Scalable Graph Synthesis with $\boldsymbol{Adj}$ and $1 - \boldsymbol{Adj}$

Jinsung Jeon[*] Jing Liu[†] Jayoung Kim[*] Jaehoon Lee[*] Noseong Park[*]
Jamie Jooyeon Lee[‡] Ozlem Uzuner[‡] Sushil Jajodia[‡]

Abstract

Graph synthesis is a long-standing research problem. Many deep neural networks that learn about latent characteristics of graphs and generate fake graphs have been proposed. However, in many cases their scalability is too high to be used to synthesize large graphs. Recently, one work proposed an interesting scalable idea to learn and generate random walks that can be merged into a graph. Due to its difficulty, however, the random walk-based graph synthesis failed to show state-of-the-art performance in many cases. We present an improved random walk-based method by using negative random walks. In our experiments with 6 datasets and 8 baseline methods, our method shows the best performance in almost all cases. We achieve both high scalability and generation quality.

1 Introduction

Graphs are one of the most popular data types in many fields. Graph synthesis has also become a popular task [4–6, 8, 9, 16, 24, 27, 29–31]. In general, graph synthesis problems can be categorized into the following two types: i) training with and synthesizing many small graphs (cf. Fig. 1 (b)), and ii) training with and synthesizing one large graph (cf. Fig. 1 (c)).

Graph synthesis methods to solve the first type typically learn and synthesize either an upper triangle of adjacency matrix or a list of edges [8, 16, 24, 27, 29–31]. They typically synthesize graphs with hundreds of vertices, e.g., molecular graphs, (see Fig. 1 (b)).

Recently, a generative adversarial network (GAN) based on random walk sampling, called NetGAN [4], has been proposed to solve the second type. Their main idea is to learn and generate short random walks that will be eventually merged into a graph (see Fig. 1 (c)). The scalability of the GAN model is one or two orders of magnitude larger (in terms of the number of vertices or edges) than the first type methods. To our knowledge, NetGAN is the state-of-the-art GAN in scalable graph generation, e.g., tens of thousands of vertices.

NetGAN's generation quality is, however, sub-optimal in many cases. It is very hard to achieve good results in both scalability and generation quality. For instance, some statistics of graphs generated by NetGAN are not close to real graphs even though other non-deep learning based generative methods show reasonable similarity. In response, *we propose NetGAN+ in this paper, and achieve the best performance in almost all cases for graph similarity and link prediction in a shorter training time than that of NetGAN.*

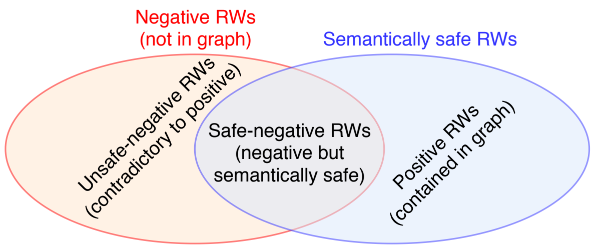
Our main idea is to use both positive and negative random walks to train our NetGAN+ (see Section 3.6 on why it is theoretically better than NetGAN). Positive random walks are the random walks explicitly contained in an original graph, and negative random walks are not contained in the original graph. Positive random walks are safe to include in the synthesized graphs, whereas negative random walks are not always favorable to synthesize (see Fig. 1 (a) and Def. 1). One can consider that our method uses both *what-to-generate* and *what-not-to-generate* simultaneously whereas previous research on GANs usually rely on *what-to-generate* only (e.g., NetGAN uses only positive random walks).

However, one challenge is that some negative random walks are semantically safe (favorable) to synthesize (hereinafter, we refer those random walks as *safe-negative random walks*) and thus, should not be suppressed during the generation process. As a matter of fact, the link prediction task is to reveal those unknown true edges from the negative class. Because of this reason, one should be very careful about utilizing negative random walks. We show that it is sub-optimal to suppress the generation of all negative random walks.

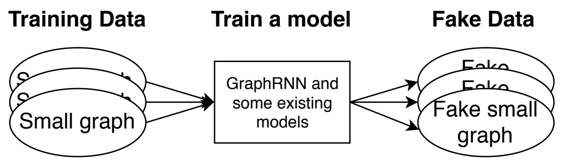
Correctly distinguishing between safe-negative and unsafe-negative random walks is difficult in our case.

[*]Yonsei University, Seoul, South Korea
[†]Walmart Labs, Reston, Virginia, USA
[‡]George Mason University, Fairfax, Virginia, USA

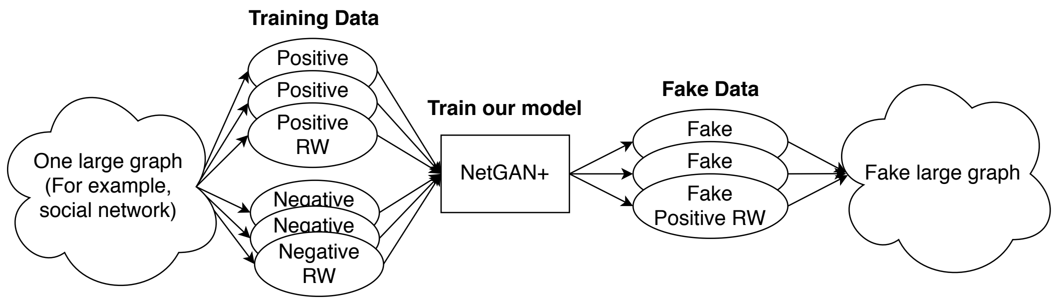
Noseong Park (noseong@yonsei.ac.kr) is the corresponding author. This work was partially supported by the IITP of the Korea government (MSIT) under grant 2020-0-01361, Artificial Intelligence Graduate School Program (Yonsei University), by the Office of Naval Research under grants N00014-18-1-2670 and N00014-20-1-2407, by the Army Research Office under grant W911NF-13-1-0421, and by the National Science Foundation under grant CNS-1822094.



