A Large Scale Synthetic Graph Dataset Generation Framework

Sajad Darabi [*]
sdarabi@nvidia.com
NVIDIA
USA

Piotr Bigaj [*]
pbigaj@nvidia.com
NVIDIA
Poland

Dawid Majchrowski
dmajchrowski@nvidia.com
NVIDIA
Poland

Artur Kasymov
akasymov@nvidia.com
NVIDIA
Poland

Pawel Morkisz
pmorkisz@nvidia.com
NVIDIA
USA

Alex Fit-Florea
afitflorea@nvidia.com
NVIDIA
USA

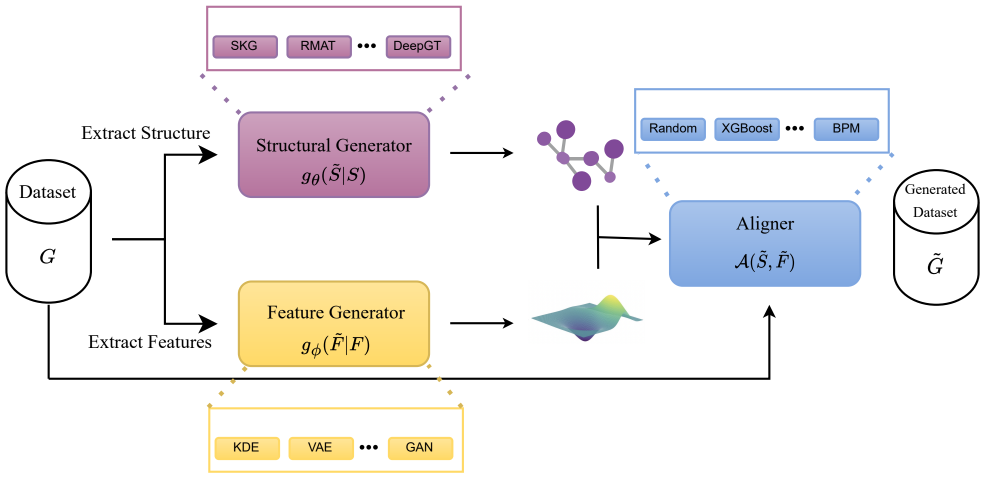


Figure 1: Overview of our proposed synthetic data generation framework. The module is composed of three parts: structural generator, which fits the graph structure, feature generator which fits the feature distribution contained in the graph, and finally an aligner, which aligns the generated features with the generated graph structure.

ABSTRACT

Recently there has been increasing interest in developing and deploying deep graph learning algorithms for many graph analysis tasks such as node and edge classification, link prediction, and clustering with numerous practical applications such as fraud detection, drug discovery, or recommender systems. Albeit there is a limited number of publicly available graph-structured datasets, most of which are tiny compared to production-sized applications with trillions of edges and billions of nodes or are limited in their application domain. In this work, we tackle this shortcoming by proposing a scalable synthetic graph generation tool. This tool can be used to learn a set of parametric models from proprietary datasets that can subsequently be released to researchers to study various graph methods on the synthetic data increasing prototype development and novel applications. Finally, the performance of the graph learning algorithms depends not only on the size but also on the graph datasets structure. We show how our framework generalizes across a set of datasets, mimicking both structural and feature distributions and the ability to scale them across varying dataset sizes. Code can be found on GitHub.[1]

[*]equal contribution
[1]https://github.com/NVIDIA/DeepLearningExamples/tree/master/Tools/DGLPyTorch/SyntheticGraphGeneration

1 INTRODUCTION

Graphs are ubiquitous data structures that capture relational and structural information between individual entities (nodes) via connections (edges) in many domains. For example, in social networks, a graph-based learning system leverages structural and feature information to make accurate user recommendations. Similarly, in an e-commerce platform, a transaction network can be used to detect fraudulent transactions. Real-world graphs are quite diverse, for a simple recommendation scenario consisting of user and item nodes, the user nodes would include information about age, gender, and income. Whereas the item nodes (e.g. a movie) would be characterized by the genre, length, and list of actors. Additionally, edge features may contain information about the rating the user gave a movie. Such attributed graphs are quite common, where the graph dataset's structure is enriched with features of the nodes and edges.

Graph Neural Networks (GNNs) have recently benefited from an increasing interest where a number of applications deal with data naturally represented as graphs. Motivated by similar developments in other domains, there has been efforts to extend the benefit of deep learning to this non-Euclidean domain enabling more streamlined approaches that leverage the relational data. Various methods have been developed to learn from graph data, such as

Node2Vec [12], graph convolution networks (GCN) [18] and graph attention networks (GAT) [33] focusing on various tasks including node classification [18], link prediction [23], graph clustering [14], where the term geometric deep learning [5] is used to refer to such methods.

A central problem in geometric deep learning is the need for real-world datasets. Most of the larger public datasets are similar and are often derived from academic citation networks [15], which are small for world-size problems. This lack of diversity limits the development of graph neural networks (GNN) and their evaluation. In this work, we propose a framework for synthetic graph generation which can systematically generate n-partite graphs with corresponding node or edge features in a scalable manner. Generating realistic large-size graph datasets, which we define graphs with billions to trillions of edges that simulate real-world datasets distributions will enable data sharing. This dataset curation will be a key component to advancing the field both for developing models that scale to such large graph sizes and from the perspective of improving the accuracy of developed GNNs, and new efficient geometric deep learning methods.

The proposed framework provides a parametric model that is flexible enough to fit a single graph as well as many graphs. It also supports generating both associated node and edge features. Our main contributions is as follows,

(1) We propose a flexible framework for synthetic graph dataset generation that can generate graph of arbitrary size based on original (usually smaller) graph characteristics containing both structure and tabular features.
(2) We show a set of case studies reflecting the generality and effectiveness of our approach on real-world datasets, simulating real-world graph statistical properties.

The paper is organized as follows: in the next sections we provide an over view of recent methods, then we define the problem and proposed method, subsequently the experimental setup to evaluate the method is presented, and finally a result section showcasing the usability of such framework on real-world datasets.

2 RELATED WORK

There has been increasing interest in curating datasets for different graph prediction tasks; for example, for node classification (CORA, PUBMED, and CITESEER) are commonly used [18], for link prediction (WN18, FB15k, OAG) [28, 29]. More recently, Open Graph Benchmark (OGB) datasets have been used for a set of challenging and realistic datasets to facilitate graph machine learning research and application development [16]. However, most of these sources are limited in scope as they are primarily derived from citation networks or social networks, limiting the scientific insight into various problems that derive similar graph data representations. As a result, synthetic graph generation has been proposed as a facilitator to this gap for investigating of different models in this domain.

The development of generative models for graphs poses unique challenges. These generative models are broadly categorized into two categories: traditional model-based methods and deep learning based methods. Simple, elegant, and general mathematical models are instrumental in graph generation. The two simplest random graph models are to select a graph uniformly at random from the set of graphs with v vertices and e edges or generate a graph uniformly random from the set of graphs and v vertices where each edge has the same probability of existing, commonly referred as the Erdos-Renyi models [9]. In biology, these models are accepted as a basic model to study biological networks where similarity of typologies and biological regulatory networks are studied or compared against [24]. Despite its usability and scalability, it violates power laws commonly found in social networks. A class of procedural generators tries to find simple mechanisms to generate graphs that match this property of graphs commonly found in the real-world. A typical representative here is the Barabasi-Albert Method [2, 3], which uses a preferential attachment idea by adding nodes and new nodes prefer to connect to existing nodes. R-MAT is an example of such a random graph generation model is the Chakrabarti et al. [6]. An alternative well-studied random graph model is the Stochastic Block Model [1] which generates graphs based on the communities within the graph and their degree distributions. Procedural synthetic graph generation is a foundation of the Stochastic Kronecker Graph method introduced in [20] which is a generalization of R-Mat for finding power-law degree distribution with the base different than two. The above methods are very attractive from a computational complexity standpoint, as well as modeling real-world graph properties.

