomorphism class. This way we check if the generator does not collapse to generating different permutations of the same few graphs. *Novelty.* We further determine what fraction of the generated graphs belong to an isomorphism class unseen in the training set. This measure further estimates how many novel graphs the generator is able to produce.

The validity, uniqueness, and novelty measures are summarized by computing the percentage of graphs that are all valid (if applicable), unique and novel. Together with the MMD ratio, this constitutes the two most important evaluation metrics we use.

Baselines. Besides the autoregressive GRAN and GraphRNN models, we construct GAN baselines inspired by MolGAN (De Cao & Kipf, 2018) and GG-GAN (Krawczuk et al., 2020). Neither of these baselines uses spectral conditioning and serve to prove the value of our approach. We choose to modify our architecture to create the baselines instead of faithfully re-implementing them, to ensure that improvements in score seen by SPECTRE do not come solely from other improvements we have made to the architecture, which include using a more powerful GNN, improved noise pre-processing and a symmetric architecture which is more stable to train. We create MolGAN*, by replacing our conditional graph generator with a three-layer MLP and a set of linear layers that convert

	Proteins									
Model	Deg. ↓	Clus. ↓	Orbit ↓	Spectral ↓	Wavelet ↓	Ratio ↓	Unique ↑	Novel ↑	Uniq. & Nov. ↑	t (s) ↓
Training set	0.0003	0.0068	0.0032	0.0009	0.0003	1.0	100.0	—	—	—
GraphRNN	0.0040	0.1475	0.5851	0.0152	0.0530	82.3	100.0	100.0	100.0	36.41
GRAN	0.0479	0.1234	0.3458	0.0125	0.0341	82.7	100.0	100.0	100.0	11.68
MolGAN*	0.0008	0.0644	0.0081	0.0021	0.0012	4.2	97.3	100.0	97.3	0.003
GG-GAN (RS)*	0.4727	0.1772	0.7326	0.4102	0.6278	875.8	100.0	100.0	100.0	0.482
GG-GAN*	0.5192	0.5220	0.7326	0.3996	0.6157	906.5	100.0	100.0	100.0	0.485
SPECTRE ($k = 16$)	0.0056	0.0843	0.0267	0.0052	0.0118	16.9	100.0	100.0	100.0	0.507
SPECTRE ($k = 16$, real spectra)	0.0013	0.0469	0.0287	0.0020	0.0022	6.0	100.0	$100.0^{\ddagger}$	$100.0^{\ddagger}$	0.485

Table 2. Protein graphs with up to 500 nodes. We also provide SPECTRE results when the graph generator is conditioned on real spectra $(\boldsymbol{\lambda}_k, \boldsymbol{U}_k)$ from the test set. ‡ novelty is compared to the test set graphs, from which $(\boldsymbol{\lambda}_k, \boldsymbol{U}_k)$ were taken. Methods marked with * are similar models implemented with building blocks from our architecture.

final embedding to an adjacency matrix, node, and edge features (if needed). We retain the noise pre-processing MLP. We also build two models which use our PPGN-based generator. GG-GAN* uses learned fixed node embeddings alongside processed noise $\boldsymbol{w}_A$ as generator input, while GG-GAN (RS)* depends only on the random pre-processed set $\boldsymbol{w}_A$. In all cases, the discriminator architecture remains unchanged.

To compare how expensive it is to generate graphs with SPECTRE and with these baselines, we also provide the time it takes to generate one mini-batch of 10 graphs.

Model selection. GANs tend to oscillate around the optima. To this end, we track the exponential moving average of model weights with a retention coefficient of 0.995. We also compute the MMD measures between training and validation sets and select the model with the best average ratio between its and the training set's MMD values.

5.1. Results

Synthetic datasets. Table 1 reports our results for distributions consisting of planar and SBM graphs and shows how SPECTRE, respectively, achieves a 169.6× and 3.9× improvement over the best non-overfitting baseline. Examples of generated graphs can be found in Figure 3 and Appendix F. We see that only SPECTRE is able to produce novel and valid planar graphs. This example also highlights the importance of measuring generated graph uniqueness, as GRAN is able to produce perfectly valid graphs, but those graphs are memorized from the training set. When looking at SBM graphs, we observe that SPECTRE achieves much better MMD scores than alternatives. It is likely that the diverse number of nodes and communities causes GRAN to underperform here, while the constant number of nodes in planar graphs helps it overfit. Notice also how GAN baselines that do not utilize spectral conditioning do not produce good results on these larger graphs, whereas they work much better when the number of nodes is reduced (see e.g., the Community results in Table 4). The latter confirms our supposition that using spectral conditioning to separate global and local scale graph structure generation helps to overcome the expressivity limits of one-shot GAN generators.

Proteins. As seen in Table 2, the findings made on synthetic datasets also hold to larger real-world protein graphs. Here, the expressivity issues are even more pronounced, with some GAN baselines mostly failing to train. A notable exception is MolGAN*, which, in part due to our improved discriminator, is able to produce unique graphs with great statistical measures. Nevertheless, upon closer inspection, we observe that the graphs generated by MolGAN* have the tendency to be minor variations of a small number of graphs. In fact, on average only 17.6% of edges differ between any two graphs produced by MolGAN* (Appendix G). Using teacher forcing and conditioning the generator and the discriminator on true spectral features in SPECTRE helps to avoid mode-collapse, which is a common issue with GANs, and helps to guide the model to convergence. Figure 3 and Appendix F display generated graphs.

Conditioning on real spectra. In Tables 1 and 2, we additionally provide the results achieved by the same SPECTRE model, when conditioned on *true* first k eigenvalues and eigenvectors taken form test graphs. We observe that when SPECTRE fails to capture the correct graph distribution (e.g. high MMDs in Table 2), conditioning on real spectra significantly improves the generation process. This behavior suggests that there is still room for improvement in our spectrum generation procedure, especially when generating a large number of eigenvectors and eigenvalues. Furthermore, it is a sign that the conditional graph generator is able to leverage accurate spectral features without overfitting. In contrast, in the case when SPECTRE achieves a very good fit of the training distribution (Table 1), the percentage of

Dataset	Eigenvalue	Wavelet (true)	Wavelet (fake)
Planar (k=2)	17.60	45.43	206.15
SBM (k=4)	9.39	7.01	19.46
Proteins (k=16)	36.83	4.94	23.42

Table 3. MMD ratios (vs training set MMD) for generated k eigenvalues (direct MMD) and generated eigenvectors (Wavelet MMD) conditioned on fake and true test eigenvalues.

	Community-small			
Model	Deg. ↓	Clus. ↓	Orbit ↓	Ratio ↓
Training set	0.02	0.07	0.01	1.0
GraphRNN	0.08	0.12	0.04	3.2
GRAN	0.06	0.11	0.01	1.9
GraphVAE	0.35	0.98	0.54	28.5
DeepGMG	0.22	0.95	0.40	21.5
GNF	0.02	0.20	0.17	6.9
EDP-GNN	0.05	0.14	0.03	2.5
MolGAN*	0.06	0.13	0.01	1.9
GG-GAN (RS)*	0.04	0.53	0.03	4.9
GG-GAN*	0.08	0.22	0.08	5.5
SPECTRE ($k=2$)	0.02	0.21	0.01	1.7

Table 4. Community graphs with up to 20 nodes. Methods marked with * are similar models implemented with building blocks from our architecture. Other baseline results are taken from (Liu et al., 2019; Niu et al., 2020)

valid graphs (i.e., graphs that are planar or match the true SBM parameters) improves when using true spectra while some MMD measures, most notably the degree MMD, become worse. This signals slight overfitting on the generated spectra by the graph generator g_A and some divergence between the generated and true spectra. Note that training g_A only on the true spectra would likely improve results.

