



HAL
open science

GRAN Is Superior to GraphRNN: Node Orderings, Kernel- and Graph Embeddings-Based Metrics for Graph Generators

Ousmane Touat, Julian Stier, Pierre-Edouard Portier, Michael Granitzer

► To cite this version:

Ousmane Touat, Julian Stier, Pierre-Edouard Portier, Michael Granitzer. GRAN Is Superior to GraphRNN: Node Orderings, Kernel- and Graph Embeddings-Based Metrics for Graph Generators. 9th International Conference on machine Learning, Optimization and Data science - LOD 2023, Sep 2023, Grasmere, Lake District, England, United Kingdom. pp.430-444, 10.1007/978-3-031-53969-5_32. hal-04702900

HAL Id: hal-04702900

<https://hal.science/hal-04702900v1>

Submitted on 24 Sep 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

GRAN is superior to GraphRNN: node orderings, kernel- and graph embeddings-based metrics for graph generators

Ousmane Touat², Julian Stier¹[0000-0001-5710-9240], Pierre-Edouard Portier²,
and Michael Granitzer¹[0000-0003-3566-5507]

¹ University of Passau

julian.stier@uni-passau.de

² INSA Lyon

ousmanetouat@outlook.com

2023/07/04 ~ 432623d

Abstract. A wide variety of generative models for graphs have been proposed. They are used in drug discovery, road networks, neural architecture search, and program synthesis. Generating graphs has theoretical challenges, such as isomorphic representations – evaluating how well a generative model performs is difficult. Which model to choose depending on the application domain?

We extensively study *kernel-based metrics* on distributions of graph invariants and *manifold-based* and *kernel-based metrics* in *graph embedding space*. Manifold-based metrics outperform kernel-based metrics in embedding space. We use these metrics to compare GraphRNN and GRAN, two well-known generative models for graphs, and unveil the influence of node orderings. It shows the superiority of GRAN over GraphRNN - further, our proposed adaptation of GraphRNN with a depth-first search ordering is effective for small-sized graphs.

A guideline on good practices regarding dataset selection and node feature initialization is provided. Our work is accompanied by open-source code and reproducible experiments.

Keywords: Graph Generative Models · Graph Neural Network · Graph Manifolds Metrics · Geometric Deep Learning

1 Introduction

For a few years now, graph generation via deep generation models has been a subject of particular attention, notably because of its promising applications, especially in pharmacy. Beyond these specified applications, graphs are also central to many fields, such as physics or sociology.

The attention on graph generation has been accentuated with the arrival of autoregressive generation models [27,14,13], able to build a graph with a sequence of elementary steps (e.g., the addition of a node and its connection to the current graph). These models are among the most scalable graph generator while still being able to approach graph distribution better. The main challenge

of these generators is that since they must learn to generate graphs in sequence, these models deal with the problem of non-unique representation of graphs, where the node orderings dictate how to represent the graph in sequence. Since there can be many such orderings and, thus, ways of sequentially representing graphs, the sequential generation problem can quickly become challenging to model. One approach is to model the problem by estimating a lower bound on the maximum likelihood, considering only a subset of the set of node orderings of a graph.

Another challenge is the evaluation of the performance of these models. Until now, the primary method of evaluation consists in using metrics based on the computation of kernels from statistical measures related to the studied graphs, computing, for example, the Maximum Mean Discrepancy of the node degree distribution [27,14,8,16]. With advances in graph neural networks (GNN), evaluation methods using data embeddings, originally applied to image generation models, have recently been transposed to the domain of generative models for graphs [4,22]. Nevertheless, these techniques still need to be sufficiently studied, and their use is just beginning to spread.

Our **contributions** comprise 1st/ extensive studies on recent evaluation techniques based on graph embeddings which are learned through pre-trained graph classifiers, on several factors such as the pre-training dataset or the node feature initialization technique, giving insights on good practices for graph evaluation accompanied with a *reproducible code repository*. 2nd/ *independent and new empirical studies* on GraphRNN and GRAN compared using several node orderings, adding experiments with a *newly proposed node ordering* (depth-first search) on GraphRNN, using both graph-embeddings- and kernel-based evaluation techniques. These contributions show new insights on graph embeddings-based metrics for generative models of graphs and motivate using depth-first-search as a node ordering when learning GraphRNN on small graph datasets.

2 Metrics for Generative Models of Graphs

Generative models of graphs are trained on graph datasets and are then used to sample new graphs. Good evaluation metrics should be able to rank graph generative models on how well they capture the characteristics of the underlying graph distribution P_G from the training set samples $X_G^{train} \sim P_G$. The graph edit distance is a standard similarity measure. However, in general, its computation is NP-hard, [28]. First papers on deep graph generation used graph statistics such as node degree and clustering coefficients to measure the performance of generative models [3].

2.1 Kernel-based Evaluation Metrics

In their work, You et al. employ Maximum Mean Discrepancy **MMD** as a measure of distance between graph invariants, such as node degree or the number of 4-orbits [27]. This allow them to assess how effectively the generated graphs

from their trained model capture properties found in reference graphs. MMD between two drawn distribution x and y based on a kernel k is defined as:

$$\text{MMD}^2(p||q) = \mathbb{E}_{x,y\sim p}[k(x,y)] + \mathbb{E}_{x,y\sim q}[k(x,y)] - 2\mathbb{E}_{x\sim p,y\sim q}[k(x,y)] \quad (1)$$

For graphs, p and q refer to the distributions of computed graph invariants, such as the node degree distribution from the generated graphs and the set of reference graphs. Those metrics are currently widely used in the graph generation community [16].

2.2 Graph Embeddings-based Evaluation Metrics

We introduce the graph embeddings-based metrics used in our study: *Fréchet Distance*, *Precision*, *Recall*, *Density*, *Coverage* and *F1-Score* use graph representations learned by the *Graph Isomorphism Network* model.