On the other hand, recent deep-graph generators (DGG) that generate graphs sequentially either by generating node-by-node, edge-by-edge, or a block of nodes have been proposed. These generators are commonly autoregressive [22, 25, 36] and are limited in generating small graphs with 100's-1000's nodes. Further, very few deep graph generators work with single graph inputs, commonly referred to as one-shot graph generators, which are auto-encoder based [13, 19]. Additionally, most DGG methods lack the ability to generate features in addition to the structure, primarily because of the complexity of modeling such a problem jointly in an end-to-end fashion. A good overview of DGG can be found in [10]. Additionally, comparing two graphs' datasets in both structure and feature modality is challenging. Most methods [7, 10, 21] use plots to compare graphs in terms of connectivity, transitivity, path stats, and spectral properties, on the other hand common metrics like associativity, clustering coefficient, largest connected component are used to compare graph generation quality which are related to the graph structure [4, 22, 36]. In addition to these there is a need to develop methods for comparing large graphs from single input models that constitute both the structure as well as features, which our proposed framework supports.

3 PROPOSED METHOD

3.1 Problem Formulation

We are interested in the problem of graph generation that aims to sample similar graphs as the original graph via a probabilistic model. Formally, a graph contains both structural information and features; as such we define the graph as a triple $G(S, F_{\mathcal{V}}, F_{\mathcal{E}})$, where $S = (\mathcal{V}, \mathcal{E})$, $\mathcal{V} = \{v_1, v_2, \cdots, v_{\mathcal{N}}\}$ is the set of $\mathcal{N}$ nodes (or vertices), $F_{\mathcal{V}} \in \Re^{\mathcal{N} \times d_{\mathcal{V}}}$ is the corresponding feature matrix associated with node features with dimension $d_{\mathcal{V}}$, $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ is the set of $\mathcal{M}$ edges, where e_{ij} is an edge that connects node v_i and $v_j \in \mathcal{V}$, and

$F_{\mathcal{E}} \in \Re^{\mathcal{M} \times d_{\mathcal{E}}}$ is the associated edge feature for each edge $e_{ij} \in \mathcal{E}$ and $d_{\mathcal{E}}$ is the dimension of each edge feature.

Given an input graph G with arbitrary number of nodes, edges, and feature sets we aim to learn the probabilistic model that generated this graph $p(G)$. New graphs are then sampled from this generative process $\tilde{G} \sim p_{model}(G)$. In a general setting a graph with $\mathcal{N}$ nodes can be represented by up to $\mathcal{N}!$ adjacency matrices A^{π}, corresponding to arbitrary node ordering, resulting in a high representation complexity especially for large graphs. Additionally, there are $2^{\mathcal{N}(\mathcal{N}-1)/2}$ such graphs in the undirected case. It is important for generative models to scale to large-scale (billions or more edges/nodes) graphs and to accommodate this complexity in the output space. Additionally, to simultaneously generate node features $F_{\mathcal{V}}$ and edge features $F_{\mathcal{E}}$ greatly increases this modeling complexity. To make the problem tractable we decompose the generative process into different components

Our proposed generative model consists of three components: structural generation, feature generation, and alignment of these components as depicted in Figure 1. As shown, we make the structural generation and feature generation independent, which are then brought together using an aligner $\mathcal{A}(S, F) \rightarrow \tilde{G}$ (the aligner is detailed in section 3.3). In the following sections we will detail each component within the framework. To simplify the notation we consider a single graph G as input to the generator. From this graph we extract the corresponding structural information S, and its associated feature sets $F_{\mathcal{V}}$, $F_{\mathcal{E}}$. Next we detail the structural generator g_{θ}.

3.2 Structure Generation

3.2.1 Motivation. As we are interested in scaling the generation tool to trillion edge graphs, we leverage model-based graph generators, where our proposed method can be seen as a generalized stochastic Kronecker matrix multiplication for structure generation. The graph structure consists of the corresponding nodes and edges without node or feature attributes, i.e. $S = (\mathcal{V}, \mathcal{E})$. The adjacency matrix A corresponding to this graph is an $n \times m$ matrix, where $\mathcal{N} = n + m$, with entries $a(i, j) = 1$ if the edge e_{ij} between node i and node j exists.

We will first formalize the problem of generating the structure for a simple graph G that is non-directed heterogeneous in nodes and homogeneous in edges. Later we will extend this to general graph to show that our generator is a generalisation of R-MAT [7].

3.2.2 Problem formulation. The objective is to generate a graph $\hat{G}$ represented by the adjacency matrix $\hat{A}$ that is similar to the original graph adjacency matrix A. The $\hat{A}$ adjacency is generated by sampling E edges from a distribution θ. Here θ is a discrete 2D probability distribution, where $\theta_{i,j}$ represents the probability a directed edge from node i to node j exists in graph $\hat{G}$. If we choose to generate the graph of the same size as the input, that is $\hat{A}$ is $n \times m$ and graph $\hat{G}$ has $n + m$ nodes, then $\hat{A} \sim \theta$ which is generated as follows

$$\theta = \theta_S^{\otimes min(m,n)} \otimes \theta_H^{\otimes min(0,n-m)} \otimes \theta_V^{\otimes min(0,m-n)} \tag{1}$$

where

$$\theta_S = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \theta_H = \begin{bmatrix} q & 1-q \end{bmatrix} \theta_V = \begin{bmatrix} p \\ 1-p \end{bmatrix}, \tag{2}$$

$$m = \lceil \log_2 M \rceil, \quad n = \lceil \log_2 N \rceil, \tag{3}$$

$$p = a + b, \quad q = a + c, \tag{4}$$

$\otimes$ is a Kronecker matrix product, and $A^{\otimes b} = \underbrace{A \otimes A \otimes ... \otimes A}_{btimes}$ is the matrix Kronecker power.

Here θ_H and θ_V are marginals of θ_S which depend only on the shape of matrix $\hat{A}$ and only one of them is effectively used to form the probability distribution θ. If matrix $\hat{A}$ is square ($n = m$) then

$$\theta = \theta_S^{\otimes N}$$

which effectively is an R-MAT algorithm. The proposed approach differs in comparison to previous methods as $\hat{A}$ might be a non-square adjacency matrix, additionally $\hat{A}$ in general is constructed in a way that the i-th row and the i-th column can represent different nodes in the graph, giving us the ability to construct graphs that are heterogeneous in nodes in K-partite graphs.

For K-partite graph the obtained adjacency matrix is a block matrix with connectivity between the nodes in the corresponding partite. In this framework in order to represent K-partite graphs with $\hat{A}$ where matrix coordinates imply node ids then, it is sufficient to consider $\hat{A_P}$ for each partite P and create $\hat{A}$ out of $\hat{A_P}$ for each partite.