In Table 3, we test the quality of the k generated eigenvalues and eigenvectors using respectively a direct MMD for the eigenvalues and the Wavelet MMD (See B) for the eigenvectors. We evaluate the eigenvector generator both on the true and on the generated set of eigenvalues. As the overall MMD ratios are high, it indicates that a) the spectral generation procedure can be improved, and that b) conditioning even on imperfect spectra can significantly improve one-shot graph generation.

Smaller datasets. We also compared our method to a larger number of baselines that are unable to successfully generate larger graphs on two standard datasets consisting of small community graphs and molecules. As shown in Table 4, SPECTRE achieves the smallest average MMD ratio. While the MLP generator (MolGAN*) performs well on this simple graph distribution, we see that spectral conditioning considerably improves performance as compared to other equivariant GANs.

	QM9		
Model	Valid ↑	Val. & Uniq. ↑	Val., Uniq. & Nov. ↑
CharacterVAE	10.3	7.0	6.3
GrammarVAE	60.2	5.6	4.5
GraphVAE	55.7	42.0	26.1
GraphVAE NoGM	81.0	20.5	11.9
GraphTransformerVAE	74.6	16.8	15.8
MolGAN	98.1	10.2	9.6
MolGAN*	99.0	0.6	0.6
GG-GAN (RS)*	51.2	24.4	24.4
GG-GAN* ($\lvert\text{emb}\rvert = 64$)	100.0	0.4	0.4
GG-GAN* ($\lvert\text{emb}\rvert = 2$)	6.6	1.8	1.8
SPECTRE ($k=2$)	87.3	31.2	29.1

Table 5. QM9 molecules with up to 9 heavy atoms. Methods marked with * are similar models implemented with building blocks from our architecture. Other baseline results are taken from (Mitton et al., 2021; De Cao & Kipf, 2018).[4]

Table 5 presents our results on the QM9 molecule dataset, which also involves node and edge feature generation. As highlighted by De Cao & Kipf (2018), GANs are prone to mode collapse on this dataset. We found that our strong PPGN discriminator aggravates this problem, especially for the GG-GAN* and MolGAN* variants.[5] On the other hand, the GG-GAN (RS)* and SPECTRE methods proved more robust, which again highlights the power of conditional spectral generation in preventing mode collapse.

6. Conclusion

We argue that spectral conditioning is a key step in overcoming the challenges of one-shot graph generation: it prevents mode collapse, thus favoring the generation of novel and unique graphs, and it aids the generator to control the global structure leading to higher quality samples. Our experiments show that SPECTRE outperforms competing methods with respect to generation time and sample quality, while also tending to generate more unique, valid, and novel graphs.

Disclosure of Funding

Andreas Loukas would like to thank the Swiss National Science Foundation for supporting him in the context of the project “Deep Learning for Graph Structured Data”, grant number PZ00P2 179981.

[4]Note, that we used the evaluation code provided by (Vignac & Frossard, 2021; De Cao & Kipf, 2018), which accepts samples with multiple connected components. If we instead only keep the largest connected component SPECTRE Val., Uniq. & Nov. score drops to 19.1.

[5]We attempted to prevent mode collapse for our GG-GAN* and MolGAN* by increasing dropout rates, increasing the perturbations used for gradient penalty, and, removing the noise preprocessing MLP. None of these helped, besides reducing the number of dimensions used by the learned embeddings for GG-GAN*.

References

Alon, N. Eigenvalues and expanders. *Combinatorica*, 6(2): 83–96, 1986.

Alon, N. and Milman, V. D. $\lambda 1$, isoperimetric inequalities for graphs, and superconcentrators. *Journal of Combinatorial Theory, Series B*, 38(1):73–88, 1985.

Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pp. 214–223, 2017.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *Advances in NIPS 2016 Deep Learning Symposium*, 2016.

Baldesi, L., Butts, C. T., and Markopoulou, A. Spectral graph forge: Graph generation targeting modularity. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pp. 1727–1735. IEEE, 2018.

Belkin, M. and Niyogi, P. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

Bevilacqua, B., Frasca, F., Lim, D., Srinivasan, B., Cai, C., Balamurugan, G., Bronstein, M. M., and Maron, H. Equivariant subgraph aggregation networks. *arXiv preprint arXiv:2110.02910*, 2021.

Bojchevski, A., Shchur, O., Zügner, D., and Günnemann, S. Netgan: Generating graphs via random walks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pp. 609–618, 2018.

Bouritsas, G., Frasca, F., Zafeiriou, S., and Bronstein, M. M. Improving graph neural network expressivity via subgraph isomorphism counting. *arXiv preprint arXiv:2006.09252*, 2020.

Bresson, X. and Laurent, T. A two-step graph convolutional decoder for molecule generation. In *NeurIPS Workshop on Machine Learning and the Physical Sciences*, 2019.

Brock, A., Donahue, J., and Simonyan, K. Large scale gan training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2018.

Chakrabarti, D. and Faloutsos, C. Graph mining: Laws, generators, and algorithms. *ACM computing surveys (CSUR)*, 38(1):2–es, 2006.

Chung, F. R. and Graham, F. C. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997.

Coifman, R. R., Lafon, S., Lee, A. B., Maggioni, M., Nadler, B., Warner, F., and Zucker, S. W. Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. *Proceedings of the national academy of sciences*, 102(21):7426–7431, 2005.

Cordella, L. P., Foggia, P., Sansone, C., and Vento, M. An improved algorithm for matching large graphs. In *3rd IAPR-TC15 workshop on graph-based representations in pattern recognition*, pp. 149–159. Citeseer, 2001.

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

De Cao, N. and Kipf, T. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

Defferrard, M., Martin, L., Pena, R., and Perraudin, N. Pygsp: Graph signal processing in python.

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

Donahue, C., McAuley, J., and Puckette, M. Adversarial audio synthesis. In *International Conference on Learning Representations*, 2018.

Donnat, C., Zitnik, M., Hallac, D., and Leskovec, J. Learning structural node embeddings via diffusion wavelets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1320–1329, 2018.

Edelman, A., Arias, T. A., and Smith, S. T. The geometry of algorithms with orthogonality constraints. *SIAM journal on Matrix Analysis and Applications*, 20(2):303–353, 1998.

Eldridge, J., Belkin, M., and Wang, Y. Graphons, mergeons, and so on! In *Advances in Neural Information Processing Systems*, pp. 2307–2315, 2016.

Erdos, P., Rényi, A., et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

Fahrmeir, L., Kneib, T., Lang, S., and Marx, B. *Regression*. Springer, 2007.

Garcia Satorras, V., Hoogeboom, E., Fuchs, F., Posner, I., and Welling, M. E (n) equivariant normalizing flows. *Advances in Neural Information Processing Systems*, 34, 2021.

Goldenberg, A., Zheng, A. X., Fienberg, S. E., and Airoldi, E. M. A survey of statistical network models. 2010.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. Improved training of wasserstein gans, 2017.

Hammond, D. K., Vandergheynst, P., and Gribonval, R. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

Hammond, D. K., Gur, Y., and Johnson, C. R. Graph diffusion distance: A difference measure for weighted graphs based on the graph laplacian exponential kernel. In *2013 IEEE Global Conference on Signal and Information Processing*, pp. 419–422. IEEE, 2013.