Graph Isomorphism Network Graph Isomorphism Network (GIN) is a robust message-passing neural network architecture, usually designed for graph classification tasks [26]. We define $G = (V, E)$ as an undirected graph defined by its set of n nodes denoted V , and a set of e edges E . A node $v_i \in V$ have its d -dimensional node feature $X \in \mathbb{R}^{e \times d}$. The neighborhood $\mathcal{N}(v_i)$ is a set of nodes connected to v_i . The hidden representation at the k -th layer $h^{(k)}(v_i)$, with $\phi^{(k)}$ being the node aggregation function, is modeled using a multilayer perceptron (MLP):

$$h^{(k)}(v_i) = \text{MLP}^{(k)}((1 + \epsilon^{(k)})h^{(k-1)}(v_i), \phi^{(k)}(\{h^{(k)}(u) \mid u \in \mathcal{N}(v_i)\})) \quad (2)$$

The graph level readout ξ_{rd} , obtaining graph-level representations, “can be a simple permutation invariant function such as summation or a more sophisticated graph-level pooling function” [26] that can be either a sum or an average over graph representations of each layer. We get the graph’s embeddings by performing such sum “ \sum ” or concatenation “ \parallel ” over pooled node representations at each layer of the GIN [26]:

$$x_{\parallel} = \parallel \left(\xi_{rd}(\{h^{(k)}(u) \mid u \in \mathcal{N}(v_i)\}) \mid l \in \{1, 2, \dots, K\} \right) \quad (3)$$

$$x_{\Sigma} = \sum_{l=1}^K \xi_{rd}(\{h^{(k)}(u) \mid u \in \mathcal{N}(v_i)\}) \quad (4)$$

For the first expression, Xu et al. proves that GIN is equivalent to the Weisfeiler-Lehman Isomorphism Test (1-WL) [25].

Pre-training GIN for Graph Feature Extraction To evaluate the quality of generative image models, metrics such as the *Fréchet Inception Distance* (FID) [10] have been used and are computed based on representations from image embedding space. Image embeddings are obtained from learned image classification

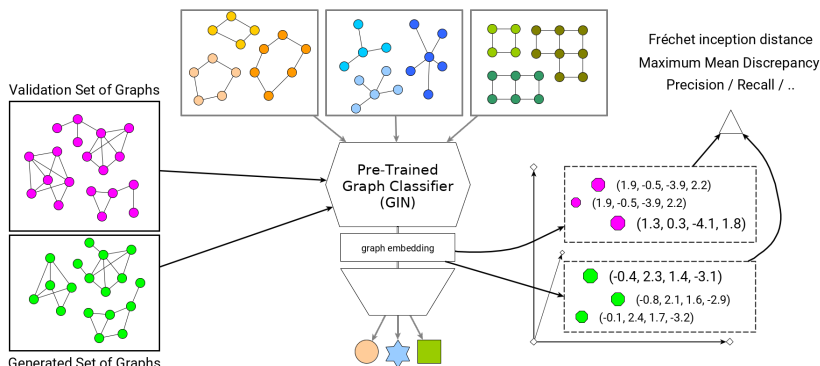


Fig. 1: Embedding-based metrics are computed in embedding spaces of a GNN such as the GIN [26]. After training this GNN on classifying types of graphs, we compare two sets of graphs by computing metrics based on their graph representations – a $N \times d$ matrix, with N the number of graphs and d the graph manifold size. The reference set of graphs is compared to a trained model generated set of graphs, to assert the generative model performance.

models such as Inception v3 on ImageNet [21]. This approach was first explored on small graphs using GIN [4]. Similarly, we use metrics in *graph* embedding space as sketched in figure 1. *Graph* embeddings are extracted from a graph neural network trained on a multi-class classification task, where each class represents one type of graph, which has also been used in training for the generative tasks (i.e., grid graphs, community graphs).

Graph Embeddings-based Metrics Let $(X_1, \dots, X_N) \in \mathbb{R}^{N \times d}$ be vector representations of size d in the graph embedding space of the pre-trained GIN obtained from graphs (G_1, \dots, G_N) through equations 3 or 4.

The first studied metric, called Fréchet Distance FD , is a popular metric derived from FID [10] used in the image generation domain. This metric compares both generated and authentic samples of graphs in the graph embedding space, taken as Gaussian distributions with means and covariances $\mathbf{p}_{gen} = (m_{gen}, C_{gen})$ and $\mathbf{p}_{real} = (m_{real}, C_{real})$. Fréchet Distance, accounting for the quality and diversity of generated samples, is computed as:

$$d^2(\mathbf{p}_{gen}, \mathbf{p}_{real}) = \|m_{gen} - m_{real}\|_2^2 + \text{Tr}(C_{gen} + C_{real} - 2(C_{gen}C_{real})^{1/2}) \quad (5)$$

We also use two pairs of metrics, denoted Data-Manifold metrics, based on multi-dimensional representation manifolds [12,15], meaning sets of close neighbors forming a hypersphere. On graphs, with $B(x, r)$ a ball of center x and radius r , and $\text{NND}_k(X_i)$ the distance between X_i , the representation of one graph extracted from the GNN, and the k th nearest neighbor in the embedding space,

the manifold is defined as

$$\text{manifold}(X_1, \dots, X_N) := \bigcup_{i=1}^N B(X_i, \text{NND}_k(X_i)) \quad (6)$$

The two pairs of metrics denoted *Precision and Recall*, and *Density and Coverage*, are used to compute *F1 Score*, a metric also accounting for the quality and diversity of the generated sample.

Precision and Recall [12] are respectively the proportion of generated samples being in the manifold of the real objects and the proportion of the real objects in the manifold of generated samples. Finally, *Density and Coverage* [15] are the proportion of the number of real object manifolds for each generated sample and the proportion of real object manifolds containing at least one generated sample.

Node feature engineering on unattributed Graphs Generally, GNNs are very powerful when the graphs are attributed, where the nodes are associated with natural feature vectors corresponding to known attributes. With unattributed graphs, we have no information other than the graph’s topology given by its adjacency matrix. In order to compute *FD* using GIN, [4] used a one-hot vector encoding of node degrees as input node feature vector, as in [26] on social network graphs. Such feature vectors have been studied and shown to be among the feature initialization that gives overall good performance in graph classification [7,5]. Recently, other feature engineering has been explored, using random node features initialization [1,17]. [17] notably takes up GIN, adding the ability to generate a random component at each pass through GIN, demonstrating its power on some problems that GIN alone cannot solve. We also implemented this approach alongside the existing node feature initialization approaches explored for generative graph evaluation.

Related Work First attempts of importing GAN-related metrics such as the Fréchet Inception Distance *FID* [10] *Improved Precision and Recall* [15] were implemented using recent Graph Neural Network techniques such as Graph Isomorphism Network *GIN* [26], but were only marginally used to implement graph version of existing metrics to work on [4,22]. O’bray and al. [16] investigated kernel-based evaluation metrics using graph invariants, especially MMD on graph statistics. They show how the kernel choice and fine-tuning of parameters are critical as the metric may become unreliable, giving an unstable ranking of the compared generative model. Thompson et al. [23] recently investigated graph embeddings-based metrics, justifying the potential of using untrained GNN to extract graph embeddings, using similar experiments as in [16]. We took [16] approach for systematically evaluating those graph embeddings-based metrics. Nevertheless, we developed our paper independently from [23] as we differentiate on studying different hyperparameter settings and their impact on the behavior of the graph embeddings metrics, such as the choice of node feature initialization and pre-training dataset choice.