3.2.3 Fitting the generator. The purpose of the structure generator is to generate a graph $\hat{G}$ that is similar in graph characteristics to original graph G. These characteristics will depend on the parameters of θ_S. We use the notion of (normalized) degree distributions of the graphs G, and $\hat{G}$ as a similarity measure therefore in order to find parameters of θ_S we need to minimize the following error

$$J(\theta_S) \propto \sum_{k^{in}=0}^{k^{in}_{max}} (c_k^{in} - \hat{c_k^{in}})^2 + \sum_{k^{out}=0}^{k^{out}_{max}} (c_k^{out} - \hat{c_k^{out}})^2, \tag{5}$$

where k is the node degree, c_k^{in} is the number of nodes in graph G having in-degree k, c_k^{out} is the number of nodes in graph G having out-degree k, $\hat{c_k^{in}}$ is the estimated number of nodes in graph $\hat{G}$ having in-degree k, and $\hat{c_k^{out}}$ is the estimated number of nodes in graph $\hat{G}$ having out-degree k.

Note that this does not imply that A and $\hat{A}$ are similar in a sense $A - \hat{A} \approx 0$, which would require appropriate node permutation in $\hat{G}$. Simple degree distributions comparison allows us to effectively compare graphs only when the number of nodes is equivalent, though our goal is to generate a graph $\hat{G}$ that is arbitrarily larger in number of nodes than G. Therefore, we require a θ_S for the graph of the same size and then generate

$\hat{c_k^{in}}$ and $\hat{c_k^{out}}$ depend on θ_S as follows

$$\hat{c_k^{out}} = \binom{E}{k} \sum_{i=0}^{m} \binom{m}{i} [p^{m-i}(1-p)^i]^k [1 - (p^{m-i}(1-p)^i)]^{E-k} \tag{6}$$

and

$$\hat{c_k^{in}} = \binom{E}{k} \sum_{i=0}^{n} \binom{n}{i} [q^{n-i}(1-q)^i]^k [1 - (q^{n-i}(1-q)^i)]^{E-k}, \quad (7)$$

where $\hat{c_k^{in}}$ and $\hat{c_k^{out}}$ depend only on p and q. Solving (5) for $p = a+b$ and $q = a+c$ leads to underdetermined system $(a+b+d+c = 1)$ as we have 3 equations and 4 variables. [7] proposes to use $\frac{a}{b} = \frac{a}{c} = \frac{3}{1}$, since this can be seen in many real world scenarios. We have encountered graph datasets that did not follow this ratio. Instead of this ratio we propose to estimate $\frac{a}{b}$ and $\frac{a}{c}$ from adjacency matrix A of the original graph G by Maximum Likelihood Estimation of these parameters as this yields better results.

3.2.4 Chunked generation. The R-MAT algorithm operates by recursively subdividing the adjacency matrix A. The i-th subdivision corresponds to the single matrix $\theta_i \in \Re^{2\times 2}$ and may be interpreted as choosing a bit for the source and destination node of the sampled edge. For the case of generating large graphs, producing a graph $\hat{G}$ by sampling E edges from the distribution θ may not fit into the available memory. In order to parallelize generation and decrease memory consumption, θ is represented as $\theta_{pref} \otimes \theta_{gen}$, where θ_{pref} is used to generate a unique chunk prefix to avoid id-overlap and both terms have the form $\theta_S^{\otimes n_x}$ as in eqn. 1. As each edge is sampled independently, we can replace sampling prefixes from θ_{pref} by the expected value of the edges for the given prefix $E_{pref} = E \cdot \mathbb{E}[\theta_{pref}]$. To this end, to produce the i-th chunk we sample E_{pref}^i edges from θ_{gen} and prepend the i-th prefix to them. The prefixes guarantee us that there will be no edge overlap between chunks and the final graph is simply constructed by concatenating them to obtain the graph $\hat{G}$.

3.3 Feature Generation

Next we consider the feature sets associated with the graph $\mathcal{D}_{\text{features}} = F_{\mathcal{V}/\mathcal{E}}$, where each row $x_i \in \Re^{d_{\mathcal{V}/\mathcal{E}}}$ is an observation sampled from a data-generating distribution $P_{\mathcal{D}_{\text{features}}}(x)$. We treat this dataset as a tabular dataset. Each row corresponds to the corresponding edge features, source node features, and destination nodes features. The objective is to learn a generative model G_{features} over this data generating process. To this end, we consider the multi-modal setting where x_i is a concatenation of discrete $\mathcal{D} = [D_1, \cdots, D_{|\mathcal{D}|}]$ and continuous features $C = [C_1, \cdots, C_{|C|}]$. Without loss of generality our generator follows a GAN architecture, though any high-capacity method that can model the underlying distribution can be used. To this end, our input layer consists of a feature tokenizer where the corresponding embedding for each feature is computed as follows:

$$E_j = b_j + f_j(x_j) \in \Re^{d_j} \quad f_j : \mathbb{X} \to \Re^{d_j}$$

where b_j is a bias term for the j-th feature and f_j is the corresponding feature tokenization function. Our model applies mode-specific normalization to continuous columns which 1) fits a variational mixture of Gaussian to continuous columns of C 2) converts the continuous elements of c into a one-hot vector denoting the specific Gaussian that best matches the element and its scalar normalized value within the selected Gaussian as done in [35]. For discrete columns we introduce embedding layers $W_{D_i} \in \Re^{|D_i| \times d_{D_i}}$, where $|D_i|$ is the number of possible unique discrete values and d_{D_i} is the dimension of the embedding. The input layer operation could be summarized as follows

$$\begin{aligned}
E_j^{cont} &= f_j^{cont}(C_j) && \in \Re^{d_{C_j}} \\
E_j^{cat} &= b_j^{cat} + e(D_j)^T W_j^{cat} && \in \Re^{d_{D_j}} \\
\bar{X} &= concat[E_1^{cont}, \cdots, E_{C_C}^{cont}, E_1^{cat}, \cdots, E_{D_\mathcal{D}}^{cat}] && \in \Re^{d_{\bar{X}}},
\end{aligned}$$

where f_j^{cont} is a single layer fully-connected network, $e(\cdot)$ converts the input into a one hot vector and the tokenized input $\bar{X}$ has a dimension $d_{\bar{X}} = \sum_j^C d_{C_j} + \sum_j^{\mathcal{D}} d_{D_j}$. In GAN training there are two separate models the generator $\mathcal{G}$ and discriminator D that estimates the probability of whether the sample came from the fake or real distribution. The generator $\mathcal{G} : \Re^{dim(z)} \to \Re^{dim(x)}$ takes the input $z \sim p(z) \in \Re^{d_z}$ and recovers samples in the original data space $\tilde{x}$. The discriminator then distinguishes between $D(x, \tilde{x}) \to [0, 1]$. Both networks are high capacity deep neural networks that follow a stack of $f(x) = \theta(\texttt{ResNetBlock}(\cdots(\texttt{ResNetBlock}(\texttt{FC}(x)))))$ where $\texttt{ResNetBlock}(x) = x + \texttt{Dropout}(\texttt{ReLU}(\texttt{FC}(\texttt{BatchNorm}(x))))$, `FC` is a fully-connected network which takes the $dim(x)$ size of its input. $\mathcal{G}$ and D are both trained together using the GAN objective

$$\begin{aligned}
\min_{\mathcal{G}} \max_{D} l(\mathcal{G}, D) = \min_{\mathcal{G}} \max_{D} & \mathbb{E}_{x \sim p_{data}(x)}[log(D(x)] \\
& + \mathbb{E}_{z \sim p(z)}[log(1 - D(\mathcal{G}(z)))].
\end{aligned}$$

The trained feature generator is then used to sample from the learned feature distribution $\tilde{x} \sim P_{\mathcal{D}_{features}}$.