(a) The concept of positive/negative random walks



(b) GraphRNN, GraphGen, and some others



(c) Our proposed NetGAN+

Figure 1: (a) The Venn diagram of all possible cases of random walks (RWs). Positive random walks are already included in a graph. Note that negative random walks consist of two cases: one represents negative random walks that contradict positive ones and the other represents random walks that are safe but mistakenly excluded from the graph. (b) Existing small graph synthesis methods (c) Our large graph synthesis

Thus, we add an additional neural network based on an attention mechanism to identify and exclude safe-negative random walks from the negative class. Therefore, our NetGAN+ consists of three neural networks: a generator, a discriminator, and an attention network. In our setting, the discriminator considers safe-negative and positive random walks as *real*, and unsafe-negative and fake random walks as *fake* (see Fig. 2 (a)).

We conduct experiments with six standard graphs. For graph similarity and link prediction, our NetGAN+ significantly outperforms other baseline methods. We also prove that NetGAN+ reduces to NetGAN if the attention network is ill-trained and does not contribute to the training process. This is, however a rare case, and NetGAN+ has a theoretical ground that guarantees improvements over NetGAN (see Section 3.6).

2 Related Work

Many graph synthesis methods have been proposed [4–6, 8, 9, 16, 24, 27, 29–31] to name a few. Graph syntheses can be categorized into the following two types: i) training with and synthesizing many small graphs as shown in Fig. 1 (b), and ii) training with and synthesizing one large graph as shown in Fig. 1 (c). In general, they are not compatible to each other (See Section 3.7).

In the first type, neural networks, such as GraphRNN [30], GRAN [16], GraphGen [6], and so forth, synthesize an upper triangle of adjacency matrix, whose complexity is $\mathcal{O}(|\mathcal{V}|^2)$, or a list of edges, whose complexity is $\mathcal{O}(|\mathcal{E}|)$. For instance, GraphRNN uses two recurrent neural networks, one to generate a sequence of vertices and the other to generate an adjacency list for each generated vertex, which correspond to an upper triangle of adjacency matrix. GraphGen learns and synthesizes a list of edges after an expensive pre-processing step, called graph canonicalization, to convert a graph into a unique sequence of edges. Its worst-case canonicalization time is $\mathcal{O}(|\mathcal{V}|!)$ which can be improved if vertices and/or edges have labels.

Both GraphRNN and GraphGen did experiments with CITESEER/CORA after randomly sampling many small subgraphs so their synthesis size is same as that of random subgraphs. After setting the size of their synthesis to that of the original CITESEER/CORA graph, they are not properly executed for spatial scalability issues. (See Section 3.7 for their empirical and theoretical complexity issues.).

NeVAE [24], D-VAE [31], and CondGen [29] are

Some papers use GANs to extract features from graphs [17, 28], whose main goals are not generation.

variational autoencoder approaches to generate small molecular or citation graphs which typically consist of hundreds of vertices. There also exist some other adjacency matrix-based methods. For instance, Guo et al. deal with graphs whose vertex numbers are no larger than hundreds [8, 9].

To our knowledge, NetGAN is the state of the art method for the second graph synthesis type, i.e., training with and synthesizing one large graph. NetGAN is similar to other GANs generating texts because it generates random walks that are similar to sentences, i.e., a sequence of vertices vs. a sequence of words. In NetGAN, the generator is an LSTM network that produces a sequence of vertices and the discriminator is also an LSTM network that distinguishes real and fake sequences. NetGAN uses the Gumbel-softmax which is a reparameterization trick of categorical sampling [15]. The dimensionality of the categorical sampling is the same as the number of vertices. For efficiency, NetGAN uses a three-step method where i) the generator LSTM produces a latent vector with a small dimension, ii) the latent vector is projected into a space whose dimensionality is the same as the number of vertices (this projection is also trained), and iii) the Gumbel-softmax performs the categorical sampling in the higher dimensional space. NetGAN uses the following (regularized) Wasserstein GAN (WGAN-GP [7]) loss to train:

$$\begin{aligned}\min_G \max_D \; & \mathbb{E}\big[D(G(z))\big]_{z\sim p_z} - \mathbb{E}\big[D(x)\big]_{x\sim p_{pos}} \\ & + \lambda \mathbb{E}\big[(\|\nabla_{\bar{x}} D(\bar{x})\|_2 - 1)^2\big]_{\bar{x}\sim p_{g\circ x}},\end{aligned}$$

where p_z is a prior distribution; p_{pos} is a distribution of positive random walks; G is a generator function; D is a discriminator function; $\bar{x}$ is a randomly weighted combination of $G(z)$ and x. The discriminator provides feedback on the quality of the generation. In addition, we let p_g be a distribution of fake data induced by the function $G(z)$ from p_z and $p_{g\circ x}$ be a distribution created after the random combination. We typically use $\mathcal{N}(\mathbf{0}, \mathbf{1})$ for the prior p_z.

In fact, generating graphs is a long-standing research problem and there have been proposed many other non-machine learning methods [3, 10, 12, 13, 20, 26]. We will compare with these baseline methods for our experiments.

3 Proposed Method

We introduce our proposed method. We first present an overview of our framework and then describe the technical details.