3 GraphRNN & GRAN

Both Graph Recurrent Neural Networks **GraphRNN** [27], and Graph Recurrent Attention Networks **GRAN** [14] are autoregressive graph generators. We will then present the autoregressive graph generation problem before detailing how GRAN and GraphRNN work.

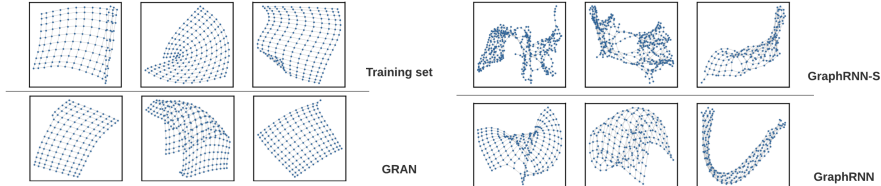


Fig. 2: Visualisation of generated 2d grid graphs from GRAN, GraphRNN-S and GraphRNN-RNN compared to originally generated grid graphs used as *training set*.

Autoregressive graph generation problem We define $G = (V, E)$ as an undirected graph defined by its set of nodes V and edges between nodes E . With the node ordering π of G , and $V^\pi = (v_1, \dots, v_n)$ the ordered set of nodes, we have A^π the adjacency matrix of G ordered by π , a $n \times n$ boolean matrix representing G . Graph generative models learn a probability distribution P_G by learning the sequence $S^\pi = (S_n^\pi, \dots, S_1^\pi)$ with S_i^π corresponding to the i -th row in the adjacency matrix A^π . We usually solve the problem of graph generation by maximizing the log-likelihood over $\log(P_G)$. Learning to generate rows of adjacency matrices in sequences is difficult because the search space of different adjacency matrices associated to one graph can be huge, up to $n!$, due to the factorial nature of node permutations. Both **GRAN** and **GraphRNN** precisely use chosen node orderings to alleviate this sampling complexity.

Graph Recurrent Neural Networks **GraphRNN** are based on two RNNs to model the sequence S^π . The first RNN is called graph-level RNN, used to save the current state of the generated graph. The second RNN is called node-level RNN, generating the rows of the adjacency matrix from the current state of the graph given by the graph-level RNN. Breadth-First-Search (BFS) restricts the maximum node permutations for one graph. BFS ordering is also used to improve scalability by allowing the model to work on adjacency matrices presenting a known lower graph bandwidth, reducing the model complexity from n^2 to $n \cdot m$ with $m \leq n$.

Graph Recurrent Attention Networks **GRAN** use attention-based Graph Neural Networks to generate a graph in, at best $O(n)$ autoregressive steps. Each step generates rows of the adjacency matrix representation of G . GRAN allows for generating multiple rows at once for a quality/time efficiency tradeoff. Compared to GraphRNN, this model does not save all the previous graph representations and only works on the current graph representation for the autoregressive

step. This model also allows using several sets of node orderings (called canonical node orderings) that could improve generation by having a higher lower bound of the maximum likelihood $\log(P_G)$. But this comes with a tradeoff between training time and sample quality.

Using several node ordering on GraphRNN and GRAN GraphRNN uses BFS to improve its scalability, but this model still suffers from having a relatively high model complexity $O(n^2)$ and having to deal with very long-term dependencies due to the presence of RNN blocks, which prevents achieving scalability large enough for applications that need it [14]. Therefore, the interest in using GraphRNN would be on small graph datasets, where we have already seen an excellent performance from this model [6]. At this scale of graphs scalability benefits of BFS become irrelevant. We present the use of Depth-First-Search (DFS) as a node ordering policy, which also makes the problem of graph generation easier by reducing the search space of the graph representation. However, how much do the performance and quality change when DFS as the node ordering?

GRAN already allows for the possibility of using different node orderings to train the model. We compare the use of these node orderings on GRAN to accompany the comparative study on GraphRNN. The main difference between node ordering in both models is that in GRAN the matrix representations of the dataset graphs are fixed at the beginning, so the node ordering function is only used at the beginning of the training. In GraphRNN, graphs matrix representations change through training because the stochastic node ordering function (e.g., GraphRNN BFS function is computed with a random starting node) is called each time a graph is picked from the dataset during the training loop to give a node ordering that can be different from the one computed previously for the same graph.

Related Graph Generative Models Early graph generative models are low parameterized and not powerful enough to model complex dependencies in real-world networks. Graph generators use deep learning techniques to mimic graphs from real-world data. Models such as variational auto-encoders [19,11] or generative adversarial networks [3] have been studied for graphs. Breakthroughs in quality and scalability happened with autoregressive models like GraphRNN [27]. Due to their sequential nature, auto-regressive formulations must restrict the space of studied node orderings to make the problem tractable. This problem has been studied e.g., in DGMG, and GraphRNN [13,20,27].

4 Experiments

Our study has two concerns: 1/ Studying the behavior of graph embeddings-based metrics using two experiments where we vary several hyperparameters. 2/ See whether GRAN or GraphRNN generates better graphs in terms of quality and how node orderings can impact both models' performances. All experiments were done using the same Python environment (Python 3.7 + Pytorch 1.8.1).

Base dataset configuration				
	Mean Rank ↓	FD ↓	F1 PR ↑	F1 DC ↑
Re-sampled	1	-0.0010±0.0007	1.0±0.0	0.994±0.003
GRAN	2	41004±14957	0.845±0.006	0.79±0.030
GraphRNN-RNN	3	91512±24215	0.040±0.010	0.019±0.009
GraphRNN-S	4	319669±94417	0.0±0.0	0.0±0.0
Ladder dataset configuration				
	Mean Rank ↓	FD ↓	F1 PR ↑	F1 DC ↑
Re-sampled	1	-0.0010±0.0008	1.0±0.0	0.992±0.005
GRAN	2	30773±23098	0.870±0.041	0.81±0.10
GraphRNN-RNN	3	127572±29497	0.042±0.015	0.02±0.01
GraphRNN-S	4	362265±149052	0.001±0.006	0.003±0.001
Full dataset configuration				
	Mean Rank ↓	FD ↓	F1 PR ↑	F1 DC ↑
Re-sampled	1	-0.0002±0.0001	1.0±0.0	0.990±0.006
GRAN	2	737±340	0.880±0.016	0.847±0.039
GraphRNN-RNN	3	45270±16467	0.046±0.006	0.027±0.007
GraphRNN-S	4	75501±29895	0.0±0.0	0.0±0.0

Table 1: Sum Graph Manifolds metrics on Gr dataset from 10 independently trained GIN. All of the metrics show consistent ranking fidelity with the base ranking hypothesis and stability. F1 Score metrics are more stable in regards to the dataset configuration, with low variance compared to FD.