3.4 Graph & Feature Aligner

Once the structural and feature generators are trained the final graph is created via an aligner. The aligner is a function that maps the generated set of features onto the generated graph structure $\mathcal{A}(S, F) \to G(\mathcal{V}, \mathcal{E}, F_{\mathcal{V}}, F_{\mathcal{E}})$. A trivial aligner, could randomly assign features to the corresponding nodes and edges of the generated graph. Instead, we propose to train a function R that matches the generated structure with the generated features, preserving some properties of the input graph G. For example, for a recommender system use case, we may have an advertisement that is clicked by the majority of population and we want to preserve its features properties.

Given the real graph G_{real} we extract a set of features from the corresponding structural information $F_S : V \to \Re^{d_S}$. These features correspond purely with the graph structure, such as node degree, node centrality, clustering coefficient, and page rank. Subsequently, a predictor R is trained to capture the correlation between the real graph structural features and the corresponding real feature set. For an edge $e_{(src,dst)}$ the predictor $R : \Re^{d_S} \times \Re^{d_S} \to \Re^{d_\mathcal{E}}$ maps it on to the feature $x = R(F_S(v_{src}), F_S(v_{dst}))$, where src is the source node index, and dst is the destination node index. We propose to choose `XGBoost` [8] as our predictor R, for each feature x_j. The proposed aligner can be summarized as follows

$$R(F_S(\mathcal{E}_{src}), F_S(\mathcal{E}_{dst})) = \texttt{stack}[\texttt{XGBoost}_1, \cdots, \texttt{XGBoost}_k] \in \Re^{d_{\mathcal{E}}}$$
$$\text{Rank} = \max_{i \in M}(\texttt{sim}(R(F_S(\mathcal{E}_{src}), F_S(\mathcal{E}_{dst})), X_i),$$

where $\mathcal{E}_{src}$ and $\mathcal{E}_{dst}$ are the set of source and destination vertices, respectively.

The series of XGBoost models are trained to infer the features from structural information, in the case of $e_{src,dst}$ both features of node $F_S(v_{src})$ and $F_S(v_{dst})$ are used as input to the model. A similarity score between the predicted vector and corresponding generated feature is used to rank the features that are assigned to the edges of the graph. For continuous values the mean squared-error is used

$$-\sum_{j \in \{C\}} (R(F_S(v_{src}), F_S(v_{dst}))^{(j)} - x_i^{(j)})^2$$

and similarly for categorical columns the cosine similarity

$$\frac{\sum_{j \in \mathcal{D}} (R(F_S(v_{src}), F_S(v_{dst}))^{(j)} X_i^j}{\sqrt{\sum_{jin \mathcal{D}} (R(F_S(v_{src}), F_S(v_{dst}))^{(j)}} \sqrt{\sum_{j \in \mathcal{D} x^{(j)}}}}.$$

Once the generated set of features are ranked, they are overlaid on the original graph structure where the node and edge information is appended to the corresponding set of features.

4 EXPERIMENTS

In this section, we introduce a set of experiments to show case the effectiveness of the proposed framework for generating real-world graphs. We describe a set of metrics to evaluate data quality across structural, feature, and both.

4.1 Methods

Our method involves fitting a single large graph and learning a parametric model that can be used to generate graphs on the same scale or larger. We compare with the following baselines:

- **Random**: We generate graph structures using Erdos-Renyi model, along with a random feature generator with ranges fitted to the original feature dimension. This model is integrated into our proposed framework.
- **GraphWorld** [26]: Is a recent method for generating arbitrary graphs using the degree corrected stochastic block model (SBM). **Note:** we improve this method and add a fitting step that fits the model onto the underlying dataset.

4.2 Dataset details

The real-world datasets used in the experiments encompassing different graph sizes with varying number of features which are summarized in Table 1. The steps used to construct the graph is detailed in Table 9 in the appendix.

4.3 Metrics & Evaluation

We use a set of metrics to assess the quality of the generated graph. These sets of metrics across different components of the generated graph are detailed below

Table 1: Dataset sizes used through out experiments.

ID	Dataset	# nodes	# edges	# features
1	Tabformer	106482	978288	5
2	IEEE-Fraud	17289	52008	48
3	Paysim	9075669	6362620	8
4	credit	1666	476414	283

Table 2: Comparison across different datasets and baseline models. ↑ denotes higher is better and ↓ denotes lower is better.

		Metric		
Dataset	**Method**	**Degree Dist. ↑**	**Feature Corr. ↑**	**Degree-Feat Dist-Dist ↓**
	random	0.8099	0.3931	0.8213
Tabformer	graphworld	0.2836	0.3609	0.8248
	ours	**0.9833**	**0.4141**	**0.8101**
	random	0.9620	0.2120	0.4335
IEEE-Fraud	graphworld	0.0010	0.4179	0.8272
	ours	**0.9865**	**0.5724**	**0.2359**
	random	0.0434	0.8370	0.6520
credit	graphworld	0.3556	0.8352	0.7584
	ours	**0.5178**	**0.8558**	**0.5516**
	random	0.6711	0.4833	0.5155
paysim	graphworld	0.6547	0.4115	0.3453
	ours	**0.9602**	**0.7500**	**0.2630**

- Degree Dist.: in networks the degree of a node is the number of connections it has with other nodes, and its the degree-distribution of the network is the distribution of these degrees over the whole network. For example, a graph has a power-law if the number of nodes N_d with degree d is given by $N_d \propto d^{-\alpha}$ where α is the power law exponent.
- Hop-plot: The diameter of a graph is D if every pair of nodes can be connected by a path of length at most D edges. As this metric is susceptible to outliers often a alternative metric called *effective diameter* is used, which is the minimum number of links in which a fraction of all pairs of nodes can be reached each other. A hop-plot extends the notion of diameter by plotting reachable pairs $d(h)$ within h hops.
- Feature Corr.: We consider correlation between columns of the features in the dataset. For correlation between continuous columns we use the standard Pearson correlation. Between continuous and categorical columns we consider the *correlation ratio* [11], and between categorical columns we consider using the Theil's U [32] as a metric, which measures the conditional entropy between two variables.
- Degree-Feat Dist-Dist: We consider the joint degree distribution and feature distribution as a measure of graph feature+structural similarity. This metric computes the JS-divergence between the joint distribution over the generated graph and real graph.

4.4 Results

In Table 2 we summarize the comparison of the proposed framework across different datasets presented in Table 1. From the table we can see that the synthetic data quality generated using our method outperforms the two baseline models. It is worth noting that we modified GraphWorld [26] to fit the underlying data. Additionally, this method can be integrated within the proposed framework where the structural generator is a SBM model, the feature generators are multi-variate Gaussian's and the aligner is a random aligner. We do not provide comparison with methods such as [22, 36] as 1) these method require multiple graphs and we are providing comparisons for single graph generation and 2) these methods do not scale beyond 1000s of nodes as previously mentioned.