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

Holland, P. W., Laskey, K. B., and Leinhardt, S. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

Huang, P.-S., Boyken, S. E., and Baker, D. The coming of age of de novo protein design. *Nature*, 537(7620): 320–327, 2016.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. 2016.

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

Jin, W., Barzilay, D., and Jaakkola, T. Hierarchical generation of molecular graphs using structural motifs. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 4839–4848. PMLR, 13–18 Jul 2020.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Kipf, T. N. and Welling, M. Variational graph auto-encoders. In *NIPS Workshop onBayesian Deep Learning*, 2016.

Krawczuk, I., Abranches, P., Loukas, A., and Cevher, V. Gg-gan: A geometric graph generative adversarial network. 2020.

Lee, J. R., Gharan, S. O., and Trevisan, L. Multiway spectral partitioning and higher-order cheeger inequalities. *Journal of the ACM (JACM)*, 61(6):1–30, 2014.

Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. Learning deep generative models of graphs. 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*, pp. 4255–4265, 2019.

Liu, J., Kumar, A., Ba, J., Kiros, J., and Swersky, K. Graph normalizing flows. *Advances in Neural Information Processing Systems*, 32:13578–13588, 2019.

Liu, M., Yan, K., Oztekin, B., and Ji, S. GraphEBM: Molecular graph generation with energy-based models. In *Energy Based Models Workshop - ICLR 2021*, 2021.

Liu, Q., Allamanis, M., Brockschmidt, M., and Gaunt, A. L. Constrained graph variational autoencoders for molecule design. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 7806–7815, 2018.

Maddison, C., Mnih, A., and Teh, Y. The concrete distribution: A continuous relaxation of discrete random variables. In *Proceedings of the international conference on learning Representations*. International Conference on Learning Representations, 2017.

Mahmood, O., Mansimov, E., Bonneau, R., and Cho, K. Masked graph modeling for molecule generation. *Nature communications*, 12(1):1–12, 2021.

Maron, H., Ben-Hamu, H., Serviansky, H., and Lipman, Y. Provably powerful graph networks. In *Advances in Neural Information Processing Systems*, pp. 2156–2167, 2019.

Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J. W., Songhori, E., Wang, S., Lee, Y.-J., Johnson, E., Pathak, O., Nazi, A., et al. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021.

Mitton, J., Senn, H. M., Wynne, K., and Murray-Smith, R. A graph VAE and graph transformer approach to generating molecular graphs. In *Graph Representation learning and beyong (ICLR Workshop)*, 2021.

Mohar, B. Some applications of laplace eigenvalues of graphs. In *Graph symmetry*, pp. 225–275. Springer, 1997.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4602–4609, 2019.

Morris, C., Rattan, G., and Mutzel, P. Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings. In *Advances in Neural Information Processing Systems*, volume 33, pp. 21824–21840. Curran Associates, Inc., 2020.

Niu, C., Song, Y., Song, J., Zhao, S., Grover, A., and Ermon, S. Permutation invariant graph generation via score-based generative modeling. volume 108 of *Proceedings of Machine Learning Research*, pp. 4474–4484, Online, 26–28 Aug 2020. PMLR.

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016a.

Oord, A. v. d., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A., and Kavukcuoglu, K. Conditional image generation with pixelcnn decoders. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, 2016b.

Papp, P. A., Martinkus, K., Faber, L., and Wattenhofer, R. DropGNN: Random dropouts increase the expressiveness of graph neural networks. volume 34, pp. 21997–22009, 2021.

Peixoto, T. P. Bayesian stochastic blockmodeling. *Advances in network clustering and blockmodeling*, pp. 289–332, 2019.

Peixoto, T. P. Merge-split markov chain monte carlo for community detection. *Physical Review E*, 102(1):012305, 2020.

Perraudin, N. *Graph-based structures in data science: fundamental limits and applications to machine learning*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 2017.

Petzka, H., Fischer, A., and Lukovnikov, D. On the regularization of wasserstein gans. In *International Conference on Learning Representations*, 2018.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017.

Ramakrishnan, R., Dral, P. O., Rupp, M., and Von Lilienfeld, O. A. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):1–7, 2014.

Sandfelder, D., Vijayan, P., and Hamilton, W. L. Ego-gnns: Exploiting ego structures in graph neural networks. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8523–8527. IEEE, 2021.

Segol, N. and Lipman, Y. On universal equivariant set networks. In *International Conference on Learning Representations*, 2019.

Serviansky, H., Segol, N., Shlomi, J., Cranmer, K., Gross, E., Maron, H., and Lipman, Y. Set2graph: Learning graphs from sets. In *Advances in Neural Information Processing Systems*, volume 33, pp. 22080–22091, 2020.

Shepard, R., Brozell, S. R., and Gidofalvi, G. The representation and parametrization of orthogonal matrices. *The Journal of Physical Chemistry A*, 119(28):7924–7939, 2015.

Shi, H., Fan, H., and Kwok, J. T. Effective decoding in graph auto-encoder using triadic closure. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 906–913, 2020.

Shi, J. and Malik, J. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

Shine, A. and Kempe, D. Generative graph models based on laplacian spectra? In *The World Wide Web Conference*, WWW '19, pp. 1691–1701, 2019.

Simonovsky, M. and Komodakis, N. Graphvae: Towards generation of small graphs using variational autoencoders. In *International conference on artificial neural networks*, pp. 412–422. Springer, 2018.

Sinclair, A. and Jerrum, M. Approximate counting, uniform generation and rapidly mixing markov chains. *Information and Computation*, 82(1):93–133, 1989. ISSN 0890-5401. doi: https://doi.org/10.1016/0890-5401(89)90067-9.

Vignac, C. and Frossard, P. Top-n: Equivariant set and graph generation without exchangeability. *arXiv preprint arXiv:2110.02096*, 2021.

Vignac, C., Loukas, A., and Frossard, P. Building powerful and equivariant graph neural networks with structural message-passing. In *NeurIPS*, 2020.

Wang, H., Wang, J., Wang, J., Zhao, M., Zhang, W., Zhang, F., Xie, X., and Guo, M. Graphgan: Graph representation learning with generative adversarial nets. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Yang, C., Zhuang, P., Shi, W., Luu, A., and Li, P. Conditional structure generation through graph variational generative adversarial nets. In *Advances in Neural Information Processing Systems*, pp. 1340–1351, 2019.

You, J., Liu, B., Ying, R., Pande, V., and Leskovec, J. Graph convolutional policy network for goal-directed molecular graph generation. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, 2018a.

You, J., Ying, R., Ren, X., Hamilton, W. L., and Leskovec, J. Graphrnn: Generating realistic graphs with deep autoregressive models. In *ICML*, 2018b.

Zhang, M. and Li, P. Nested graph neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

A. Model Implementation Details

In this section, we present additional information describing in details our general architecture presented in Figure 2 left.

The input noise used by each generator is sampled from a Gaussian distribution $z \sim \mathcal{N}(0,1)$, normalized to unit length and processed by a 4-layer MLP, with input, output, and hidden sizes of 100. Three different MLPs are used, each for a different generator. Every linear layer, including ones in MLPs except the final model output layer use layer normalization (Ba et al., 2016). We use GELU (Hendrycks & Gimpel, 2016) activation function throughout our architecture, except for the eigenvalue generator which uses gated activation units (Oord et al., 2016b;a). In our model, *we ignore the very first eigenvalue and eigenvector* as the first normalized Laplacian eigenvector only caries degree information.