Datasets For both parts, we focus on synthetic datasets for two reasons: First, some synthetic graphs datasets are based on mathematical models such as Barabasi-Albert **BA** and Watts-Strogatz **WS** models or have directly identifiable topologies such as community datasets. Finally, using such synthetic datasets enable us to directly compute benchmarks for a *perfect* generation being the synthetic graph generator itself, which is further explained in the Results section.

The following datasets are used for our experiment: (1) **BA**: 500 Barabasi-Albert graphs [2] with $100 \leq n \leq 200$ using the B-A model, using the integrated `networkx.barabasi_albert_graph` function [9], with parameter $k = 4$.

(2) **WS**: 500 Watts-Strogatz graphs [24] with $100 \leq n \leq 200$ using the W-S model, where each node is linked to its four nearest neighbor and the probability of rewiring is 0.1.

(3) **C2L**: 500 2-community graphs using the synthetic community generator from [27], where communities are formed with Erdos-Renyi graphs of $6 \leq n_{community} \leq 10$ and $p = 0.7$ from which $0.1 \times n$ edges between communities are added uniformly at random.

(4) **C2S**: 500 2-community graphs using the synthetic community generator from [27], where communities are formed with Erdos-Renyi graphs of $30 \leq n_{community} \leq 80$ and $p = 0.3$ from which $0.05 \times n$ edges between communities are added uniformly at random.

(5) **Gr**: 100 2D grid graphs with $100 \leq n \leq 400$ using the integrated `networkx.2d_grid_graph` function.

(6) **Ld**: 500 ladder graphs with $100 \leq n \leq 200$ using the integrated `networkx.ladder_graph` function.

Base dataset configuration				
	Mean Rank ↓	FD ↓	F1 PR ↑	F1 DC ↑
Re-sampled	1	-0.001±0.001	1.0±0.0	0.994±0.003
GRAN	2	26639±14495	0.852±0.010	0.789±0.016
GraphRNN-RNN	3	65917±24978	0.042±0.005	0.026±0.006
GraphRNN-S	4	201977±89750	0.0±0.0	0.0±0.0
Ladder dataset configuration				
	Mean Rank ↓	FD ↓	F1 PR ↑	F1 DC ↑
Re-sampled	1	-0.002±0.002	1.0±0.0	0.990±0.003
GRAN	2	10228±11339	0.879±0.029	0.832±0.069
GraphRNN-RNN	3	91113±24230	0.047±0.005	0.034±0.011
GraphRNN-S	4	212305±78070	0.003±0.008	0.0006±0.001
Full dataset configuration				
	Mean Rank ↓	FD ↓	F1 PR ↑	F1 DC ↑
Re-sampled	1	-6e-4±-6e-4	1.0±0.0	0.990±0.007
GRAN	2	1264±360	0.885±0.0147	0.862±0.029
GraphRNN-RNN	3	59471±9750	0.048±0.001	0.033±0.003
GraphRNN-S	4	91310±19924	0.001±0.006	0.0003±0.001

Table 2: Concatenation Graph Manifolds metrics on Gr dataset from 10 independently trained GIN. All of the metrics show consistent ranking fidelity with the base ranking hypothesis and stability. F1 Score metrics are more stable in regards to the dataset configuration, with low variance compared to FD.

(7) **ER**: we create this dataset with one of the previously mentioned graph datasets, where an ER graph is created with $n = \max(n_g, \text{label}(g) = i)$ and parameter $p = \frac{e_g}{n_g^2}$, this dataset contains the same amount of graph as the linked graph dataset (e.g., we would generate 500 ER graphs "equivalent" to the **BA** dataset, or 100 ER graphs "equivalent" to the **Gr** dataset)

4.1 Testing Evaluation Metrics with Graph Embeddings

We evaluate the behavior of graph embeddings-based metrics computed using Graph Neural Networks trained on a graph classification task. We want to observe how changes in the training process, such as the initialization of node feature vectors or the choice of the training dataset, would impact the behavior of the resulting metrics and gain insights into optimizing the training process to get more expressive metrics.

Experiment Description In their work, Obray and al. discussed criteria for evaluating comparison metrics, among which we find expressiveness, where the ideal comparison metric can rank graph samples effectively [16]. To evaluate expressiveness, we use two experiments: First, in the **Perturbation experiment**, we compare a set of graphs to a copy set but perturbed with interpolation between those graphs and rewired ER-graphs as in [27,16]. We expect the metrics to react monotonically to those perturbations, keeping the similarity rank of each perturbed graph set with the reference set. We also look for metric stability if

With constant feature				
	Mean Rank	FD ↓	F1 PR ↑	F1 DC ↑
Re-sampled	1	-8e-5±5e-5	1.0±0.0	0.975±0.010
GRAN	2	211±29	0.973±0.015	0.973±0.015
GraphRNN-RNN	4	30904±2275	0.007±0.010	0.052±0.0001
GraphRNN-S	3	26335±2016	0.60±0.027	0.056±0.010
With random feature (rGIN)				
	Mean Rank	FD ↓	F1 PR ↑	F1 DC ↑
Re-sampled	1	0.87±0.82	0.988±0.006	1.013±0.021
GRAN	2	61±13	0.967±0.013	0.989±0.036
GraphRNN-RNN	4	8989±1076	0.017±0.006	0.058±0.005
GraphRNN-S	3	8062±925	0.535±0.074	0.065±0.021

Table 3: Graph manifolds metrics of Gr dataset on 10 independently trained GIN (\overline{F} ull dataset), using rGIN and constant features. While the metrics were able to keep consistent results through GIN training repetition, they kept inconsistent ranking results in regards to the base hypothesis, ranking the GraphRNN-S better than the GraphRNN-RNN generated grid graphs.

those metrics keep the low variance regarding experiment repetition, which includes pre-training and metrics computation. Second, in the **Grid experiment**, we compare Gr graphs from the real data distribution with ones sampled from GRAN and GraphRNN, which gives a closer outlook on using those metrics in an actual research workflow. We expect the metrics based on graph embeddings to rank these graph sets consistently with our visual inspection.

Experiment Parameters In both experiments, we varied training training hyperparameters of the graph classifier: *1st/* Node feature vector initialization for pre-training and evaluation. We compare three feature initializations: One-hot degree encoding, Constant node encoding, and GIN-random [17], where each node feature is generated by sampling a random integer each time the model is called during training and inference time. *2nd/* The graph dataset used for pre-training. We concatenated the graph data to work with three graph classification datasets: (1) The **base dataset** \overline{B} consists of the graph datasets BA, WS, C2L, C2S and Gr. (2) The **ladder dataset** \overline{L} takes the base dataset and add Ld graphs set. (3) The **full dataset** \overline{F} use the ladder dataset and add ER graph sets for each already existing graph set. *3rd/* The GIN graph embedding function, whether summation or concatenation.