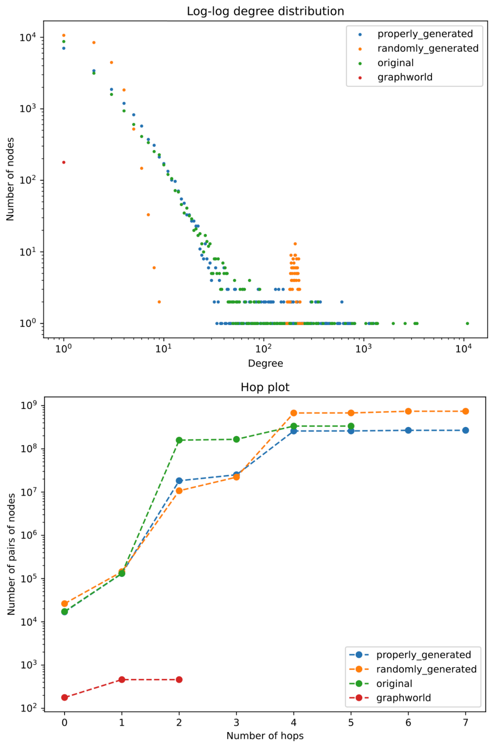


Figure 2: Comparison of degree distribution (top) and hop plot (bottom) across our proposed method (properly generated) and other baselines.

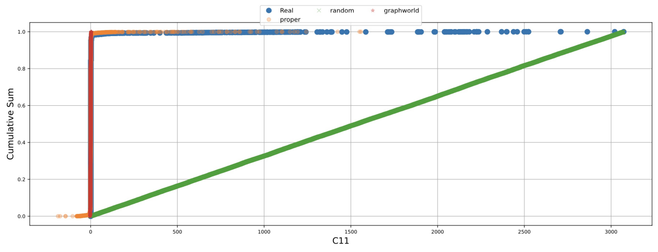


Figure 3: Cumulative distribution comparison on feature column C11 of IEEE-Fraud dataset.

We also qualitatively assess the quality of the generators compared to the baselines. from Figures 2, 3, we show visualization comparing across the original graph distribution, ours and baseline models, demonstrating the effectiveness of each component in the generation step. For example, in Figure 2 we see that the fitted generator best resembles the long tail degree distribution of the original dataset (typically observed in social networks), whereas graphworld and randomly generated fail to capture this scaling property. Similarily, the cumulative distribution generated using a proposed fitted GAN architecture best resembles the original feature distribution. Finally to qualitatively evaluate the aligned features with the graph structure, a plot depicting the degree-distribution versus feature-distribution is used to compare across the methods (Figure 5 in Appendix), where visually the heatmaps of the synthetic data should match the original graphs heatmap. This is the case for our proposed method.

Table 3: Random graph generation timings

nodes	100e6				
edges	100e9	250e9	500e9	750e9	1e12
time	22 min	46 min	103min	130 min	179 min

Table 4: Synthetic MAG240m [34] generation timings

scale	total nodes	total edges	time
1x	244e6	1.7e9	~12 min
2x	488e6	14e9	~23 min
4x	997e6	110e9	~113 min
8x	1953e6	885e9	~838 min

4.5 Big Graph Generation

We validated our approach to generating large graphs by producing big random graphs using Erdos-Renyi model and mimicking the structure of the biggest publicly available graph MAG240m [34]. For the experiments with the Erdos-Renyi model presented in Table 3, we freezed the number of nodes and increased the number of edges up to a trillion. In contrast to a simple random homogeneous graph, we scaled the heterogeneous MAG240m from the real world and summarized the results in Table 4. We scaled the number of nodes linearly and the number of edges cubically. All measurements were done on the same machine with 8 NVIDIA V100 16GB GPUs.

4.6 Downstream Tasks

Common downstream tasks in graphs datasets are node classification and edge classification. The proposed framework can generate both node-level and edge-level features, hence it supports these various tasks using the proposed generator. To this end, we train our synthetic data generator on both node classification and edge classification datasets, subsequently we generate a graph of the same size for pre-training. Finally, we fine-tune on the original real dataset. The results are presented in Table 5, where no-pretraining refers to the case where we simply train on the downstream dataset.

For node-classification task we use the Cora [30] as toy example which is a citation network dataset, with node labels as topics and features as multi-hot vectors; and for edge-classification task the ieee-fraud dataset, which contains edge features as well as labels denoting whether a particular transaction is fraudulent. For all

Table 5: Comparison of Pre-training followed by fine-tuning for node classification and edge classification tasks.

Dataset	Generator	Model	Accuracy ↑
Cora	random	GCN	0.7470
		GAT	0.7595
	ours	GCN	0.7677
		GAT	0.7720
	no-pretraining	GCN	0.76275
		GAT	0.7650
ieee-fraud	random	GCN	0.9788
		GAT	0.9793
	ours	GCN	0.9831
		GAT	0.9840
	no-pretraining	GCN	0.9823
		GAT	0.9830

datasets, we train the models for a maximum of 200 epochs (fine-tuning epochs + pre-training epochs) and use Adam [17] with a starting learning rate of 0.01, with early stopping after ten epochs of no improvement on a held-out validation set as defined by the original datasets. The networks are 2-layer GCN [18], GAT[33] models with hidden dimension set to 128.0. From this table, training on a randomly generated graph hinders downstream performance, whereas pre-training using a graph with similar characteristics as the original graph results in slight improvements.

4.7 Synthetic Graph Ablation Study

We experiment on a synthetic graph with pre-determined properties for structure & the feature set to determine when aligning graph structure with features is essential and cases where GNNs are helpful in the first place. Namely, a set of synthetic graphs are generated that contain high (or low) homophily and high (or low) signal-to-noise ratio (SNR) of this structure for the features of the graph. In total, four datasets are generated that are used for this experiment. By constructing such a graph, the downstream model would be able to discriminate using solely graph structure, feature, or both depending on the setting of the dataset low/high for either part. The synthetically generated graphs contain 1000 nodes with an edge density of 0.06 (24000 edges). We consider a high homophily/SNR and low homophily/SNR value as 0.85/1.5 and 0.15/0.5, respectively. Note that a h for homophily indicates that inter clusters are h times more likely to be connected than intra-clusters. Similarly, a signal-to-noise ratio of SNR indicates how correlated the feature sets are compared to the downstream label, where the intra-clusters have the same label. In this experiment, the downstream task is set to be node classification. We train a GNN model that leverages both the structure and features, specifically a GAT model, with the same configuration as in section 4.6. On the other hand, an XGBoost model is trained only on the graph features.

From the results in Figure. 4, we can observe the following: 1) there is an asymmetry in the informativeness of the graph connectivity and graph features for downstream prediction task, and that a noisy graph structure could hinder performance (e.g. comparing GAT and XGboost on the original dataset, we can see XGBoost out performs GAT if the structure is noisy). 2) fitting is required for the graph structure/features/and their alignment if both graph structure and feature is informative. Otherwise for cases where the graph may not follow particular connectivity pattern or structure it is not necessary to fit the graph and a randomly generated graph will do just as well.