Both PPGN models (Maron et al., 2019) have 8 layers (3 layers for QM9 experiments) with hidden dimension of 64, PointNetST models (Segol & Lipman, 2019) use 3-layer MLPs with base size of 32 (see Section A.2) and the CNN models have 4 convolutional layers $[5 \times 32, 9 \times 16, 17 \times 8, 25 \times 1]$ for the generator, with discriminator having same architecture in reverse. For each dataset, we select the number of spectral components k to be at least 2, but as small as is sufficient for the conditional graph generator to achieve good validity, while conditioned only on true spectral features. Our MolGAN* generator uses MLP of size $[128, 256, 512]$ following (De Cao & Kipf, 2018). For GG-GAN* we use a learned embedding size of 64.

The models are trained on a machine with two 64 Core AMD EPYC 7742 CPUs, 500GB RAM, and eight 24GB GeForce RTX 3090 GPUs. All of the individual runs only use one GPU at a time, except protein generation with GAN models. There we use 4 GPUs to achieve an effective batch size of 4. For SBM experiments we use a batch size of 5, for community and planar graphs a batch size of 10, and for QM9 following (De Cao & Kipf, 2018) we use a batch size of 128.

A.1. Provably Powerful Graph Network (PPGN)

The PPGN layer (Maron et al., 2019) consists of four operations on channel-first input $\boldsymbol{X} \in \mathbb{R}^{h \times n \times n}$. First, two versions of the matrix are built using two MLPs applied to each feature independently, i.e., repeated across the $n \times n$ dimensions.

$$\boldsymbol{M}_1 = \text{MLP}_1(\boldsymbol{X}) \in \mathbb{R}^{h_2 \times n \times n},$$
$$\boldsymbol{M}_2 = \text{MLP}_2(\boldsymbol{X}) \in \mathbb{R}^{h_2 \times n \times n}.$$

The h_2 matrices of size $n \times n$ are then multiplied as

$$\boldsymbol{M}^i = \boldsymbol{M}_1^i \boldsymbol{M}_2^i.$$

Eventually, the output is obtained by concatenating[6] $\boldsymbol{M}$ together with the original transformed by a third MLP:

$$\boldsymbol{X}^{t+1} = \text{MLP}_3(\boldsymbol{X}^t \parallel \boldsymbol{M}).$$

We use 8 layers, an embedding size of 64, and a skip connection from each layer (including inputs after the initial embedding linear layer) to the final readout. In the readout, a linear layer is applied on the concatenation of all of the layer outputs. For the generator, this is passed through an MLP to produce the adjacency matrix, while for the discriminator it is read out by applying an MLP over the sum of diagonal values and another MLP over the sum of off-diagonal values (Maron et al., 2019). Instance normalization is used after each PPGN layer. Disregarding the skip (concatenation) connection and instance normalization, our PPGN architecture matches the original implementation with one small, but important improvement. We normalize $\boldsymbol{M}^i = \boldsymbol{M}_1^i \boldsymbol{M}_2^i$ by $1/\sqrt{n}$, to avoid growth of the gradient and value variance. We apply a dropout of 0.1 before the final readout linear layer in PPGN.

Extending PPGN to generate node and edge features as well is simple. a) To produce the node features, a readout MLP that uses the concatenation of the diagonal elements from all of the layer outputs can be added. b) To produce the edge features, we change the adjacency readout MLP to produce not just a single channel matrix, but a multi-channel one, and interpret the new channels as different edge types or features. These node and edge features can then be fed into the discriminator.

A.2. PointNetST

A PointNetST layer (Segol & Lipman, 2019) is mean to process set inputs $\boldsymbol{X}^{n \times h}$ and is comprised of three MLPs: MLP_{feat}, MLP_{agg} and MLP_{cat}. All MLPs have different relative sizes, with respect their base hidden dimension h. MLP_{feat} is small ($[h, h, h]$) and is used for input preprocessing:

$$\boldsymbol{Y} = \text{MLP}_{\text{feat}}(\boldsymbol{X}) \in \mathbb{R}^{n \times h}.$$

MLP_{agg} enlarges the embeddings ($[h, h*2, h*4]$) and prepares them for global aggregation:

$$\bar{\boldsymbol{Y}} = \frac{1}{n} \sum_n \text{MLP}_{\text{agg}}(\boldsymbol{X}) \in \mathbb{R}^{1 \times 4*h}.$$

MLP_{cat} of size ($[h*4, h*2$, output dimension$]$) concatenates individual set element features $\boldsymbol{Y}$ with the (repeated) global set feature vector $\bar{\boldsymbol{Y}}$ and produces the final output:

$$\boldsymbol{X} = \text{MLP}_{\text{cat}}(\boldsymbol{Y} \parallel \bar{\boldsymbol{Y}}) \in \mathbb{R}^{n \times \text{output dimension}}.$$

[6]The concatenation operation is written $(\cdot \parallel \cdot)$

Whenever a global output vector is required in our architecture we apply another MLP on the mean-aggregated outputs of the PointNetST layer as discussed before. In such cases PointNetST output dimension is set to $h * 4$ and this final MLP is of size $[h * 2, h, \text{final output dimension}]$. When this readout MLP is used we apply a dropout of 0.1 to its inputs.

A.3. Dynamic Number of Nodes

To handle graphs with a varying number of nodes we create a binary mask using the conditioning number of nodes. We apply such $n \times h$ sized masks before any global set operation, such as mean (we also correct the mean for the true number of nodes). We also apply $n \times n \times h$ sized mask before any graph convolution or pooling (we again correct mean pooling for the true number of nodes). This allows us to use batched dense tensor operations with dynamic graph sizes in our model.

A.4. Gradient Penalty

To ensure good convergence of our GANs, we use Wasserstein loss (Wang et al., 2018) with gradient penalty (Gulrajani et al., 2017). In particular, we use the less restrictive gradient penalty formulation (Petzka et al., 2018):

$$\text{gp} = \gamma(\max\{0, ||\nabla d(\hat{\boldsymbol{x}})|| - 1\})^2,$$

where d is a discriminator and $\hat{\boldsymbol{x}}$ are inputs on which gradient penalty is computed. In general, $\hat{\boldsymbol{x}}$ is created by interpolation between true and fake samples. While interpolation of eigenvalues is straightforward, it is challenging to interpolate between eigenvectors. To interpolate eigenvectors, we first apply a canonical set ordering, which is achieved by flipping the eigenvector sign, such that the largest absolute value is positive, and sort them lexicographically. We then compute a cheap approximate interpolation, by interpolating the eigenvector matrices in Euclidean space and then down-projecting to a Stiefel manifold using a QR decomposition. Graph interpolation is similarly difficult because, while sets have a true canonical ordering, graphs do not. To compute gradient penalty for graphs, we randomly rewire them with $p = 0.1$, add Gaussian noise with the variance of 0.05 to the resulting adjacency and clip it to a range of $[0, 1]$. We do this for both, true and fake graphs. On top of this, we found it helpful to also compute gradient penalty directly on the (unpermuted) true and fake samples of graphs and eigenvectors.

As QM9 has node and edge features, we add random perturbations to those categorical features. This is achieved by adding Gaussian noise and with certain probability randomly permuting the indices of the one-hot vector. We use the same variance and permutation probability as before.

Adding Gaussian noise to otherwise discrete values is important, as our generator uses either sigmoid or softmax to produce them and we pass the resulting output from the generator directly to the discriminator without hard sampling as we find that improves the stability of the training.