3.1 Overall Architecture We use lower boldface to denote vectors and upper boldface to denote matrices. Let us first define positive and negative random walks as follows:

Definition 1. *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and its adjacency matrix $\boldsymbol{Adj} \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|}$, positive random walks are random walks on $\boldsymbol{Adj}$ and negative random walks are random walks on $1 - \boldsymbol{Adj}$.*

Our proposed architecture is shown in Fig. 2 (a). Our NetGAN+ consists of the following three neural networks. The generator produces fake random walks. The discriminator is a binary classifier to distinguish the following two classes: i) safe-negative/positive class, and ii) unsafe-negative/fake class. We note that safe-negative and positive (real) random walks constitute a class and unsafe-negative and fake random walks constitute the other class (cf., Fig. 1). The attention network assigns weights to negative random walks. The attention weight is designed to be high for random walks dissimilar to positive ones (i.e., unsafe-negative random walks), and low for those similar to positive ones (i.e., safe-negative random walks). By referring to attention weights, therefore, the discriminator can know which ones are safe-negative and unsafe-negative.

3.2 Generator and Discriminator Networks We adopt the original generator and discriminator networks of NetGAN. They are LSTM networks for dealing with a sequence of vertices. Generating and classifying random walks is very similar to generating and classifying sentences. Thus, standard LSTM networks, which have been shown to perform well at sentence generation and classification, work well in our task.

To further increase scalability, NetGAN uses two transformation matrices to scale down and up inputs and outputs of the two LSTM networks, respectively. For instance, the discriminator reads one-hot vectors but if there are many vertices, the dimensionality of the one-hot vectors will be very large, which is not desired for scalability. Thus, the i-th LSTM output of the generator produces a latent vector $\boldsymbol{o}_i \in \mathcal{R}^h$, where h=128, that will be transformed to a large vector $\boldsymbol{l}_i \in \mathcal{R}^{|\mathcal{V}|}$ to be processed by Gumbel-Softmax as follows:

$$\begin{aligned}\boldsymbol{o}_i &= LSTM_{generator}(\boldsymbol{h}_{i-1}, \boldsymbol{c}_i),\\ \boldsymbol{l}_i &= \boldsymbol{o}_i \boldsymbol{W_{up}},\\ \boldsymbol{v}_i &= categorical\_sampling(\boldsymbol{l}_i),\end{aligned}$$

where $\boldsymbol{o}_i \in \mathcal{R}^h$ is i-th output from the generator, $\boldsymbol{W_{up}} \in \mathcal{R}^{h\times|\mathcal{V}|}$ is a projection matrix for scaling up, and $\boldsymbol{v}_i$ is a vector representing i-th vertex in the generated random walk. In particular, NetGAN uses the Gumbel-softmax for the last sampling function to make the entire process differentiable and trainable without policy gradients.

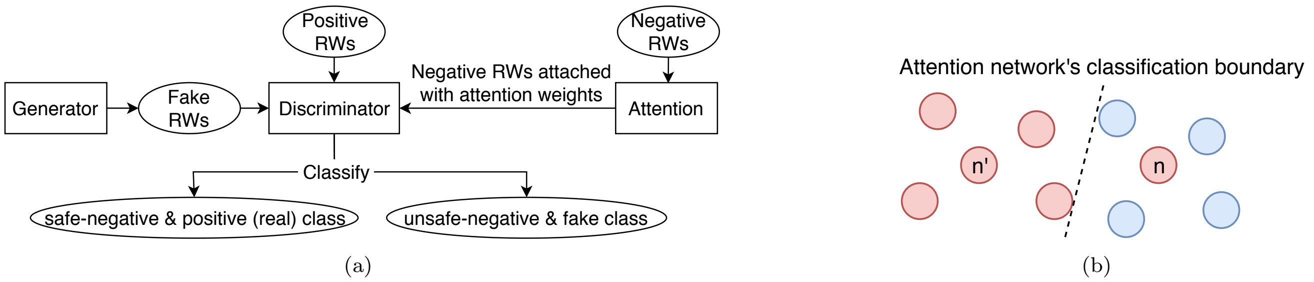
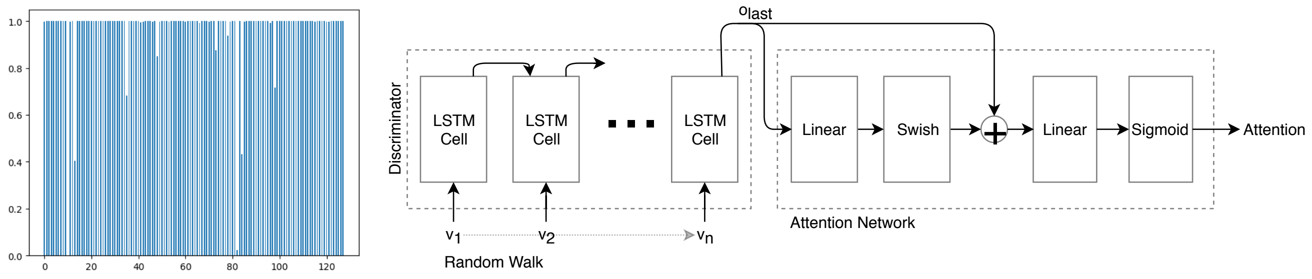

Generator
Fake RWs
Positive RWs
Discriminator
Negative RWs attached with attention weights
Negative RWs
Attention
Classify
safe-negative & positive (real) class
unsafe-negative & fake class
(a)
Attention network's classification boundary
n'
n
(b)

Figure 2: (a) The conceptual workflow of NetGAN+. The attention network assigns weights to negative random walks and the discriminator is trained with all of positive, negative, and fake random walks. (b) Red means negative random walks and blue means positive random walks in the feature space right before the attention network's final sigmoid activation.



(a) Attention weights (b) Attention network

Figure 3: (a) Attention weights of 128 negative random walks of CORA-ML in a mini-batch at the last epoch. Note that the majority of them have large weights close to 1.0, which means the semantically unsafe-negative random walks outnumber the semantically safe-negative random walks. (b) The detailed architecture of the attention network. The attention network takes the last output, denoted as o_{last}, of the discriminator.