Experimental Setup We follow [4] by using GIN [26] as our primary graph classifier. The official implementation of GIN by the authors of the paper is [available on github](#), using the GIN-0 version. We use the recommended hyperparameters, i.e., five layers of GIN with two layers of MLP for each, a hidden size of 64, and summation as the graph- and node-pooling method. For the pre-training of the models, we set the number of epochs to 64, and we used the Adam optimizer with a learning rate of 0.01. Then, we train our model using an 80-20 train-test

		WS				BA			
		Deg	Clust	4-orbits	Spec	Deg	Clust	4-orbits	Spec
Re-sampled		2.718e-6	0.0013	9.780e-6	0.0003	3.822e-5	0.0035	0.0047	0.0001
E-R		0.3412	1.3495	0.1904	0.1436	0.1074	0.6967	0.6889	0.0564
GRAN	BFS	0.0038	0.4425	0.0468	0.0282	0.0199	0.0534	0.0360	0.0023
	DFS	0.0054	0.1818	0.0043	0.0035	0.0339	0.0751	0.0436	0.0012
	Degree Descent	0.0180	0.5491	0.0511	0.0090	0.0706	0.1045	0.0553	0.0015
	Kcore	0.0208	0.4999	0.0232	0.0091	0.0587	0.0992	0.0379	0.0018
	Default	0.0001	0.0752	0.0046	0.0023	0.1017	0.0967	0.0367	0.0031
GraphRNN	BFS	0.0986	0.6891	0.1641	0.0250	0.1083	0.3152	0.1525	0.0184
	BFS-Max	0.0873	0.7388	0.2242	0.0277	0.1280	0.3378	0.1377	0.0150
	DFS	0.0780	0.7094	0.1508	0.0203	0.0342	0.3234	0.1690	0.0090
	Uniform	0.1801	1.1016	1.7263	0.0711	0.0937	0.3775	0.1531	0.0105

Table 4: Ablation study on Node Ordering of GRAN \blacklozenge and GraphRNN \clubsuit on BA and WS graphs, using reported MMD values. Lower values yields better generation quality. Overall GRAN gets lower MMD metrics values on both WS and BA datasets, while the choice of node ordering clearly influence results. On GraphRNN, DFS ordering give comparable results to the vanilla BFS orderings.

split on all our datasets. We use the [PRDC library](#) by [15] and the Frechet distance implementation from pytorch-fid [18]. For the perturbation experiment, we use the absolute Spearman rank correlation coefficient (see [29]) to assess the ranking ability of the given metric, with good ranking ability evaluated with an absolute value close to one and lousy ranking ability evaluated close to zero.

Perturbation experiment In Table 7, we observed that using a more extensive and more diverse training dataset impacts the behavior of FD positively and significantly, reducing the variance and improving the ranking ability. The F1-Score metrics exhibit more stability regarding the implementation except on the BA dataset and lower absolute Spearman rank correlation. We get a lower value for the F1-Score as they show high sensitivity towards such random perturbation on graphs, meaning that from a certain perturbation threshold, usually around 10 or 20%, both F1-Score fall down to 0. In Table 8, we observe that on the perturbation of grid graphs, the three given node feature initialization gave the same ranking coefficient, also with perfect stability through independent runs. Now we review an experiment closer to a real graph generation model benchmark.

Grid experiment In Table 9, we first observe that all metrics, computed using the degree node feature initialization, are consistently ranking the resampled grid graphs first when using the node, followed by GRAN generated grid graphs and the grid graph generated by both GraphRNN variants, which is consistent with the base assumption (see Figure 2). We also observe how The F1-Score metrics (PR and DC) show way less variance than the FD through independent runs. FD variance between independent runs is affected by the training dataset configuration, with the full dataset heavily reducing the variance for the resampled and GRAN-generated grid graphs. F1-Score is less affected by the training dataset configuration. The choice in the embedding function inside GIN

		C2L				C2S			
		Deg	Clust	4-orbits	Spec	Deg	Clust	4-orbits	Spec
Re-sampled		0.0004	0.0036	0.0039	0.0001	0.0005	0.0037	0.0006	0.0011
E-R		0.1481	0.2562	0.1042	0.0452	0.1715	0.1603	0.2597	0.0730
GRAN	BFS	0.0052	0.0189	0.0070	0.0017	0.0009	0.0441	0.0074	0.0223
	DFS	0.0012	0.0190	0.0057	0.0007	0.0022	0.0374	0.0087	0.0108
	Degree Descent	0.0262	0.0236	0.0073	0.0029	0.0038	0.0417	0.0138	0.0215
	Kcore	0.0082	0.0215	0.0070	0.0011	0.0038	0.0483	0.0405	0.0263
	Default	0.0042	0.0115	0.0054	0.0010	0.0080	0.0473	0.0801	0.0248
GraphRNN	BFS	0.0136	0.0400	0.0068	0.0092	0.0085	0.0486	0.0088	0.0411
	BFS-Max	0.0075	0.0346	0.0075	0.0050	0.0041	0.0472	0.0103	0.0400
	DFS	0.0171	0.0524	0.0083	0.0068	0.0027	0.0472	0.0051	0.0196
	Uniform	0.1594	0.2463	0.0103	0.0079	0.0431	0.0503	1.193	0.0478

Table 5: Ablation study on Node Ordering of GRAN and GraphRNN on C2L and C2S graphs, using reported MMD values. Lower values yields better generation quality. While on C3L graphs GRAN gets lower MMD values, GraphRNN gets similar results except for the Uniform ordering. On C2S graphs both model also gave similar results, with a slight advantage for GRAN. GraphRNN-DFS gave comparable results to the other GraphRNN node ordering variants.

(summation x_{Σ} or concatenation x_{\parallel}) does not change the behavior of all used metrics. In Table 3, we see that using other node feature initialization techniques, such as constant and random node features, gives inconsistent metrics results, with all tested graph embeddings-based metrics, in terms of ranking in the grid experiment.

Key Takeaways Like kernel-based measurement techniques, graph-embedding metrics behavior is highly sensitive to underlying hyperparameters [16]. More precisely, graph embeddings-based metrics that rely on trained graph classifiers are highly sensitive to more factors related to the training process, leading to reduced reproducibility and limited applicability of these metrics. Manifold-based metrics in graph embedding space can alleviate the sensitivity issue, making the selection of such metrics more interesting. Graph embeddings-based metrics use is more exciting in benchmarking graph generators compared to kernel-based metrics as they can capture global information of the graph. For instance, in the last section, two well-known graph generators were also compared using graph embeddings-based metrics.