B. Spectral Graph Features (Wavelet)

As graph generation is conditioned on the eigenvectors and eigenvalues, it is of interest to build a measure of spectral similarity between two different graphs. Therefore, we need the features to be a) invariant to node permutation, and b) descriptive for a specific range of eigenvalues. Our construction takes inspiration from graph spectrograms (Perraudin, 2017, Chapter 3.2) and diffusion wavelet embeddings (Donnat et al., 2018). Given a collections of P kernels $\phi_p : \mathbb{R}^+ \to \mathbb{R}$, we define spectral features for each node as:

$$\boldsymbol{S}[p, i] := \|[\phi(\boldsymbol{L})]_i\|_2^2 = \sum_{\ell=1}^{n} \phi(\lambda_\ell)^2 \boldsymbol{u}_\ell^2[i] \tag{2}$$

We obtain invariance with respect of the node order by computing the histogram of the node independently of p

$$\bar{\boldsymbol{S}}[p, q] = \text{hist}(\boldsymbol{S}[\boldsymbol{p}, \boldsymbol{\cdot}])[q] \tag{3}$$

The Maximum Mean Discrepancy (MMD) can then be computed using the vectorized version of $\bar{\boldsymbol{S}}$. We selected $P = 12$ ab-spline wavelet functions using the PyGSP (Defferrard et al.) for the kernels ϕ_p.

C. Datasets

As existing one-shot generative models only work with relatively small graphs (Krawczuk et al., 2020; De Cao & Kipf, 2018), we take two commonly used datasets for their evaluation:

Community-small Dataset (You et al., 2018b): This synthetic dataset consists of 100 random community graphs, which have between 12 and 20 nodes.

QM9 (Ramakrishnan et al., 2014): This dataset contains 134k organic molecules with up to 9 heavy atoms: carbon, oxygen, nitrogen, and fluorine. We follow the evaluation setup from (Simonovsky & Komodakis, 2018; De Cao & Kipf, 2018), by using 10k molecules for validation, 10k molecules for testing, and the remaining ones for training. Due to the very small size of graphs in this dataset, we reduce the number of PPGN layers used in the generator and discriminator to 3. As usually only validity, uniqueness and novelty are evaluated on this dataset. We use the number of valid, unique, and novel generated graphs for model selection instead of the MMD ratio.

To truly test our model's capability to generate large and valid graphs, we create two new challenging larger graph datasets:

Planar Graph Dataset: The dataset consists of 200 planar graphs with 64 nodes. The graphs are generated by applying Delaunay triangulation on a set of points that were placed uniformly at random.

Stochastic Block Model Dataset: We build 200 Stochastic Block Model graphs, with $[2, 5]$ communities (sampled at random) and $[20, 40]$ nodes (sampled at random) in each of them. The inter-community edge probability is 0.3 and the intra-community edge probability is 0.05. To determine whether a generated graph is a valid SBM we employ the following procedure: We use Bayesian inference (Peixoto, 2019), to estimate communities present in the graph and then refine the community assignments using a merge-split Markov chain Monte Carlo scheme (Peixoto, 2020). From these assignments we recover the inter- and intra- community edge probabilities. We then use a Wald test (Fahrmeir et al., 2007) to determine the probability that the predicted parameters match the original parameters. We only consider the graph valid, if the match probability is at least 0.9. On top of that, we check if the generated graph has between 2 and 5 communities and if each of them has between 20 and 40 nodes. If the graph does not meet these restrictions we deem it invalid.

We also use a large real-world graph dataset:

Protein Dataset (Dobson & Doig, 2003): This dataset consists of 918 protein graphs, where each protein graph is constructed by connecting nodes (amino acids) by an edge if they are less than 6 Angstroms away. These protein graphs have between 100 and 500 nodes.

D. Baselines

For the GraphRNN (You et al., 2018b) and GRAN (Liao et al., 2019) baselines, we use the settings recommended by their authors. For GraphRNN we use the generation procedure proposed originally, where 1024 graphs are generated, and then the ones with the most similar node count to test graphs are selected. To be compatible with this, for GRAN and our models we directly condition on the test graph node distribution.

E. Additional Background: Spectrum

In the following, we provide further information about how the first few eigenvalues and eigenvectors convey and summarize properties relating to the global structure (Figure 4):

Connectivity. The multiplicity of the zero eigenvalue equals the number of connected components (Chung & Graham, 1997).

Clusterability. The differences between the smallest successive eigenvalues of connected graphs reveal how easy it is to partition the graph. The phenomenon is particularly prominent for the difference of the first two eigenvalues, often referred to as the spectral gap, which by the Cheeger inequality (Alon & Milman, 1985; Alon, 1986; Sinclair & Jerrum, 1989) upper and lower bounds the graph expansion and conductance. Nevertheless, analogous results also hold for higher-order eigenvalues (Lee et al., 2014).

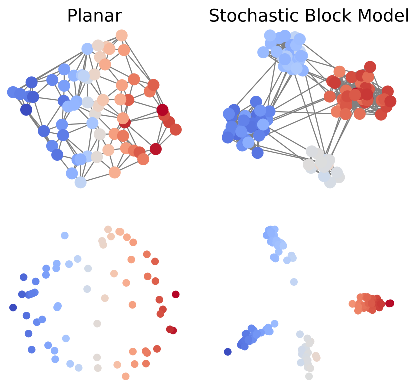


Figure 4. Normalized Laplacian eigenvectors which correspond to the lowest eigenvalues for different graphs. **Top**: Source graph. **Bottom**: The spectral embedding computed using the 2nd and 3rd eigenvectors of the graph normalized Laplacian captures the coarse graph structure. We do not use the 1st eigenvector as it only captures the degree distribution. Node color represents the value of the 2nd eigenvector.

Diameter. The first non-zero eigenvalue λ_1 can be used to bound the graph diameter as $\Delta \geq 1/2m\lambda_1$, with $m = |\mathcal{E}|$ being the number of edges of G.

Node embeddings. The first eigenvectors provide node embeddings that correlate with the geodesic distance. Eigenvectors have been used to embed the nodes of a graph (as in Laplacian Eigenmaps (Belkin & Niyogi, 2003) and in Diffusion Maps (Coifman et al., 2005)) or to perform clustering (as in graph spectral clustering (Shi & Malik, 2000)).

Link between the graph eigenvectors and the diffusion distance. One way to link spectral embedding with graph nodes is to use the graph diffusion distance (Hammond et al., 2013). Given a decreasing kernel function $\phi(x)$ (such as $\phi(x) = e^{-tx}$ for example), one can define a diffusion distance between two vertices v_i and v_j as

$$\begin{aligned}\mathcal{D}(v_i, v_j) &:= \|[\phi(\boldsymbol{L})]_{i,:} - [\phi(\boldsymbol{L})]_{j,:}\|_2^2 \\ &= \sum_{\ell=1}^{n} \phi\left(\lambda_\ell\right)^2 \left(\boldsymbol{u}_\ell[i] - \boldsymbol{u}_\ell[j]\right)^2,\end{aligned}$$

where $\phi(\boldsymbol{L}) = \boldsymbol{U}\phi(\boldsymbol{\Lambda})\boldsymbol{U}^\top$. Therefore, when embedding graph vertices using the eigenvectors associated with the

k lowest eigenvalues, one obtains a map where points are separated with a specific graph diffusion distance.

