NETWORK MOTIF PREDICTION USING GENERATIVE MODELS FOR GRAPHS

BY

ANUTHTHARI GAMARALLAGE

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2020

Urbana, Illinois

Adviser:

Professor Olgica Milenkovic

ABSTRACT

Graphs are commonly used to represent pairwise interactions between different entities in networks. Generative graph models create new graphs that mimic the properties of already existing graphs. Generative models are successful at retaining the pairwise interactions of the underlying networks but often fail to capture higher-order connectivity patterns between more than two entities. A network motif is one such pattern observed in various real-world networks. Different types of graphs contain different network motifs, an example of which are triangles that often arise in social and biological networks. Motifs model important functional properties of the graph. Hence, it is vital to capture these higher-order structures to simulate real-world networks accurately. This thesis introduces a motif-targeted graph generative model based on a generative adversarial network (GAN) architecture that generalizes and outperforms the current benchmark approach, NetGAN, at motif prediction. This model and its extension to hypergraphs are tested on real-world social and biological network data, and they are shown to be better at both capturing the underlying motif statistics in the networks as well as predicting missing motifs in incomplete networks.

ACKNOWLEDGMENTS

The work was supported by the NSF Center for Science of Information under grant number 0939370.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Given the ubiquity of network structures in real-world data, graph generative models have been studied extensively as a means of simulating graphs with different properties. Classical stochastic models, such as the Erdős-Rényi, Barabasi-Albert, and the stochastic block model generate graphs based on a predefined set of parameters, such as the probability of edge formation within and between communities [1]. In contrast, modern approaches to graph generation based on deep learning, including NetGAN [2], GraphGAN [3], and GraphRNN [4], are flexible enough to learn multiple different properties of an input graph simultaneously. The graphs generated by these architectures may be used for downstream learning tasks such as data augmentation [5], recommendation [6], and link prediction [7].

Many real-world networks consist of entities with complex mutual interrelations. Such networks cannot be modeled effectively as graphs with simple pairwise relations, despite the fact that pairwise relations provide a wealth of information for learning. Studying higher-order relationships in a graph is fundamental for our understanding of the network behavior and function. Higher-order relationships are usually termed hyperedges (collections of more than two nodes) [8] or network motifs (recurrent node connectivity patterns that are statistically significant compared to some ground truth random graph model) [9]. These higher-order structures are the actual building blocks of complex networks, as they capture fundamental functional properties.

Existing implicit graph generative models successfully capture pairwise relationships within the graph and associated graph statistics, but they are not as successful in retaining higher-order relationships like motifs or hyperedges. To address this issue, we propose **Multi-MotifGAN (MMGAN)**, a novel motif-targeted graph generative model that preserves network motif statistics in the output graphs. MMGAN generalizes NetGAN, an architecture that uses random walks on an input graph to learn characteristics of the network. The generalization consists of combining multiple random walk statistics, where each type of random walk is biased towards one type of motif structure. We consider two variants of MMGAN: the first is designed to reflect the motif statistics of the input graph accurately, and the second aims to improve motif prediction in networks with missing edges. Both variants combine multiple random walk outputs generated by differently biased GANs, each of which targets a specific motif type. We show experimentally that MMGAN outperforms benchmark generative models such as NetGAN at retaining multiple network motif statistics of the original graph, as evidenced by its competitive results in generation and link prediction on real-world social media networks.

We also propose an extension of the MMGAN architecture named MMGAN-H, which is designed for motif prediction in hypergraphs. Hypergraphs are generalizations of graphs: a graph consists of nodes and edges which connect two nodes to each other, while a hypergraph consists of nodes and hyperedges which connect multiple nodes to each other. We show that MMGAN-H achieves better motif prediction on real-world biological networks that are modeled as hypergraphs.

1.1 Preliminaries

Some key concepts that will be used in later chapters are introduced below.

1.1.1 Network Motif

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph with a set of nodes $\mathcal{V} = \{1, \ldots, n\}$ and a set of undirected edges $\mathcal{E} = \{(i, j) \mid i, j \in V\}$. Given some network consisting of n entities interacting with each other, the nodes represent the different entities while the edges represent pairwise interactions between them. A *subgraph* of $\mathcal{G}$ is a smaller graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ contained within $\mathcal{G}$ such that $\mathcal{V}' \subset \mathcal{V}$ and $\mathcal{E}' \subset \mathcal{E} \cap (\mathcal{V}' \times \mathcal{V}')$. Suppose $\mathcal{G}'' = (\mathcal{V}'', \mathcal{E}'')$ is another such subgraph. Then, $\mathcal{G}'$ and $\mathcal{G}''$ are *isomorphic* to each other if they both consist of the same number of nodes with the same pattern of edge connectivity. More formally, the two subgraphs are isomorphic if there exists a bijection $f : \mathcal{V}' \to \mathcal{V}''$ such

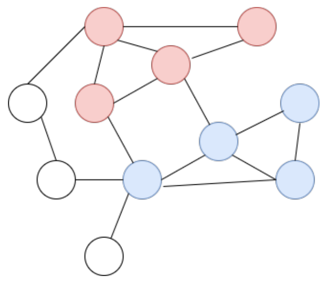
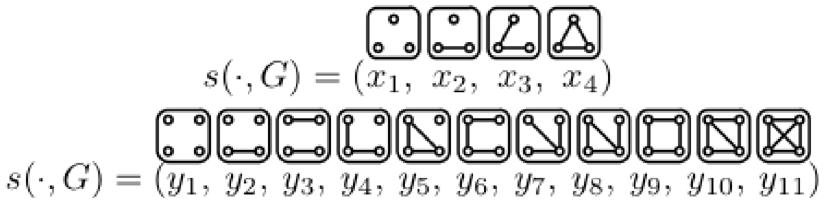