The input to the discriminator is scaled down to a latent vector after the multiplication with $\boldsymbol{W_{down}} \in \mathcal{R}^{|\mathcal{V}|\times h}$ for the same reason.

3.3 Attention Network We introduce our attention network in this subsection. Our key idea is using negative random walks to improve the generation quality. Specifically, we can suppress many semantically unsafe random walk generations. Fig. 1 shows all possible categories of random walks. The generation should happen with all the safe random walks, which means we have to identify which negative random walks are semantically safe, and which are unsafe. However, the majority of negative random walks will fall into the category of "semantically unsafe" and correctly identifying them is very challenging. This problem is, in fact, similar to *noisy* data classification [11, 18, 23] in an *unsupervised* manner.

To this end, we design an attention mechanism. Our proposed attention network is to assign a weight to each negative random walk, which we call *sample-wise attention*. Its neural network is defined as follows:

$$\begin{aligned} \boldsymbol{r} &= swish(\boldsymbol{o}_{last}\boldsymbol{W}_1 + \boldsymbol{b}_1),\\ \boldsymbol{m} &= \boldsymbol{o}_{last} + \boldsymbol{r},\\ l &= \boldsymbol{m}\boldsymbol{W}_2 + b_2,\\ attention &= sigmoid(l), \end{aligned}$$

where $\boldsymbol{o}_{last}$ is the last latent vector output of the discriminator; $\boldsymbol{W}_1 \in \mathcal{R}^{h\times h}$, $\boldsymbol{b}_1 \in \mathcal{R}^{h}$, $\boldsymbol{W}_2 \in \mathcal{R}^{h\times 1}$ and $b_2 \in \mathcal{R}$ are parameters; *swish* is the swish activation [22] which is similar to the concept of leaky ReLU; l is a logit value. Fig. 3 (b) is a diagram representing this architecture.

How to calculate the logit is inspired by residual networks. In residual networks, given an input vector

For a better illustrate, we show the discriminator and the attention network as separate networks in the conceptual diagram of Fig. 2 (a). To reduce the processing overhead, however, the attention network does not process from scratch. Referring to the discriminator's opinion (i.e., $\boldsymbol{o}_{last}$), the attention network independently continues to decide the attention weight.

$\boldsymbol{y}$ each residual block learns only $f(\boldsymbol{y})$ that will be later added with $\boldsymbol{y}$, i.e., $\boldsymbol{y} + f(\boldsymbol{y})$ or in our case, $\boldsymbol{m} = \boldsymbol{o}_{last} + swish(\boldsymbol{o}_{last}\boldsymbol{W}_1 + \boldsymbol{b}_1)$. There are two advantages in using the residual approach: i) Learning only the residual part, i.e., $f(\boldsymbol{y})$, is much easier than learning the entire value, i.e., $\boldsymbol{y} + f(\boldsymbol{y})$, from scratch, and ii) $\boldsymbol{y} + f(\boldsymbol{y})$ means basically an ensemble. By nesting many residual blocks, we can have a more complex ensemble. For instance, by nesting two residual blocks, we can have an expanded ensemble of $\boldsymbol{y} + f(\boldsymbol{y}) + f'(\boldsymbol{y} + f(\boldsymbol{y}))$, where f and f' mean the first and second residual block. We use two residual blocks if the number of edges is larger than 20K. This number was empirically found in our experiments after some preliminary tests. We train the attention network using the following loss:

$$\mathcal{L}_{attn} = \mathbb{E}\big[A(x)\big]_{x \sim p_{pos}} - \mathbb{E}\big[A(n)\big]_{n \sim p_{neg}},$$

where A outputs scalar attention weights, p_{pos} is a positive random walk distribution, and p_{neg} is a negative random walk distribution. p_{neg} is further separated into p_{safe} and p_{unsafe}.

The intuition behind this loss is that i) for an obviously positive random walk $x \sim p_{pos}$, $A(x)$ will be close to zero, ii) for an obviously negative random walk $n \sim p_{neg}$, $A(n)$ will be close to one, and iii) for those that are neither obviously positive nor negative, the attention network should decide a value between 0 and 1. For instance, for the two negative random walks n and n' in Fig. 2 (b), the attention network classifies n as positive because n has many positive neighbors in the feature space. By strategically classifying n as positive, the attention network can minimize the loss. This is also the case in many other binary classifications with noisy data. Fig. 3 (a) shows an attention example.

3.4 Overall Training Method The original NetGAN uses the training loss defined by WGAN-GP. We use the same loss for the generator. However, the discriminator in our work uses both positive and negative random walks, and we propose the the following loss — the parts modified by us are denoted in red:

$$\begin{aligned}\mathcal{L}'_D =& \mathbb{E}\big[D(G(z))\big]_{z \sim p_z} - \mathbb{E}\big[D(x)\big]_{x \sim p_{pos}} \\ &+ \mathbb{E}\big[A(n)D(n)\big]_{n \sim p_{neg}} - \mathbb{E}\big[(1 - A(n))D(n)\big]_{n \sim p_{neg}} \\ &+ \lambda\mathbb{E}\big[(\|\nabla_{\bar{x}}D(\bar{x})\|_2 - 1)^2\big]_{\bar{x} \sim p_{g,neg^+ \circ x}}.\end{aligned} \tag{3.1}$$

Before discussion, recall that D in WGAN-GP, NetGAN, and our NetGAN+ returns the Wasserstein distance which theoretically ranges in $[0, \infty]$. In the first red term, the Wasserstein distance of negative random walk n will be multiplied with its attention weight $A(n)$. In the second red term, it is multiplied with $1 - A(n)$. $A(n) \approx 0$ (resp. $A(n) \approx 1$) means that a negative random walk n is semantically safe (resp. unsafe). Only if $A(n) = 0.5$, i.e., neutral, the two terms cancel each other. One of the two terms is more emphasized than the other if $A(n) \neq 0.5$.