F. Graph samples

In this section, we showcase uncurated graph samples produced by the different models. Figure 5, 6, 7 and 8 displays respectively planar, SBM, protein and community graphs. In every row, each model is conditioned on the same number of nodes. For protein graphs (Figure 7) all models fail to consistently produce connected graphs, thus following previous work (You et al., 2018b; Liao et al., 2019) we display only the largest connected component. From these sample graphs, it is easy to see that spectral conditioning greatly improves the quality of the generated graphs and is the only model that truly respects hard constraints imposed on the training set, such as the maximum number of nodes allowed in an SBM component or graph planarity. We also provide uncurated unique molecule samples produced by SPECTRE in Figure 9. While the given molecules are valid, in the sense that they can theoretically exist, they can be very unstable. It is also worth noting that while spectral conditioning prevents total mode collapse in this molecule generation task, the variety of the generated molecules is still somewhat low. Meaning, that while they are non-isomorphic they are mostly comprised of similar substructures.

G. Adjacency matrices of MolGAN* proteins

In Figure 10, we show, that MolGAN* produces nearly identical adjacency matrices for different generated protein graphs. Observe, that while the node count differs in these adjacency matrices, as we cut the matrix based on the n the model was conditioned on, resulting in overall non-isomorphic graphs, the edge connections are almost identical.

Model	Mean Edit Distance
Test Dataset	78.8
GraphRNN	83.7
GRAN	48.8
MolGAN*	17.6
GG-GAN (RS)*	75.4
GG-GAN*	90.9
SPECTRE ($k = 16$)	97.0

Table 6. Mean edit distance as percentage of different edges between generated protein graphs. Consistent node ordering is assumed.

To show this effect more clearly, we evaluate the percentage of different edges between any two graphs in the generated set in Table 6. The edit distance is computed assuming that the node ordering is consistent between the graphs. Given that all models, except GG-GAN (RS)*, either directly use an ordering or some (semi)-fixed initialization this is a reasonable assumption and the resulting value gives us the upper bound on the true mean edit distance. When comparing two graphs of different size, the larger graph is clipped to match the size of the smaller one. It is clear to see, that the mean edit distance between graphs generated by MolGAN* is very low and much lower than between graphs produced by any other model.

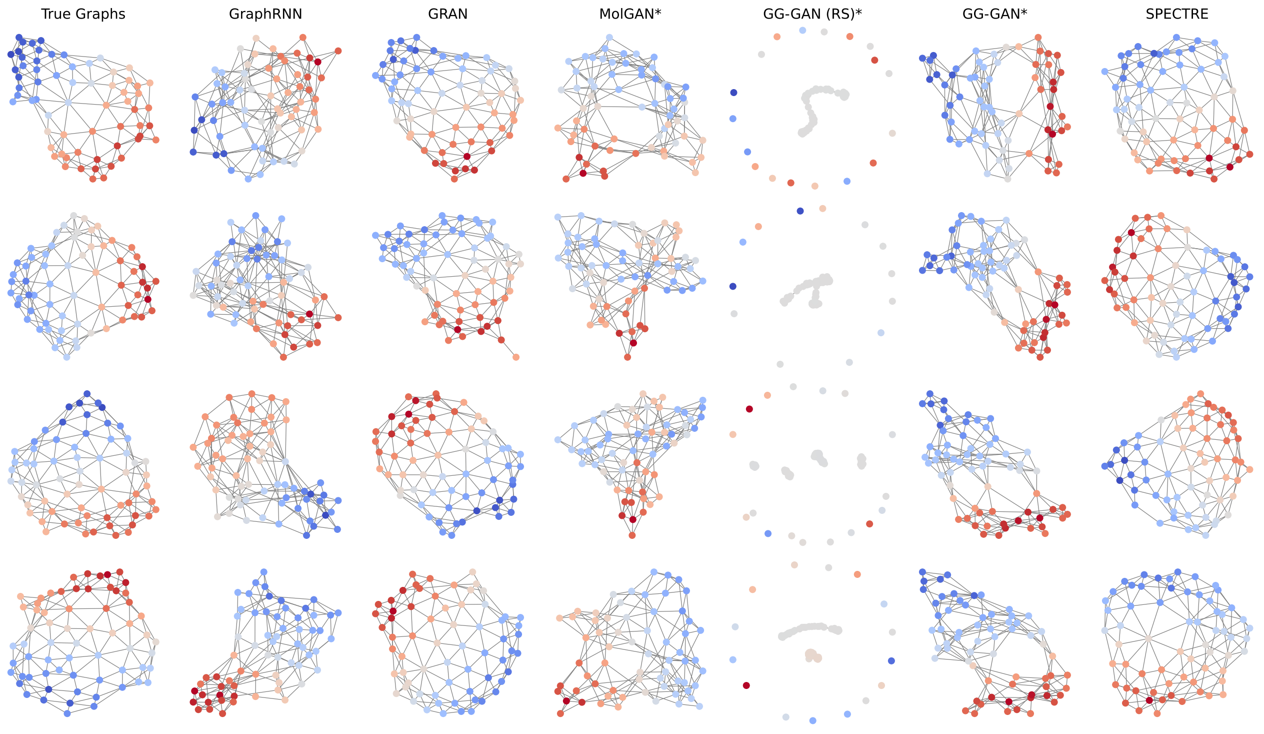


Figure 5. Uncurated set of sample Planar graphs produced by the models. Each row is conditioned on the same number of nodes.

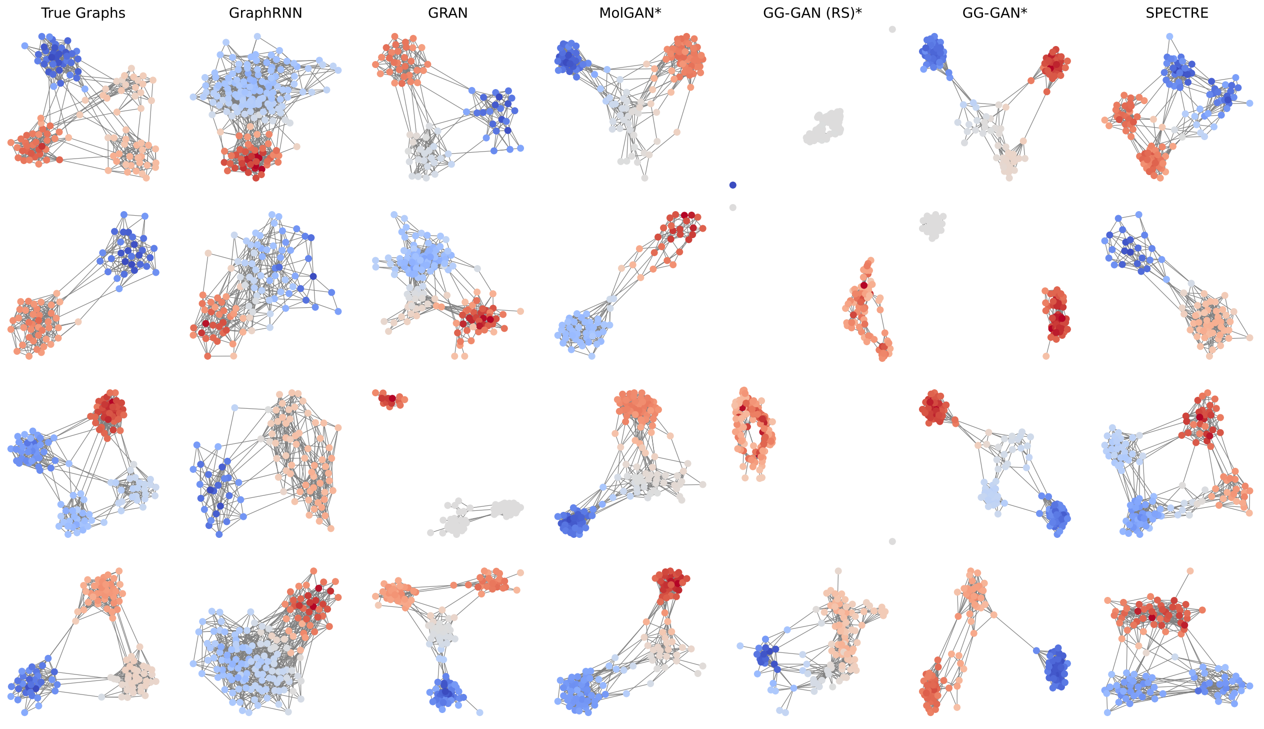


Figure 6. Uncurated set of sample Stochastic Block Model graphs produced by the models. Each row is conditioned on the same number of nodes.