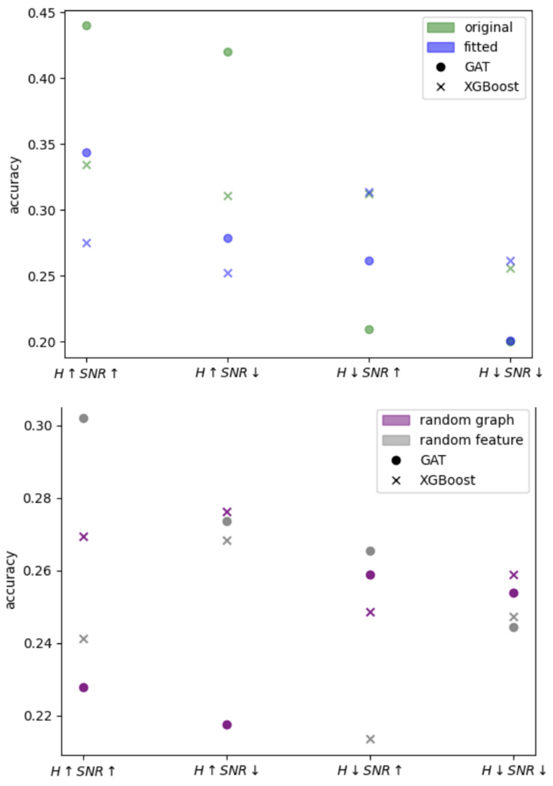


Figure 4: Comparison of training a GAT model on both structure/feature versus training a XGBoost model solely on features for different dataset settings corresponding to high/low homophily ($H \uparrow / \downarrow$) and high/low signal to noise ratio ($SNR \uparrow / \downarrow$). (Top) Depicts the performance on the original synthetic dataset, and fitted corresponding to fitting our proposed generator. (Bottom) shows the results by replacing the components of the generator with random counterparts.

4.8 Ablation Study

We conduct an ablation study by varying the components in our proposed framework. Specifically, we substitute the feature generator with one of {GAN, Kernel Density Estimator (KDE), Random}, the structural generator with one of {Ours, TrillionG [27], Random} and the aligner with either {XGBoost, Random} across our proposed components and evaluate the synthetic data quality. The results are presented in 6. This showcases the benefit of providing a fitting mechanism for each component.

4.9 GNN Performance Analysis

In this section, we run additional experiments analyzing GNN's throughput using our generated and randomly generated datasets. We time the throughput of the GNN network by sampling subgraphs

Table 6: Ablation study on IEEE Dataset

Struct. Generator	Feature Generator	Aligner	Metric Degree Dist. ↑	Feature Corr. ↑	Degree-Feat Dist-Dist ↓
Ours	**GAN**	xgboost	0.989123 (±0.0014)	0.566755 (±0.0121)	0.319410
		random	0.989123 (±0.0014)	0.566755 (±0.0121)	0.330570
	KDE	xgboost	0.989123 (±0.0014)	0.810647 (±0.0277)	0.198234
		random	0.989123 (±0.0014)	0.810647 (±0.0277)	0.512846
	Random	xgboost	0.989123 (±0.0014)	0.220692 (±0.0090)	0.355433
		random	0.989123 (±0.0014)	0.220692 (±0.0090)	0.364814
TrillionG [27]	**GAN**	xgboost	0.848164 (±0.0012)	0.566755 (±0.0121)	0.316364
		random	0.848164 (±0.0012)	0.566755 (±0.0121)	0.423613
	KDE	xgboost	0.848164 (±0.0012)	0.810647 (±0.0277)	0.261013
		random	0.848164 (±0.0012)	0.810647 (±0.0277)	0.360605
	Random	xgboost	0.848164 (±0.0012)	0.220692 (±0.0090)	0.434184
		random	0.848164 (±0.0012)	0.220692 (±0.0090)	0.484187
Random [9]	**GAN**	xgboost	0.962508 (±0.0011)	0.566755 (±0.0121)	0.420893
		random	0.962508 (±0.0011)	0.566755 (±0.0121)	0.480556
	KDE	xgboost	0.962508 (±0.0011)	0.810647 (±0.0277)	0.38334
		random	0.962508 (±0.0011)	0.810647 (±0.0277)	0.400145
	Random	xgboost	0.962508 (±0.0011)	0.220692 (±0.009081)	0.434184
		random	0.962508 (±0.0011)	0.220692 (±0.0090)	0.613355

Table 7: Comparison of throughput for GCN [18] across different datasets. ↑ denotes higher is better and ↓ denotes lower is better.

Dataset	Method	Metric Rel. Timing ↑	Timing
Tabformer	original	1.0	107.5836
	random	0.8611 ±0.8611	93.3228
	ours	**0.9376** ±0.0094	**100.1258**
ieee-fraud	original	1.0	1.9833
	random	0.7922 ±0.1484	2.06162
	ours	**0.8039** ±0.1450	**2.04135**
credit	original	1.0	28.6372
	random	0.9668 ±0.0003	26.9836
	ours	**0.9822**±0.0083	**28.1271**
paysim	original	1.0	153.4566
	random	0.9302 ±0.0025	143.1298
	ours	**0.9581** ±0.0020	**159.1917**

Table 8: Comparison of throughput for GAT[33] across different datasets. ↑ denotes higher is better and ↓ denotes lower is better.

Dataset	Method	Metric Rel. Timing ↑	Timing
Tabformer	original	1.0	191.6026
	random	0.6903 ±.0004	132.2720
	ours	**0.8710** ±0.0038	**166.8899**
ieee-fraud	original	1.0	3.9867
	random	0.6975 ±0.0795	3.41476
	ours	**0.8003** ±0.0940	**3.5655**
credit	original	1.0	70.2702
	random	0.9370 ±0.0011	65.8476
	ours	**0.9609**±0.0034	**67.8039**
paysim	original	1.0	235.1643
	random	0.9248 ±0.0016	217.4626
	ours	**0.9657** ±0.0046	**243.2182**

on the original graph using a Multi-Layer Neighborhood Sampler [2] and measuring the time of every epoch. The results are summarized in Table. 7. As can be seen from the results, generally, for datasets where there's a higher discrepancy in the metrics reported in Table 2 there is a larger gap in the timing between random vs original and ours. The relative timing (Rel. Timing) is calculated by subtracting the epoch times on the original dataset from the generated and normalizing, i.e. (Rel. Timing = $1.0 - \frac{|t_{generated} - t_{original}|}{t_{original}}$

5 CONCLUSION

The objective of this paper was to create a method of generating graph datasets that consist of (1) structure describing how nodes are connected to other nodes (2) along with tabular node and edge features contained in the graph. Using our method, these two components (structural and tabular) are aligned in the sense that the degree distributions of nodes match the probability distributions of features connected with nodes.We have shown the generality of such a framework that can more accurately replicate and scale with respect to different graph sizes and structure. We hightlight that generating large-scale graph datasets that consist of structure and tabular portion serves many real-world use cases starting such as data anonymization, as a tool to benchmark GNN models by profiling them on arbitrary-sized graph datasets, as well as potentially using synthetic datasets increasing the accuracy of GNN by using such data to pre-training the GNN model and fine-tune it on the original dataset. Future directions can include modifying the framework to these different ends.

[2]dgl.dataloading.neighbor.MultiLayerNeighborSampler

REFERENCES

[1] Emmanuel Abbe. 2017. Community detection and stochastic block models: recent developments. *The Journal of Machine Learning Research* 18, 1 (2017), 6446–6531.

[2] Réka Albert and Albert-László Barabási. 2000. Topology of Evolving Networks: Local Events and Universality. *Phys. Rev. Lett.* 85 (Dec 2000), 5234–5237. Issue 24. https://doi.org/10.1103/PhysRevLett.85.5234

[3] Réka Albert and Albert-László Barabási. 2002. Statistical mechanics of complex networks. *Reviews of modern physics* 74, 1 (2002), 47.