The third penalty term should also be modified. In WGAN-GP and NetGAN, they do not use negative samples and require unit gradients for all samples interpolated between real and fake samples (random walks). In NetGAN+, however, the unit gradient requirement should be adjusted accordingly. p_{g,neg^+} means a mixture distribution of fake and negative random walks whose attention weights are larger than ths (we use $ths = 0.9$ in default after some preliminary experiments). This is based on our heuristic because the Wasserstein distance cannot be defined over samples with uncertainty. For those negative random walks with large attention weights close to one, however, we are certain that they are most likely to be semantically unsafe. Thus, our modified regularization term requires unit gradients only when it is certain that they are semantically unsafe. Each of the discriminator, generator, and attention network is alternately trained as in other GANs.

3.5 Merge Random Walks To generate a graph, both NetGAN and our NetGAN+ should generate many random walks and merge them into a graph. We use the method proposed in NetGAN for this. We repeatedly sample many random walks and count the number of occurrences for each edge. After normalizing the number of occurrences, this becomes the probability of the existence of the edges. Lastly, edges are sampled one more time following the probability distribution.

3.6 Theoretical Results It has been known that $WD(p_g, p_{data})$, the Wasserstein distance between the original and generated data distributions, is minimized in the equilibrium state of WGAN-GP. In our case, we prove that $WD(p_g, p_{pos,safe})$, where $p_{pos,safe}$ means a mixture of the two distributions for positive and safe-negative random walks, can be minimized. In NetGAN, however, only $WD(p_g, p_{pos})$ is minimized.

Theorem 3.1. *If $A(n)$, where $n \sim p_{neg}$, is able to correctly identify safe and unsafe-negative random walks, $WD(p_g, p_{pos,safe})$ is minimized in the equilibrium state.*

Proof. If $A(n)$ is always correct, the discriminator considers p_{pos} and p_{safe} are in a class, and p_{neg} and p_{unsafe} are the other class in Eq. (3.1). This implies that $p_{pos,safe}$ corresponds to p_{data} in the original GAN definition and the remaining proof follows the original equilibrium state proof. ∎

THEOREM 3.2. *If the attention network produces neutral predictions, i.e., $A(n) = 0.5$, $\forall n \sim p_{neg}$, then $WD(p_g, p_{pos})$ is minimized in the equilibrium state.*

Proof. If $A(n) = 0.5$, the middle two red terms cancel each other in Eq. (3.1) and p_{g,neg^+} becomes p_g by the definition of p_{neg^+}. Therefore, $\mathcal{L}'_D$ reduces to the discriminator loss of NetGAN and the remaining proof follows the original equilibrium state proof. ∎

In particular, the second theorem teaches us that even when our attention network is not capable of doing a proper task and just outputs neutral opinions, our method guarantees the quality as good as NetGAN.

3.7 Comparison with Existing Work GraphRNN produces an upper triangle of adjacency matrix and GraphGen produces a canonical sequence of edges. There are pros and cons in their approaches. GraphGen requires $\mathcal{O}(\mathcal{E})$ steps (excluding the canonicalization preprocessing step) to generate a graph. More precisely, GraphGen uses a custom LSTM whose sampling size is $\mathcal{O}(\mathcal{E} + 2\mathcal{V})$ every step. Thus, GraphGen requires $\mathcal{O}(\mathcal{E})$ steps, each of which steps has the sampling space of $\mathcal{O}(\mathcal{E} + 2\mathcal{V})$. GraphRNN, which does not have any such pre-processing step, has $\mathcal{O}(\mathcal{V}^2)$

As mentioned at the end of Section 4.1.3 of the GraphGen paper [6], the sizes of the sampled subgraphs range from $1 \leq |\mathcal{V}| \leq 102$ and $1 \leq |\mathcal{E}| \leq 121$ in CITESEER, and $9 \leq |\mathcal{V}| \leq 111$ and $20 \leq |\mathcal{E}| \leq 124$ in CORA in their experiments. In other words, they did not use the full-sized CITESEER and CORA graphs but their random subgraphs. Therefore, the number of vertices in a graph is about a hundred in their experiments.

Both GraphRNN and GraphGen produce GPU memory limitation errors in the final generation phase after training with the random subgraphs and setting their generation sizes to the original CITESEER and CORA sizes — training with the original sized CITESEER and CORA graphs produces the same errors even in the training phase for their high spatial complexity.

On the other hand, NetGAN and NetGAN+ synthesize many short random walks that will be merged into a graph, which enables large scale graph syntheses. A data sample in NetGAN+ is a random walk whose length is pre-determined, i.e., $\mathcal{O}(l)$, where l is the length of random walk. $l = 16$ is used in our experiments. The sampling size at each random walk step in NetGAN and NetGAN+ is $\mathcal{O}(|\mathcal{V}|)$. Therefore, our $\mathcal{O}(l)$ steps of a sampling complexity $\mathcal{O}(|\mathcal{V}|)$ are much more scalable than GraphGen's $\mathcal{O}(\mathcal{E})$ steps of a sampling complexity $\mathcal{O}(\mathcal{E} + 2\mathcal{V})$.

The graph canonicalization of GraphGen augments a sequence of edges with some additional information (to make it unique) and thus, what it has to generate is more than a sequence of vertex identification numbers. Therefore, the sampling size in a step becomes large.

Table 1: Statistics of six graphs we used. Their types are either citation network, knowledge graph, or political sentiment network. Note that each of them is a large graph, rather than consisting of many small graphs.