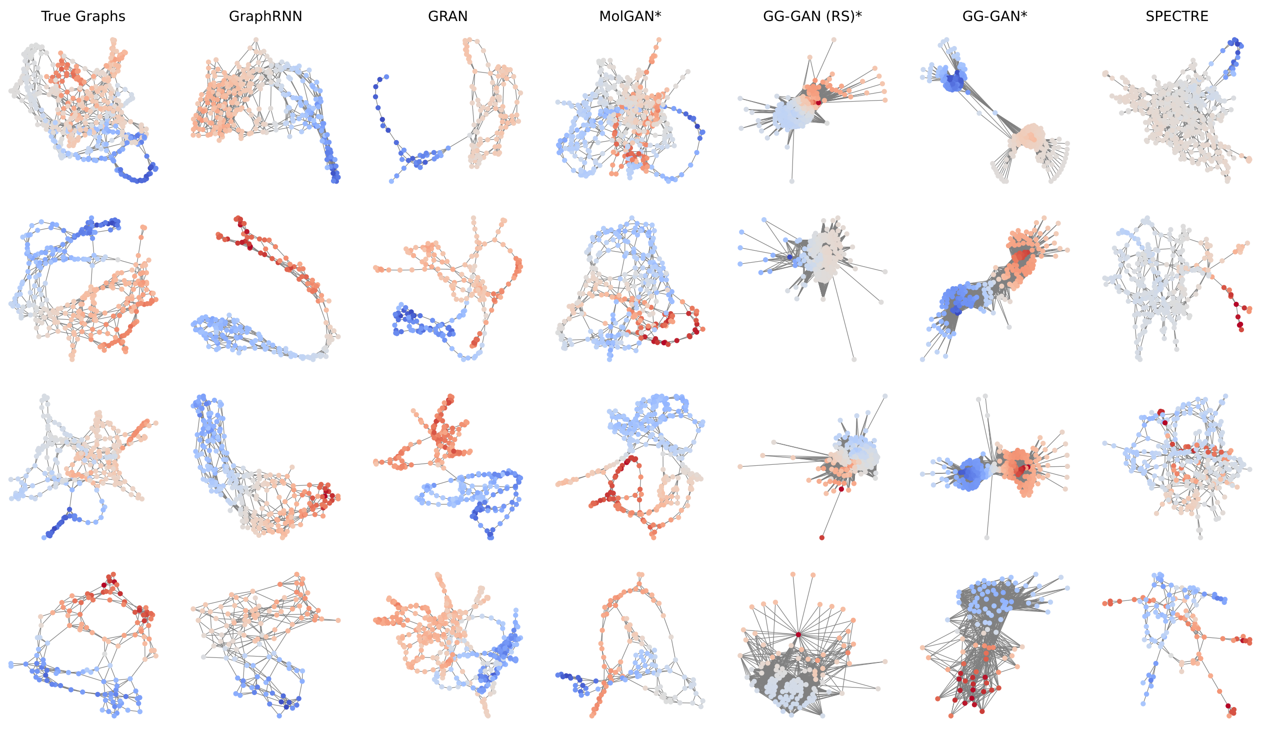


Figure 7. Uncurated set of sample Protein graphs produced by the models. Each row is conditioned on the same number of nodes.

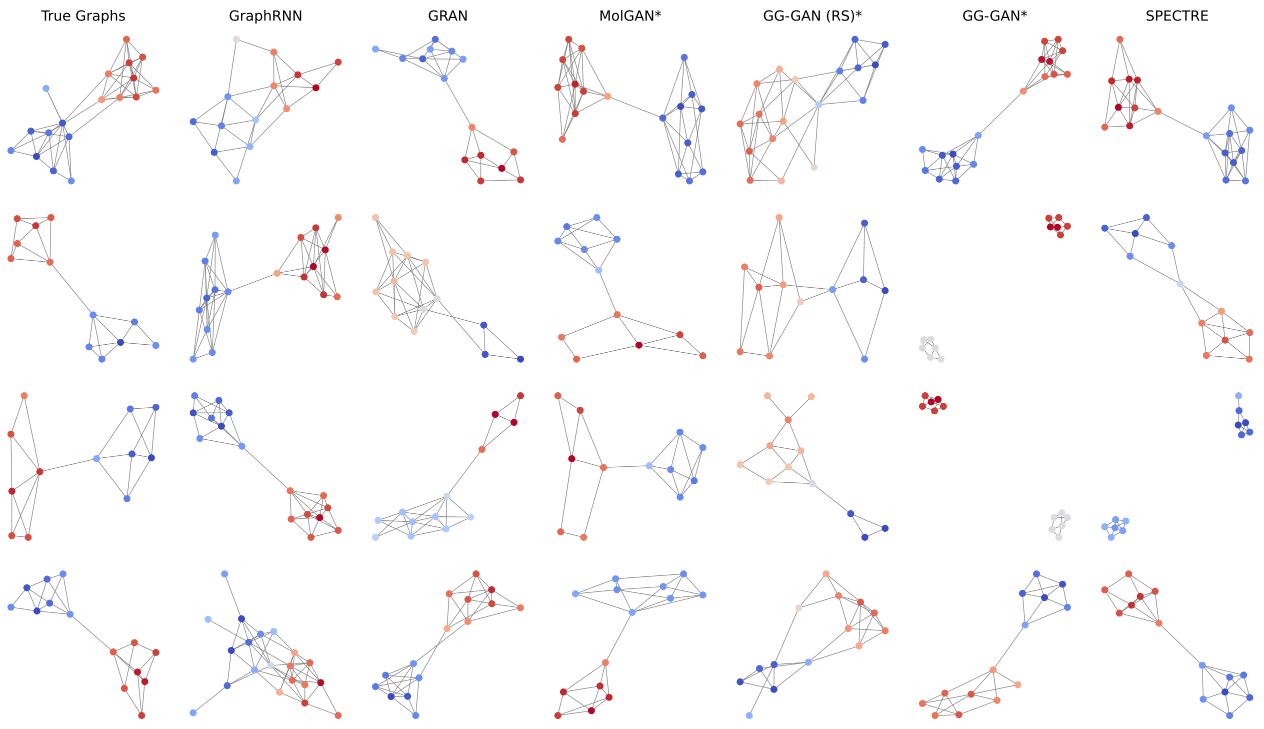


Figure 8. Uncurated set of sample Community graphs produced by the models. Each row is conditioned on the same number of nodes.