4.2 Comparing GRAN and GraphRNN

We evaluate two state-of-the-art graph generation methods using the kernel-based MMD on distributions of graph invariants and previously explored metrics in graph embedding space. For both models, we used the official github source code at [JiaxuanYou/graph-generation](https://github.com/JiaxuanYou/graph-generation) and [lrjconan/GRAN](https://github.com/lrjconan/GRAN) with recommended hyperparameters.

	BA			WS			C2S			C2L		
	FD ↓	F1 PR ↑	F1 DC ↑	FD ↓	F1 PR ↑	F1 DC ↑	FD ↓	F1 PR ↑	F1 DC ↑	FD ↓	F1 PR ↑	F1 DC ↑
♥	13.89	0.9540	0.9567	4.450	0.9799	1.001	6.80	0.424	0.375	1487	0.9799	1.011
♦ BFS	2399	0.2132	0.0771	1332	0.4297	0.1344	16.29	0.8207	0.3466	47782	0.8881	0.6984
DFS	3323	0.1431	0.0315	1708	0.4546	0.2341	26.59	0.7951	0.3557	21005	0.9201	0.7523
DegDes	17275	0.0366	0.0065	2189	0.2535	0.0646	31.41	0.7335	0.2813	54148	0.8300	0.5670
Kcore	7057	0.0514	0.0101	1843	0.2535	0.0771	25.18	0.7415	0.2764	22293	0.7822	0.5559
Default	14364	0.0068	0.0011	308	0.6964	0.4212	74.07	0.6491	0.2679	45217	0.8819	0.7630
♣ BFS	20077	0.0	0.0	44503	0.0	0.0	44.69	0.2208	0.0549	12751	0.9588	0.7883
BFS-Max	30684	0.0	0.0	37641	0.0	0.0	61.37	0.2305	0.0552	5312	0.9678	0.8590
DFS	9762	0.0	0.0	22886	0.0	0.0	13.94	0.8060	0.3336	12946	0.9242	0.7411
Uniform	28248	0.0	0.0	72402	0.0	0.0	182.1	0.5762	0.1847	794711	0.1463	0.0976

Table 6: Graph embedding metrics for re-sampled graphs of a graph type are denoted in the first row with ♥. Reported pretrained Graph manifolds-based metrics on GRAN (♦) and GraphRNN (♣) for BA and WS dataset. Those graph manifolds metrics give similar observation with the evaluation using MMD-based metrics.

	Train Data	$\rho@FD$	$\rho@F1-PR$	$\rho@F1-DC$
Grid graphs	\overline{B} ase	0.72 ± 0.021	-0.5 ± 0.0	-0.5 ± 0.0
	\overline{F} ull	1.0 ± 0.0	-0.5 ± 0.0	-0.5 ± 0.0
BA graphs	\overline{B} ase	$0.99 \pm 1e-5$	-0.90 ± 0.001	-0.92 ± 0.002
	\overline{F} ull	1.00 ± 0.0	-0.79 ± 0.002	-0.81 ± 0.006
WS graphs	\overline{B} ase	0.87 ± 0.077	-0.67 ± 0.0	-0.67 ± 0.0
	\overline{F} ull	0.92 ± 0.015	-0.68 ± 0.001	-0.68 ± 0.001

Table 7: Ablation study on training dataset setup and Evaluation metrics behavior. Mean Spearman rank correlation and variance, collected for FD, F1-Score PR and F1-Score DC on 10 runs. The ranking correlation coefficient were computed by measuring 10 values corresponding to increasing perturbation of the BA, WS and Grid datasets.

Init	$\rho@FD$	$\rho@F1-PR$	$\rho@F1-DC$
Constant	1.0 ± 0.0	-0.5 ± 0.0	-0.5 ± 0.0
Random	1.0 ± 0.0	-0.5 ± 0.0	-0.5 ± 0.0
DOE	1.0 ± 0.0	-0.5 ± 0.0	-0.5 ± 0.0

Table 8: Ablation study on node feature initialization and Evaluation metrics behavior. Mean Spearman rank correlation and variance, collected for FD, F1-Score PR and F1-Score DC on 10 runs. The ranking correlation coefficient were computed by measuring 10 values corresponding to increasing perturbation of the Grid dataset.

Experiment Parameters In this experiment comparing GRAN and GraphRNN, we also varied the used node ordering policy used to reduce the complexity graph adjacency matrices space to learn from. For GRAN, we use BFS and DFS that is computed using the node with the highest degree, and other node ordering functions such as the default node ordering, degree-descent, and their denoted K-core ordering [14]. For GraphRNN, we use the vanilla BFS version, the uniform random ordering, our introduced DFS ordering, and a BFS variant where we do not reduce the graph matrix representation.

Experimental Setup For GraphRNN, we use 4-layer GRU cells with a 128-dimensional hidden state to encode the graph information and a 16-dimensional hidden state to encode the edge information. The output of the edge-level RNN is mapped from 16 to 8 dimensions using an MLP, then passed through a ReLU to be mapped to a scalar to pass through a sigmoid activation function finally.

The model is trained on each dataset in a minibatch of size 32 using the Adam optimizer for 3000 epochs each, with a learning rate of 0.001 reduced by 0.3 at the 400th and 1000th epoch. For GRAN, we keep the same hyperparameters for the model, using GNN with 7 layers and the block size and stride fixed at 1 because this will assure the highest generation quality possible. The model also uses an Adam optimizer for the training phase and is trained with a learning rate of 0.0001 and no decay. We train both models on the **BA**, **WS**, **C2L**, and **C2S** graph datasets using several node ordering, using 80-20 train-test splits.

We compared GRAN and GraphRNN evaluation results and also added two reference measurements for each dataset in order to give a sense of “scale”, as recommended by O’Bray et al. [16]. First, the evaluation compares the reference set of graphs with a resampled set of graphs from the same generator denoted with ♥ or "Resampled," a very similar set of graphs. Furthermore, we use the **ER** graph model as a benchmarking reference [14], computing metrics for a random graph generative model compared to the test graph sets. We resample an instance of BA, WS, C2L, and C2S graph datasets using their base generation function, which will be compared to the generated graph datasets. We train GIN on the full dataset configuration with one-hot node degree feature initialization and concatenation graph embedding function for Graph-Embeddings-based measurements. We then use the BA, WS, C2L, and C2S sets from the training dataset, which will be compared to generated graph sets.

Results Measurements based on kernel computation and those from GNN training give consistent comparisons between GraphRNN and GRAN (compare 5 and 6). Through our experiments and measurements, we find that generally, GRAN generates graph lists that are more faithful to the base dataset, especially on medium-sized datasets like BA or WS in table 4. However, GraphRNN shines when trained on small graph datasets like on C2S (compare 5). Across several tested node orderings for both GRAN and GraphRNN, we found discrepancies in generation quality, especially on GRAN. Our proposed DFS ordering for GraphRNN offers good performance, competing with the BFS ordering on our datasets, and even offers better performances on small datasets.