Name	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	Name	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$
CORA-ML [19]	2,810	7,981	PUBMED [25]	19,717	44,324
CORA [19]	18,800	64,529	DBLP [21]	16,191	51,913
CITESEER [25]	2,110	3,757	POLBLOGS [2]	1,222	16,714

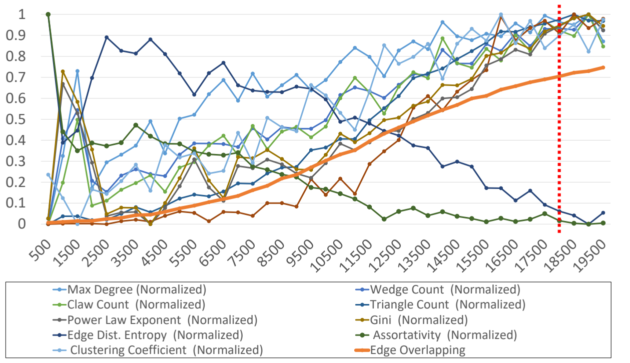


Figure 4: Graph statistics in CORA-ML are saturated around the epoch number 18K where the edge overlap is 70% and do not improve more even with a larger edge overlap — for NetGAN, this statistics saturation occurs around 40K epochs with the edge overlap of 50%.For better visualization, we normalize statistics.

4 Experimental Results

We evaluate our graph generation method in two different tasks with six datasets. First, we compare various graph statistics (such as degree, clustering coefficient, edge distribution, etc.) between ground-truth graphs and generated graphs. We consider many other baseline graph synthesis methods: CM [20], DC-SBM [12], ERGM [10], BTER [26], VGAE [14], and NetGAN [4]. We exclude all other non-scalable methods [8, 16, 24, 27, 29–31] because i) their task is different from our task, and ii) they mostly produce GPU memory limitation errors for our large graphs.

Second, we perform link prediction. For this task, we consider a different set of link prediction-oriented baseline methods: Adamic-Adar [1], DC-SM, node2vec [13], VGAE, and NetGAN. We perform all these experiments using the six datasets summarized in

Table 2: Graph statistics of real and fake graphs for CITESEER and CORA-ML. The best results that are closest to the ground truth values are marked in bold font. LCC stands for the size of the largest connected component and Assort. means assortativity.

	Method	Max Degree	LCC	Wedge Count	Claw Count	Triangle Count	Power Law Exponent	Gini	Edge Dist. Entropy	Assort.	Clustering Coefficient
CITESEER	Ground Truth	77	2,110	16,824	125,701	451	2.239	0.404	0.959	-0.022	1.08e-2
	DC-SBM	53	1,697	15,531	69,818	257	2.066	0.502	0.938	0.022	7.75
	ERGM	66	1,753	16,346	80,510	416	2.0	0.474	0.945	0.052	1.49e-2
	BTER	70	1,708	18,193	113,425	**449**	2.049	0.491	0.940	0.065	1.22e-2
	VGAE	92	**2,110**	8,141	6,611	2	2.039	0.256	0.986	-0.057	1.35e-3
	NetGAN	63	2,053	15,202	94,149	227	2.204	0.385	0.963	-0.054	7.71e-3
	NetGAN+ (w/o Attn)	67	2,044	15,654	100,756	377	2.165	0.392	0.943	-0.052	9.54e-3
	NetGAN+	**77**	2,023	**16,819**	**130,821**	417	**2.231**	**0.399**	**0.959**	**-0.048**	**0.96e-2**
CORA-ML	Ground Truth	240	2,.810	101,872	3.1e6	2,814	1.86	0.482	0.942	-0.075	2.73e-3
	DC-SBM	165	2,474	73.921	1.2e6	1,403	1.814	0.523	0.934	-0.052	3.30e-3
	ERGM	243	2,489	98,615	**3.1e6**	2,293	1.786	0.517	0.932	**-0.077**	2.17e-3
	BTER	199	2,439	91,813	2.0e6	3,060	1,787	0.515	0.935	0.033	4.62e-3
	VGAE	13.1	**2,810**	31,290	46,586	14	1.674	0.223	0.990	-0.010	1.17e-3
	NetGAN	233	2,807	86,763	2.6e6	1,588	1.793	0.42	0.954	-0.066	2.44e-3
	NetGAN+ (w/o Attn)	208	2,808	83,926	2.03e6	1,477	1.813	0.443	0.950	-0.080	2.17e-3
	NetGAN+	**240**	2,802	**99,243**	3.03e6	**2,815**	**1.847**	**0.474**	**0.942**	-0.083	**2.78e-3**

Table 1.

For the validation in the first graph synthesis experiments, the original NetGAN uses the overlapping edge ratio of 50% (i.e., half of generated edges exist in the original graph) as an early stopping criterion because it gives the most similar graph statistics to the original graph. If the ratio is larger than 50%, its graph statistics similarity is rather decreased. However, our NetGAN+ shows different behaviors and its best performance is made around the ratio of 70%. We think this is due to our inclusion of negative random walks during training. We use the overlapping edge ratio of 70% as the best model selection criterion (see Fig. 4). Note that each original graph is known for training so we can directly compare graph statistics between synthesized and real graphs.

For the validation in the second link prediction experiments, we used the official validation set contained in each dataset.

After contacting the authors of NetGAN, we could get all recommended hyperparameter configurations. We use the following improved hyperparameter configurations after preliminary experiments, which are a little different from the recommended configuration: i) For CORA, DBLP, and POLBLOGS, the generator LSTM has 100 units, and the discriminator LSTM has 80 units; ii) The L-2 regularization coefficients of the discriminator and the generator are 5e-5 and 1e-7 respectively; ii) The positive random walk sampling has two parameters, $p = \{0.5, 1.0, 2.0, 4.0\}$ and $q = \{0.5, 1.0, 2.0, 4.0\}$. The length of random walks is 16 in all experiments. We run on machines with Tesla V100 (32GB VRAM), i7 CPU, and 256GB RAM. We execute with five different seed values and report their average performance.