[4] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. 2018. NetGAN: Generating Graphs via Random Walks. (2018). https://doi.org/10.48550/ARXIV.1803.00816

[5] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 34, 4 (2017), 18–42.

[6] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. 2004. R-MAT: A recursive model for graph mining. In *Proceedings of the 2004 SIAM International Conference on Data Mining*. SIAM, 442–446.

[7] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. 2004. R-MAT: A recursive model for graph mining. *SIAM Proceedings Series* 6. https://doi.org/10.1137/1.9781611972740.43

[8] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[9] P. Erdös and A. Rényi. 1959. On Random Graphs I. *Publicationes Mathematicae Debrecen* 6 (1959), 290.

[10] Faezeh Faez, Yassaman Ommi, Mahdieh Soleymani Baghshah, and Hamid R. Rabiee. 2020. Deep Graph Generators: A Survey. *CoRR* abs/2012.15544 (2020). arXiv:2012.15544 https://arxiv.org/abs/2012.15544

[11] Ronald Aylmer Fisher. 1992. Statistical methods for research workers. In *Breakthroughs in statistics*. Springer, 66–70.

[12] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[13] Aditya Grover, Aaron Zweig, and Stefano Ermon. 2019. Graphite: Iterative generative modeling of graphs. In *International conference on machine learning*. PMLR, 2434–2444.

[14] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017).

[15] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. 2021. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430* (2021).

[16] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems* 33 (2020), 22118–22133.

[17] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[18] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[19] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).

[20] Jurij Leskovec, Deepayan Chakrabarti, Jon Kleinberg, and Christos Faloutsos. 2005. Realistic, Mathematically Tractable Graph Generation and Evolution, Using Kronecker Multiplication. In *Knowledge Discovery in Databases: PKDD 2005*, Alípio Mário Jorge, Luís Torgo, Pavel Brazdil, Rui Camacho, and João Gama (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 133–145.

[21] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. 2008. Kronecker Graphs: An Approach to Modeling Networks. *Journal of Machine Learning Research* 11 (12 2008). https://doi.org/10.1145/1756006.1756039

[22] Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Charlie Nash, William L. Hamilton, David Duvenaud, Raquel Urtasun, and Richard Zemel. 2019. Efficient Graph Generation with Graph Recurrent Attention Networks. In *NeurIPS*.

[23] David Liben-Nowell and Jon Kleinberg. 2007. The link-prediction problem for social networks. *Journal of the American society for information science and technology* 58, 7 (2007), 1019–1031.

[24] Avi Ma'ayan, Azi Lipshtat, Ravi Iyengar, and Eduardo D Sontag. 2008. Proximity of intracellular regulatory networks to monotone systems. *IET Systems Biology* 2, 3 (2008), 103–112.

[25] Karolis Martinkus, Andreas Loukas, Nathanaël Perraudin, and Roger Wattenhofer. 2022. SPECTRE: Spectral Conditioning Helps to Overcome the Expressivity Limits of One-shot Graph Generators. *arXiv preprint arXiv:2204.01613* (2022).

[26] John Palowitch, Anton Tsitsulin, Brandon Mayer, and Bryan Perozzi. 2022. GraphWorld: Fake Graphs Bring Real Insights for GNNs. *arXiv preprint arXiv:2203.00112* (2022).

[27] Himchan Park and Min-Soo Kim. 2017. TrillionG: A trillion-scale synthetic graph generator using a recursive vector model. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 913–928.

[28] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Chi Wang, Kuansan Wang, and Jie Tang. 2019. Netsmf: Large-scale network embedding as sparse matrix factorization. In *The World Wide Web Conference*. 1509–1520.

[29] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European semantic web conference*. Springer, 593–607.

[30] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93–93.

[31] C. Seshadhri, Ali Pinar, and Tamara Kolda. 2011. A Hitchhiker's Guide to Choosing Parameters of Stochastic Kronecker Graphs. *CoRR* abs/1102.5046 (01 2011).

[32] Claude E Shannon. 1948. A mathematical theory of communication, Bell Systems Technol. *J* 27, 3 (1948), 379–423.

[33] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[34] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. 2020. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies* 1, 1 (2020), 396–413.

[35] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. 2019. Modeling Tabular data using Conditional GAN. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/254ed7d2de3b23ab10936522dd547b78-Paper.pdf

[36] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. 2018. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International conference on machine learning*. PMLR, 5708–5717.

A ADDING NOISE TO STRUCTURE GENERATOR

Graph $\hat{G}$ generated by (1) will produce oscillations on the degree distribution as described in [31]. To address oscillations we propose to add a noise component on each $max(m, n)$ step of (1). This changes (1) to

$$\theta = \underbrace{\theta_{S,0} \otimes ... \otimes \theta_{S,min(n,m)}}_{min(m,n) times} \otimes \underbrace{\theta_{H,0} \otimes ... \otimes \theta_{H,min(0,n-m)}}_{min(0,n-m) times} \otimes \underbrace{\theta_{V,0} \otimes ... \otimes \theta_{V,min(0,m-n)}}_{min(0,m-n) times},$$

where $\theta_{S,i}$, $\theta_{H,i}$, $\theta_{V,i}$ are noisy versions of θ_S, θ_H, θ_V from (1), respectively. If noise is not added then (**??**) and (1) are equivalent. $\theta_{S,i}$ (and analogically $\theta_{H,i}$, $\theta_{V,i}$) are modifications of θ_S (and θ_H, θ_V, respectively)

$$\theta_{S,i} = \theta_S + N_i(\theta_S), \tag{8}$$

analogically for $\theta_{H,i}$, $\theta_{V,i}$. Mean value of noise added to cascade (1) has to be zero, but careful mathematical analysis shows that also elements of noise matrix N_i added to matrix θ_S (and respectively θ_H, θ_V) have to be zero.

Noise added by N_i depends on θ_S and in practice can be controlled by a single parameter sampled from uniform distribution. An exemplary noise for symmetric θ_S can be

$$N_i = \begin{bmatrix} \frac{-2n_f*a}{a+d} & nf \\ nf & \frac{2n_f*a}{a+d} \end{bmatrix} n_f \sim U[min(\frac{a+d}{2}, b, c)], \tag{9}$$

Where $U[x, y]$ denotes the uniform distribution .... Overall, adding noise on each step of generator requires only $max(n, m)$ parameters.

B ADDITIONAL EXPERIMENTS

B.1 Degree-Distribution - Feature-Distribution

The final graph can be compared visually by plotting the degree distribution vs feature distribution across the feature sets. In Figure 5 we provide a comparison across the original dataset, our properly generated and the baseline methods. Darker regions in this figure correspond to lack of feature values for a particular binned degree. Note as the graph is bipartite in this example, the x-axis is the source degree and y axis is the feature distribution.