5 Conclusion & Future Work

This work compares GraphRNN and GRAN on different node orderings. We notice cases where GRAN wins, especially on medium-sized graphs, while GraphRNN has an advantage on smaller graphs. The node ordering influences these models’ performance, especially the generated graphs’ quality. Through extensive experiments, we have traced several insights into using GNNs to evaluate graph generators in expressiveness and stability. The first insight is with changing pre-training datasets, where we show that augmenting pre-training datasets with Erdős-Rényi-graphs can improve the performance of the GNN for graph generator evaluation. Also, we recommend using one-hot degree vectors to initialize the features of the nodes, which have shown the best results in terms of expressiveness and stability of the metrics. Finally, we find that using manifold-based

metrics also reduces the influence of other hyperparameters on the evaluation expressiveness and stability, in line with an observation made by [23], thus giving our recommendation to favor these metrics as reference metrics for future studies of GGMs. For future work, this work raises the interest in looking at other node orderings for using GraphRNN on small graphs. Elaborate feature engineering approaches for graph nodes could improve the behavior of graph-embedding-based metrics.

References

1. Abboud, R., İsmail İlkan Ceylan, Grohe, M., Lukaszewicz, T.: The surprising power of graph neural networks with random node initialization (2021)
2. Barabasi, A.L., Albert, R.: Emergence of scaling in random networks. *Science* **286**(5439), 509–512 (1999). <https://doi.org/10.1126/science.286.5439.509>, <https://science.sciencemag.org/content/286/5439/509>
3. Bojchevski, A., Shchur, O., Zügner, D., Günnemann, S.: Netgan: Generating graphs via random walks (2018)
4. Chia-Cheng Liu, H.C., Luk, K.: Auto-regressive graph generation modeling with improved evaluation methods (2019)
5. Cui, H., Lu, Z., Li, P., Yang, C.: On positional and structural node features for graph neural networks on non-attributed graphs (2021)
6. Du, Y., Wang, S., Guo, X., Cao, H., Hu, S., Jiang, J., Varala, A., Angirekula, A., Zhao, L.: Graphgt: Machine learning datasets for graph generation and transformation. In: *NeurIPS 2021* (2021)
7. Errica, F., Podda, M., Bacciu, D., Micheli, A.: A fair comparison of graph neural networks for graph classification (2020)
8. Goyal, N., Jain, H.V., Ranu, S.: Graphgen: a scalable approach to domain-agnostic labeled graph generation. In: *Proceedings of The Web Conference 2020*. pp. 1253–1263 (2020)
9. Hagberg, A., Swart, P., S Chult, D.: Exploring network structure, dynamics, and function using networkx. Tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States) (2008)
10. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium (2018)
11. Kipf, T.N., Welling, M.: Variational graph auto-encoders (2016)
12. Kynkäänniemi, T., Karras, T., Laine, S., Lehtinen, J., Aila, T.: Improved precision and recall metric for assessing generative models (2019)
13. Li, Y., Vinyals, O., Dyer, C., Pascanu, R., Battaglia, P.: Learning deep generative models of graphs (2018)
14. Liao, R., Li, Y., Song, Y., Wang, S., Nash, C., Hamilton, W.L., Duvenaud, D., Urtasun, R., Zemel, R.S.: Efficient graph generation with graph recurrent attention networks. *CoRR* **abs/1910.00760** (2019), <http://arxiv.org/abs/1910.00760>
15. Naeem, M.F., Oh, S.J., Uh, Y., Choi, Y., Yoo, J.: Reliable fidelity and diversity metrics for generative models (2020)
16. O’Bray, L., Horn, M., Rieck, B., Borgwardt, K.: Evaluation metrics for graph generative models: Problems, pitfalls, and practical solutions (2021)
17. Sato, R., Yamada, M., Kashima, H.: Random features strengthen graph neural networks (2021)

18. Seitzer, M.: pytorch-fid: FID Score for PyTorch. <https://github.com/mseitzer/pytorch-fid> (August 2020), version 0.2.1
19. Simonovsky, M., Komodakis, N.: Graphvae: Towards generation of small graphs using variational autoencoders (2018)
20. Stier, J., Granitzer, M.: Deepgg: A deep graph generator. In: International Symposium on Intelligent Data Analysis. pp. 313–324. Springer (2021)
21. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. CoRR **abs/1512.00567** (2015), <http://arxiv.org/abs/1512.00567>
22. Thompson, R., Ghalebi, E., Devries, T., Taylor, G.W.: Building lego using deep generative models of graphs. ArXiv **abs/2012.11543** (2020)
23. Thompson, R., Knyazev, B., Ghalebi, E., Kim, J., Taylor, G.W.: On evaluation metrics for graph generative models. In: International Conference on Learning Representations (2022), <https://openreview.net/forum?id=EnwCZixjSh>
24. Watts, D.J., Strogatz, S.H.: Collective dynamics of ‘small-world’ networks. *nature* **393**(6684), 440–442 (1998)
25. Weisfeiler, B., Lehman, A.: A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968 (1968)
26. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? (2019)
27. You, J., Ying, R., Ren, X., Hamilton, W.L., Leskovec, J.: Graphrnn: A deep generative model for graphs. CoRR **abs/1802.08773** (2018), <http://arxiv.org/abs/1802.08773>
28. Zeng, Z., Tung, A.K., Wang, J., Feng, J., Zhou, L.: Comparing stars: On approximating graph edit distance. *Proceedings of the VLDB Endowment* **2**(1), 25–36 (2009)
29. Zwillinger, D., Kokoska, S.: CRC standard probability and statistics tables and formulae sect.14.7. Chapman & Hall/CRC, Boca Raton (2000)

A Appendix

Used node ordering for autoregressive models For both GraphRNN and GRAN, we trained those models independently using several node ordering which are describe as follows :

- **BFS-Deg**: A BFS node ordering built by the BFS tree of each graph with the largest degree node as its root
- **DFS-Deg**: A DFS node ordering variant similar to BFS
- **Default**: The default node ordering used by NetworkX [9] graphs
- **Degree Descent**: An ordering based of the degree of nodes (in descent order)
- **K-core**: A novel node ordering introduced by GRAN, called k-core ordering, built from the k-core graph decomposition

We compare the GraphRNN method under four different versions:

- **BFS-Random**: A BFS node ordering built by the BFS tree of each graph where its root node is selected randomly, which is the default node ordering, which benefits from it by reducing the size of the graph adjacency matrix from $O(N^2)$ to $O(MN)$