4.1 Graph Statistics

We use CITESEER and CORA-ML for this task and compare the following graph statistics: the maximum degree, the number of vertices in the largest connected component, the number of wedge/claw/triangle-shaped connections, the exponent of the fitted power-law degree distribution, and so forth. Detailed statistics are summarized in Table 2.

For CORA-ML, our NetGAN+ shows the best similarity for the max degree, wedge/triangle count, clustering coefficient, etc. NetGAN+ significantly outperforms NetGAN and other baseline methods. For some metrics such as the max degree, triangle count, and edge distribution entropy, NetGAN+ successfully reproduces all the ground truth values. NetGAN also shows a reasonable generation for the largest connected component.

This is somewhat counter-intuitive. We analyzed about this phenomenon and concluded that it recalls existing edges to capture latent characteristics but too much recall decreases the flexibility of generation.

Table 3: Link prediction in AUCROC and Average Precision (AP)

Method	CORA-ML		CORA		CITESEER	
	AUC	AP	AUC	AP	AUC	AP
Admic/Adar	92.16	85.43	93.00	86.18	88.69	77.82
DC-SBM	96.03	95.15	98.01	97.45	94.77	93.13
node2vec	92.19	91.76	**98.52**	**98.36**	95.29	94.58
VGAE	95.79	96.30	97.59	97.93	95.11	96.31
NetGAN (500K)	94.00	92.32	82.31	68.47	95.18	91.93
NetGAN (100M)	95.19	95.24	84.82	88.04	95.30	96.89
NetGAN+ (w/o Attn, 2.5M)	94.54	95.03	95.89	96.08	96.31	96.89
NetGAN+ (2.5M)	**96.10**	**96.49**	97.94	97.10	**96.46**	**96.90**

Method	DBLP		PUBMED		POLBLOGS	
	AUC	AP	AUC	AP	AUC	AP
Admic/Adar	91.13	82.48	84.98	70.14	85.43	92.16
DC-SBM	**97.05**	96.57	96.76	95.64	95.46	94.93
node2vec	96.41	96.36	96.49	95.97	85.10	83.54
VGAE	96.38	**96.93**	94.50	96.00	93.73	94.12
NetGAN (500K)	82.45	70.28	87.39	76.55	95.06	94.61
NetGAN (100M)	86.61	89.21	93.41	94.59	95.51	94.83
NetGAN+ (w/o Attn, 2.5M)	94.23	94.54	91.65	94.78	95.46	94.95
NetGAN+ (2.5M)	96.35	96.63	**96.95**	**96.11**	**95.73**	**95.34**

For CITESEER, NetGAN+ still shows the best performance and clearly outperforms NetGAN.

Fig. 4 shows the improvement pattern of graph statistics over training time. Around the epoch number 18,000, all the statistics are stabilized and the best result reported in Table 2 is produced. For the same chart of NetGAN, readers are referred to Figure 3 in their paper [4]. After the epoch number 40,000 in their figure, there are no more improvements for the graph statistics and edge overlapping. This proves the efficacy of our NetGAN+ over the original NetGAN. In less than 50% of the training epochs, NetGAN+ achieves the overwhelming performance.

4.2 Link Prediction We evaluate link prediction in AUCROC and average precision (AP) and summarize the results in Table 3. NetGAN+ achieves the best link prediction performance for CORA-ML, CITESEER, PUBMED, and POLBLOGS. For this task, we generate 2.5M random walks with our NetGAN+ and construct the normalized frequency (probability) matrix described in Section 3.5 and use the same protocol as was done with NetGAN. Note that NetGAN requires at least 100M for stable predictions whereas NetGAN+ requires 40 times smaller random walks (2.5M). Thus, NetGAN+ not only improves the link prediction accuracy but also decreases the generation time.

Our attention network is parasitic on the discriminator and does not have many parameters to learn so it incurs small additional overheads. So NetGAN+ converges about 50% faster than NetGAN in our machine.

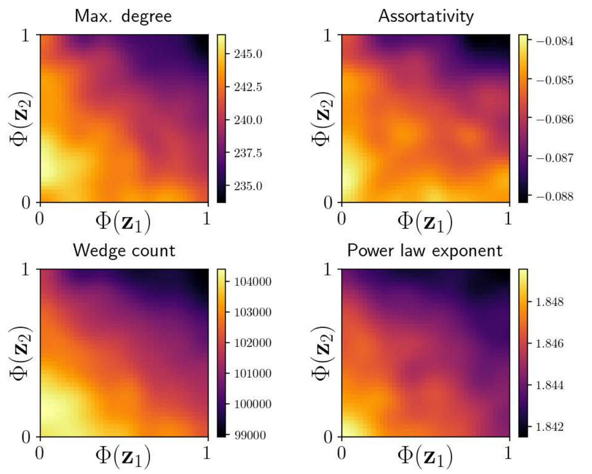


Figure 5: The interpolation of fake graphs for CORA-ML

4.3 Without the Attention Network To check the efficacy of the proposed attention mechanism, we fix the attention weight to 1.0 for all negative random walks and repeat the experiments. This is equivalent to removing the attention network and considering all negative random walks as semantically unsafe.

For graph statistics, NetGAN+ without the attention network (marked with "w/o Attn" in Table 2) is comparable to (or slightly better than) NetGAN but worse than NetGAN+. In the link prediction results in Table 3, NetGAN+ consistently outperforms NetGAN+ (w/o Attn).

5 Interpolation of Latent Vectors

Interpolation is one way to sample generations and check the diversity of generations in GANs. After training, we project the latent vectors used as inputs to the generator and their corresponding graph statistics onto a 2-dimensional space.