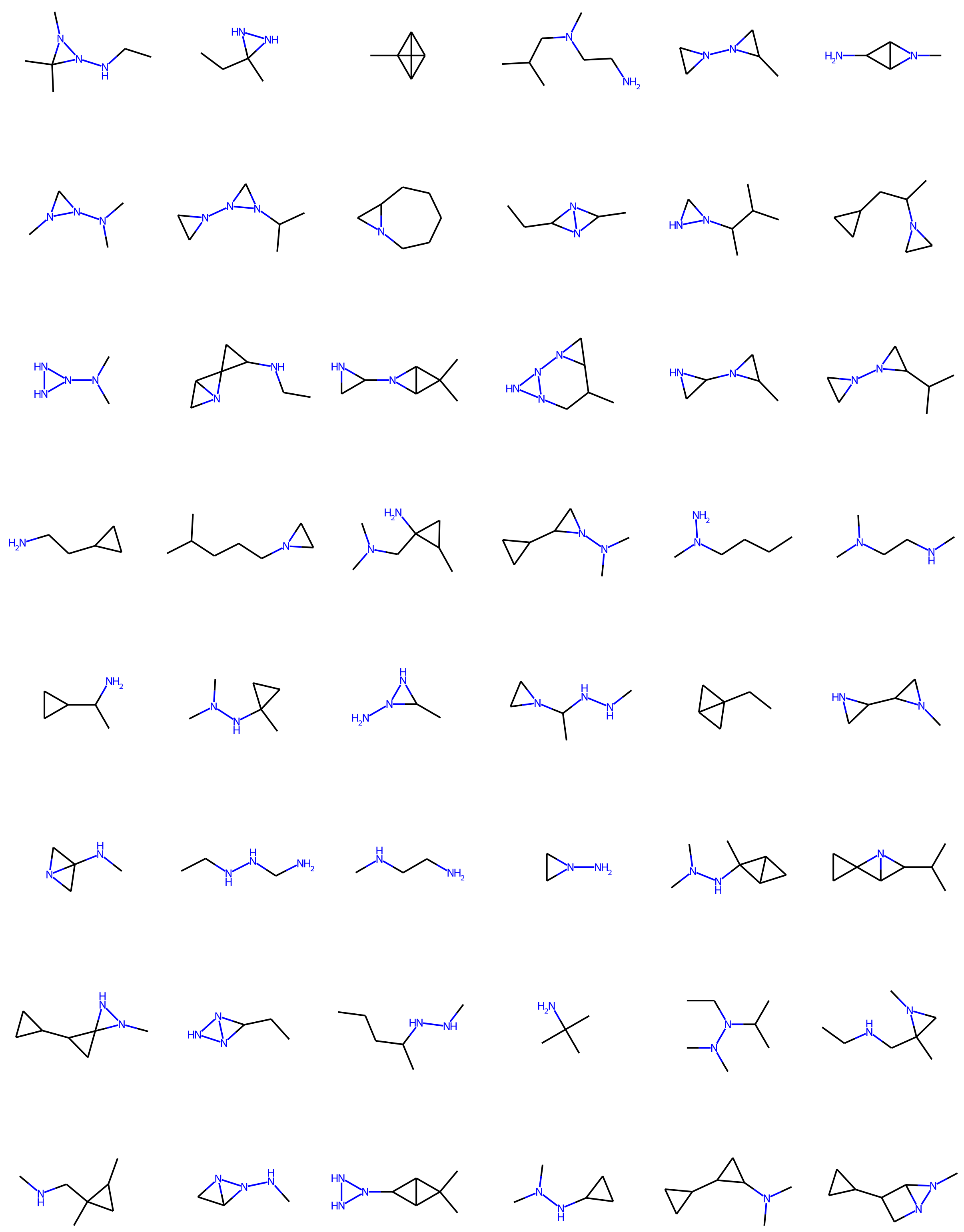


Figure 9. Uncurated set of unique sample QM9 molecules produced by SPECTRE.

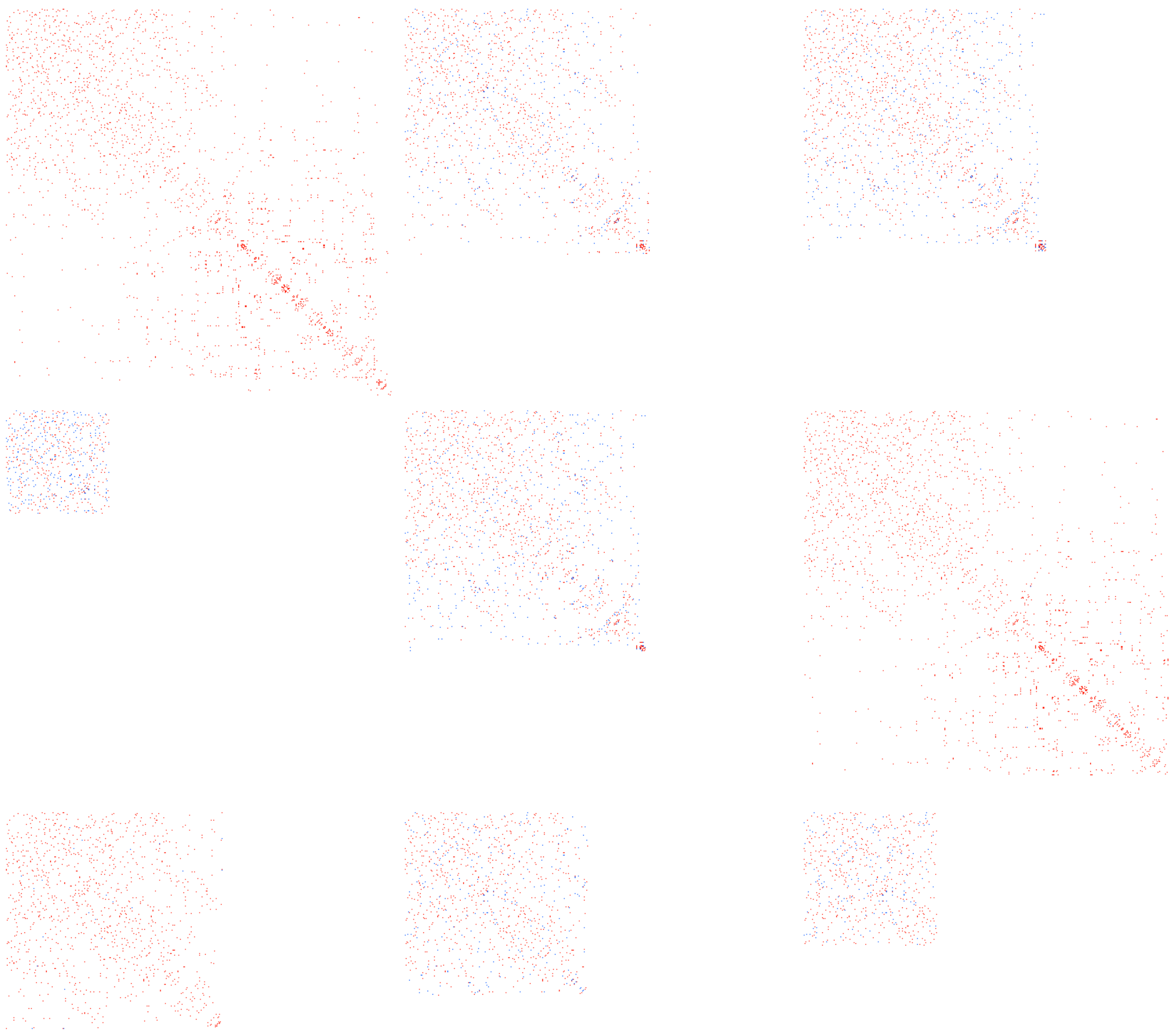


Figure 10. Adjacency matrices of 9 random proteins produced by MolGAN*. For each graph we highlight in red the edges that are also present in the first graph. All graphs are sub-graphs of the larger graph with a few changed edges.