- **DFS-Random** (Novel): We implemented a DFS node ordering variant similar to the default BFS one but without the benefits of using a reduced adjacency matrix, specific to BFS
- **BFS-Max**: Similar to BFS-Random, however, we used the complete graph adjacency matrix for DFS-Uniform. This will be useful to alleviate any bias linked to the choice of the hyperparameter M when comparing with DFS-Uniform
- **Default-Uniform**: The node ordering for each graph are completely shuffled at random

Base dataset configuration				
	Mean Rank ↓	FD ↓	F1 PR ↑	F1 DC ↑
Re-sampled	1	-0.0010±0.0007	1.0±0.0	0.994±0.003
GRAN	2	41004±14957	0.845±0.006	0.79±0.030
GraphRNN-RNN	3	91512±24215	0.040±0.010	0.019±0.009
GraphRNN-S	4	319669±94417	0.0±0.0	0.0±0.0
Ladder dataset configuration				
	Mean Rank ↓	FD ↓	F1 PR ↑	F1 DC ↑
Re-sampled	1	-0.0010±0.0008	1.0±0.0	0.992±0.005
GRAN	2	30773±23098	0.870±0.041	0.81±0.10
GraphRNN-RNN	3	127572±29497	0.042±0.015	0.02±0.01
GraphRNN-S	4	362265±149052	0.001±0.006	0.003±0.001
Full dataset configuration				
	Mean Rank ↓	FD ↓	F1 PR ↑	F1 DC ↑
Re-sampled	1	-0.0002±0.0001	1.0±0.0	0.990±0.006
GRAN	2	737±340	0.880±0.016	0.847±0.039
GraphRNN-RNN	3	45270±16467	0.046±0.006	0.027±0.007
GraphRNN-S	4	75501±29895	0.0±0.0	0.0±0.0

Table 9: Sum Graph Manifolds metrics on Gr dataset from 10 independently trained GIN. All of the metrics show consistent ranking fidelity with the base ranking hypothesis and stability. F1 Score metrics are more stable in regards to the dataset configuration, with low variance compared to FD.

Additional experiments results

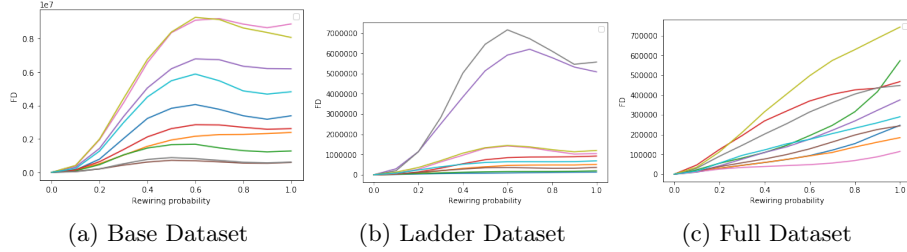


Fig. 3: Plots of Frechet Distance for perturbed Grid Graphs, over 10 independent trainings of GIN. Graph features are extracted with **concatenation**.

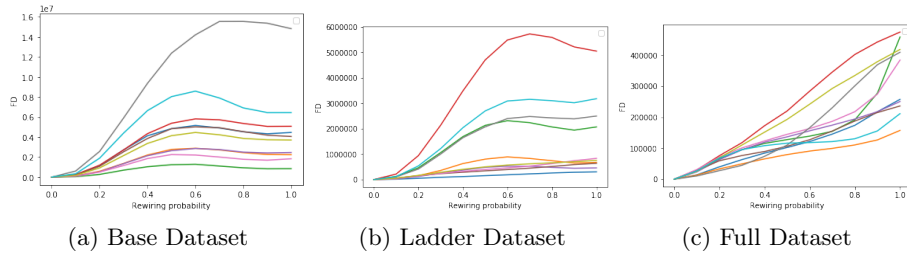


Fig. 4: Plots of Frechet Distance for perturbed Grid Graphs, over 10 independent trainings of GIN. Extracting graph features with **summation**.

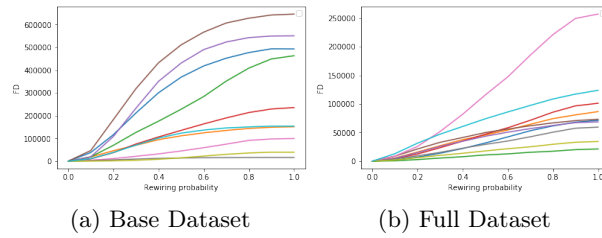


Fig. 5: Plots of FD for perturbed Barabasi-Albert Graph, over 10 independent trainings of GIN, extracting graph features with **concatenation**.

Base dataset configuration				
	Mean Rank ↓	FD ↓	F1 PR ↑	F1 DC ↑
Re-sampled	1	-0.001±0.001	1.0±0.0	0.994±0.003
GRAN	2	26639±14495	0.852±0.010	0.789±0.016
GraphRNN-RNN	3	65917±24978	0.042±0.005	0.026±0.006
GraphRNN-S	4	201977±89750	0.0±0.0	0.0±0.0
Ladder dataset configuration				
	Mean Rank ↓	FD ↓	F1 PR ↑	F1 DC ↑
Re-sampled	1	-0.002±0.002	1.0±0.0	0.990±0.003
GRAN	2	10228±11339	0.879±0.029	0.832±0.069
GraphRNN-RNN	3	91113±24230	0.047±0.005	0.034±0.011
GraphRNN-S	4	212305±78070	0.003±0.008	0.0006±0.001
Full dataset configuration				
	Mean Rank ↓	FD ↓	F1 PR ↑	F1 DC ↑
Re-sampled	1	-6e-4±-6e-4	1.0±0.0	0.990±0.007
GRAN	2	1264±360	0.885±0.0147	0.862±0.029
GraphRNN-RNN	3	59471±9750	0.048±0.001	0.033±0.003
GraphRNN-S	4	91310±19924	0.001±0.006	0.0003±0.001

Table 10: Concatenation Graph Manifolds metrics on Gr dataset from 10 independently trained GIN. All of the metrics show consistent ranking fidelity with the base ranking hypothesis and stability. F1 Score metrics are more stable in regards to the dataset configuration, with low variance compared to FD.

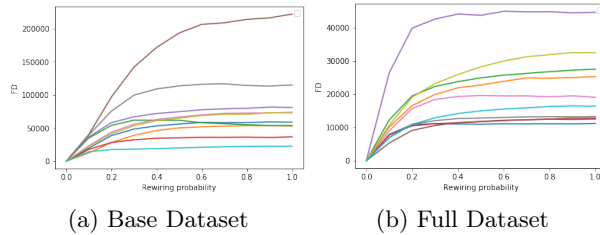


Fig. 6: Plots of FD for perturbed Watts-Strogatz Graph, over 10 independent trainings of GIN, extracting graph features with **concatenation**.