Fig. 5 shows the interpolation for CORA-ML. It is obvious that there exist several patterns that graph statistics are gradually changed in the latent vector space. A wide range of graph statistics can be generated as shown in the figure, which means we can generate a diverse set of fake graphs.

6 Conclusions

We presented an advanced, random walk-based graph generating method. To identify semantically safe and unsafe negative random walks, we designed one sample-wise attention mechanism. We conducted in-depth experiments with six datasets. Our approach equipped with the proposed method significantly outperforms all the baseline methods for graph statistics and link prediction in almost all cases. We also theoretically proved that NetGAN+ is as good as NetGAN even when the attention network is not trained properly.

References

[1] L. A. Adamic and E. Adar, *Friends and neighbors on the web*, SOCIAL NETWORKS, 25 (2001), pp. 211–230.

[2] L. A. Adamic and N. Glance, *The political blogosphere and the 2004 u.s. election: Divided they blog*, In LinkKDD, 2005.

[3] E. A. Bender and E. Canfield, *The asymptotic number of labeled graphs with given degree sequences*, Journal of Combinatorial Theory, Series A, 24 (1978), pp. 296 – 307.

[4] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, *NetGAN: Generating graphs via random walks*, in ICML, 2018.

[5] N. D. Cao and T. Kipf, *Molgan: An implicit generative model for small molecular graphs*, CoRR, abs/1805.11973 (2018).

[6] N. Goyal, H. Vardhan Jain, and S. Ranu, *GraphGen: A Scalable Approach to Domain-agnostic Labeled Graph Generation*, arXiv e-prints, (2020), arXiv:2001.08184.

[7] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, *Improved training of wasserstein gans*, arXiv preprint arXiv:1704.00028, (2017).

[8] X. Guo, L. Wu, and L. Zhao, *Deep graph translation*, CoRR, abs/1805.09980 (2018).

[9] X. Guo, L. Zhao, C. Nowzari, S. Rafatirad, H. Homayoun, and S. M. P. Dinakarrao, *Deep multi-attributed graph translation with node-edge coevolution*, in ICDM, 2019.

[10] P. W. Holland and S. Leinhardt, *An exponential family of probability distributions for directed graphs*, Journal of the American Statistical Association, 76 (1981), pp. 33–50, .

[11] I. Jindal, M. S. Nokleby, and X. Chen, *Learning deep networks from noisy labels with dropout regularization*, CoRR, abs/1705.03419 (2017).

[12] B. Karrer and M. E. J. Newman, *Stochastic blockmodels and community structure in networks*, Phys. Rev. E, 83 (2011).

[13] M. Kim and J. Leskovec, *Modeling social networks with node attributes using the multiplicative attribute graph model*, CoRR, abs/1106.5053 (2011).

[14] T. N. Kipf and M. Welling, *Variational graph autoencoders*, CoRR, abs/1611.07308 (2016).

[15] M. J. Kusner and J. M. Hernández-Lobato, *GANS for Sequences of Discrete Elements with the Gumbel-softmax Distribution*, ArXiv e-prints, (2016), arXiv:1611.04051.

[16] R. Liao, Y. Li, Y. Song, S. Wang, W. Hamilton, D. K. Duvenaud, R. Urtasun, and R. Zemel, *Efficient graph generation with graph recurrent attention networks*, in NeurIPS, 2019.

[17] W. Liu, P. Chen, F. Yu, T. Suzumura, and G. Hu, *Learning graph topological features via gan*, IEEE Access, 7 (2019).

[18] X. Liu, S. Li, M. Kan, S. Shan, and X. Chen, *Self-error-correcting convolutional neural network for learning with noisy labels*, in FG, 2017.

[19] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, *Automating the construction of internet portals with machine learning*, Inf. Retr., 3 (2000), pp. 127–163.

[20] M. Molloy and B. Reed, *A critical point for random graphs with a given degree sequence*, Random Structures & Algorithms, 6, pp. 161–180.

[21] S. Pan, J. Wu, X. Zhu, C. Zhang, and Y. Wang, *Tri-party deep network representation*, in IJCAI, 2016.

[22] P. Ramachandran, B. Zoph, and Q. V. Le, *Searching for activation functions*, CoRR, abs/1710.05941 (2017).

[23] S. E. Reed, H. Lee, D. Anguelov, C. Szegedy, D. Erhan, and A. Rabinovich, *Training deep neural networks on noisy labels with bootstrapping*, CoRR, abs/1412.6596 (2014).

[24] B. Samanta, A. De, N. Ganguly, and M. Gomez-Rodriguez, *Nevae: A deep generative model for molecular graphs*, in AAAI, 2019.

[25] P. Sen, G. M. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, *Collective classification in network data*, AI Magazine, 29 (2008), pp. 93–106.

[26] C. Seshadhri, T. G. Kolda, and A. Pinar, *Community structure and scale-free collections of erdös-rényi graphs*, CoRR, abs/1112.3644 (2011).

[27] S. Tavakoli, A. Hajibagheri, and G. Sukthankar, *Learning social graph topologies using generative adversarial neural networks*, in SBP-BRiMS, 2017.

[28] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo, *Graphgan: Graph representation learning with generative adversarial nets*, in AAAI, 2018.

[29] C. Yang, P. Zhuang, W. Shi, A. Luu, and P. Li, *Conditional structure generation through graph variational generative adversarial nets*, in NeurIPS, 2019.

[30] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, *Graphrnn: A deep generative model for graphs*, CoRR, abs/1802.08773 (2018).

[31] M. Zhang, S. Jiang, Z. Cui, R. Garnett, and Y. Chen, *D-vae: A variational autoencoder for directed acyclic graphs*, in NeurIPS, 2019.