B.2 Comparing degree distribution

Our structure generator fits its parameters to degree distribution of original graph according to (5). Estimating the quality of that fitting can be done visually by comparing plots like in Figure 2. Still, in most cases, it isn't easy to assess whether the improvement of the fitting procedure makes a degree distribution of the synthetic graph closer to the original. This holds especially for comparing a different size synthetic graph in terms of N,M and E than the original graph. We propose a single scalar metric that captures the alignment of degree distribution for two graphs. This metric is calculated as follows:

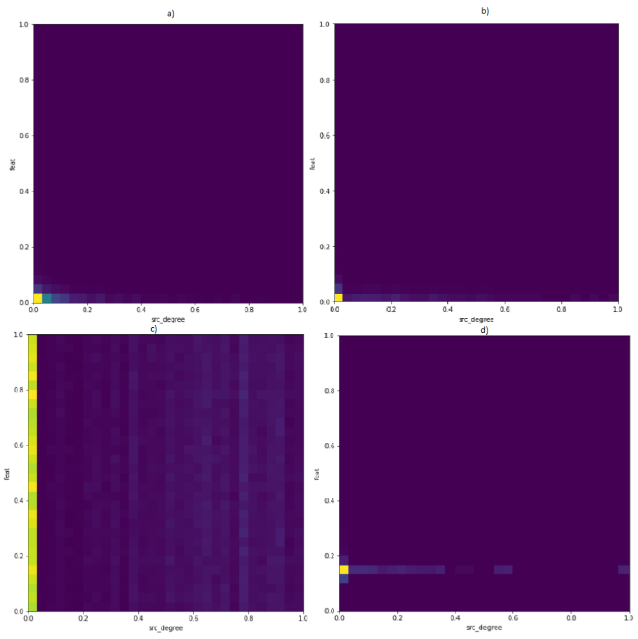


Figure 5: Histograms comparing degree distribution and feature distribution for IEEE-Fraud dataset. a) original graph, b) ours generated, c) randomly generated, d) GraphWorld generated with the added fitting.

$$DCC = \frac{1}{K} \sum_{k \in logspace(0,1)} \frac{c_k^{norm} - c_k^{\hat{norm}}}{c_k^{norm}} \tag{10}$$

where K is the number of distinct degree k sampled logarithmically from $[0, 1]$ c_k^{norm}, $c_k^{\hat{norm}}$ are normalised degree distributions i.e. degree is normalised by maximum degree in the graph and number of nodes is normalised by maximum c_k and $\hat{c_k}$ respectively.

Normalization of the degree distribution is needed as G and $\hat{G}$ may have different sizes both in terms of number of nodes and edges. For G and $\hat{G}$ of the same number of edges E the DCC in (10) simplifies to

$$DCC = \frac{1}{K} \sum_{k \in logspace(0,k_{max})} \frac{c_k - \hat{c_k}}{c_k}. \tag{11}$$

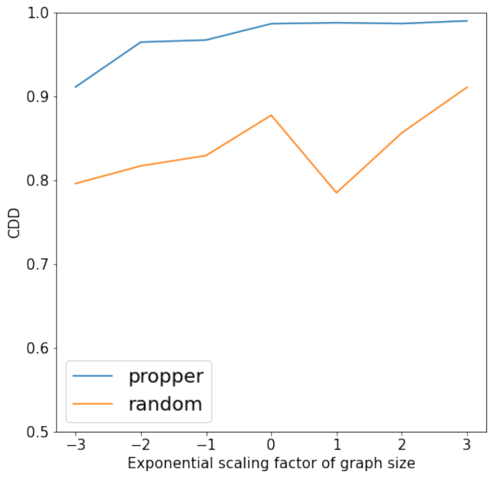
For generating larger graphs, we need to ensure that graph density is preserved i.e.:

$$\frac{E}{N * M} = \frac{\hat{E}}{\hat{N} * \hat{M}}, \tag{12}$$

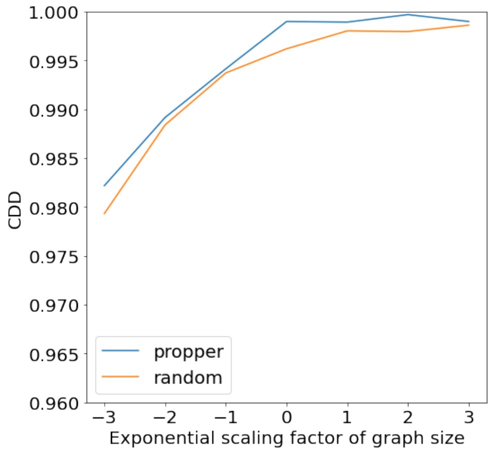
where $\hat{E}, \hat{N}, and \hat{M}$ are number of edges and nodes in partites of $\hat{G}$.

For example, for homogeneous graph when increasing the number of nodes twice, one needs to increase the number of edges four times to preserve the constant density.

Our method consistently outperforms ER model not only for the same graph size but across all scaling factors, see Figure 6. It also provides very high values of CDD for large graphs which in fact means that when generating large synthetic graphs, the degree distribution curve shape preserves its power-law shape. This statement holds for generating smaller graphs that is equivalent to subsampling larger graph in a stratified way where nodes' degree proportion are preserved - this can be primary seen in Tabformer dataset where partites are imbalanced in size (2^{17} user nodes and 2^{11} merchant nodes).



(a)



(b)

Figure 6: CDD coefficient calculated according to 10 for different datasets (a) Tabformer, (b) ieee-fraud. Two generating models are compared: our method marked as 'propper' and random which is ER [9]. X-axis of each graph is an exponential scaling factor by which the number of nodes in each partite is multiplied e.g. 0 means graph of the same size, +3 means graph for which $\hat{N} = 2^3 N$,$\hat{M} = 2^3 M$ and $\hat{E} = 2^6 E$, -3 means graph for which $\hat{N} = 2^{-3} N$,$\hat{M} = 2^{-3} M$ and $\hat{E} = 2^{-6} E$ etc

C DATASET DETAILS

Constructing graphs from a tabular dataset requires capturing sample relations. Unlike graph datasets that have the structure and features given directly, sample relations are not immediately given in tabular settings. As a result, these need to be extracted from the data. In practice, tabular datasets are from very diverse domains ranging from fraud detection to recommender systems and electronic health records; these relationships require to be inferred from the features with domain knowledge. The feature columns used for extracting the corresponding relationship and edges from the tabular dataset are summarized in Table. 9.

Table 9: Details on how to construct graph from tabular features. Node column corresponds to the set of features used to construct node types, and the condition column details the condition that must be satisfied for an edge to exist between the nodes.

Dataset	Nodes	Condition
Tabformer	concat(User, Card) Merchant ID	same row
IEEE-Fraud	concat(7 features)[3] concat(2 features)[4]	same row
Paysim	nameOrig nameDest	same row
Credit	concat(first, last) merchant	same row

D EXPERIMENT DETAILS

We provide details on hyperparameters used in our experiments. Our structural generator has a fitting portion that fits the underlying dataset without requiring the user to specify the parameters. The XGBoost[5] aligner learning rate is set to a default value of 0.1, max depth of 5, and the number of estimators is set to 100, with an alpha value of 10., We set the hidden dimensions to be equal to the input dimension. For all experiments for training to our proposed method, we use Adam [17] optimizer with an initial learning rate of $1e-3$ decayed every ten epochs by a factor of 0.1 and trained for a maximum of 20 epochs with early stopping. Note that we train on the complete input data and do not split the dataset into train, validation, and test splits as we aim to generate a single graph while our input is also a single graph. For most datasets, it suffices to train the feature generator for about 5 epochs. Note that for datasets that contain categorical columns, the embedding size for these columns is set to $min(600, round(1.6 * |D|^{0.56})$ where $|D|$ is the number of unique values for the categorical column. Our code is available on Github [6]

[5]https://rapids.ai/xgboost.html
[6]https://github.com/<anon>/<anon>