Figure 1.1: *A graph with two isomorphic subgraphs highlighted in red and blue. The frequency of this subgraph type is two. The frequency of triangular subgraphs is four.*



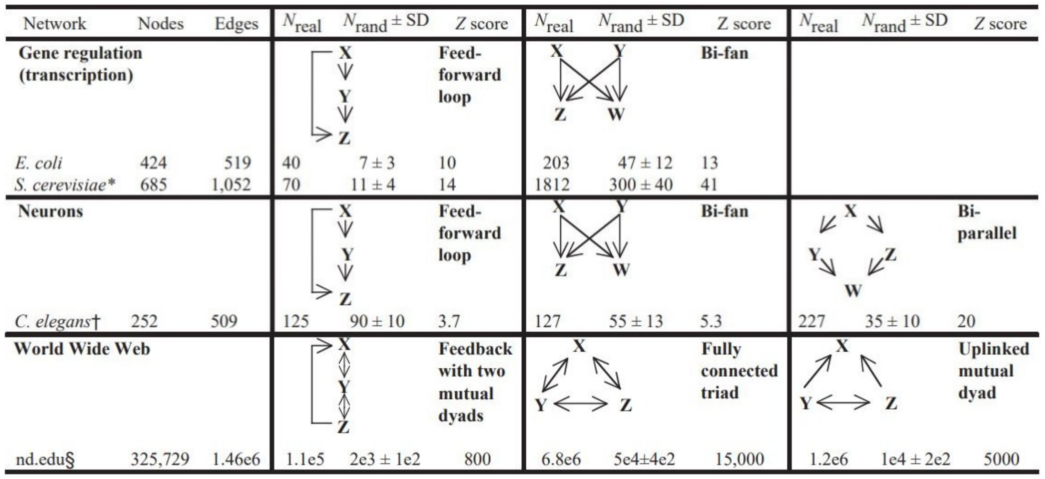
$s(\cdot, G) = (x_1,\ x_2,\ x_3,\ x_4)$
$s(\cdot, G) = (y_1,\ y_2,\ y_3,\ y_4,\ y_5,\ y_6,\ y_7,\ y_8,\ y_9,\ y_{10},\ y_{11})$

Figure 1.2: *Motifs in social networks, adapted from a study by Ugander et al. [11].*

that the edge $(i, j)'$ exists in $\mathcal{G}'$ if and only if the edge $(f(i), f(j))$ exists in $\mathcal{G}''$. Figure 1.1 shows such isomorphic subgraphs. The *frequency of a subgraph* $\mathcal{G}'$ is the number of appearances of subgraphs in $\mathcal{G}$ that are isomorphic to $\mathcal{G}'$.

A *randomized network* of $\mathcal{G}$, represented by $\mathcal{G}_r$, is a graph with the same number nodes and the same node degree distribution as $\mathcal{G}$, i.e. each node in $\mathcal{G}_r$ has the same number of incoming and outgoing edges as the corresponding node in $\mathcal{G}$ [9]. For an undirected graph like $\mathcal{G}$, all edges are both incoming and outgoing, so $\mathcal{G}_r$ having the same degree distribution as $\mathcal{G}$ is essentially each node in $\mathcal{G}_r$ having the same number of edges as the corresponding node in $\mathcal{G}$.

A *network motif* is defined as a subgraph that recurs in a network $\mathcal{G}$ with a higher frequency than in a randomized network $\mathcal{G}_r$ [9]. Network motifs were originally studied in the context of gene regulation networks [9, 10], but the presence of distinct network motifs in different types of real-world networks such as food webs, the world wide web, social networks, power grid networks, etc., has been established in prior literature [9, 11, 12]. Figure 1.2 shows network motifs prevalent in social networks and Figure 1.3 shows those present in biological networks like gene regulation and neural networks, as well as a social network like the world wide web.



Network	Nodes	Edges	N_{real}	$N_{rand} \pm$ SD	Z score	N_{real}	$N_{rand} \pm$ SD	Z score	N_{real}	$N_{rand} \pm$ SD	Z score
Gene regulation (transcription)					**Feed-forward loop**			**Bi-fan**			
E. coli	424	519	40	7 ± 3	10	203	47 ± 12	13			
*S. cerevisiae**	685	1,052	70	11 ± 4	14	1812	300 ± 40	41			
Neurons					**Feed-forward loop**			**Bi-fan**			**Bi-parallel**
C. elegans†	252	509	125	90 ± 10	3.7	127	55 ± 13	5.3	227	35 ± 10	20
World Wide Web					**Feedback with two mutual dyads**			**Fully connected triad**			**Uplinked mutual dyad**
nd.edu§	325,729	1.46e6	1.1e5	2e3 ± 1e2	800	6.8e6	5e4±4e2	15,000	1.2e6	1e4 ± 2e2	5000

Figure 1.3: *Different types of network motifs found in real-world networks, adapted from the original study on network motifs by Milo et al. [9].* N_{real} *gives the number of motifs of the given type found in the real network,* N_{rand} *gives the number of such motifs found in randomized networks.*

These network motifs generally represent some fundamental functional properties of the network. For example, triangles are prevalent in social networks such as the world wide web (Figure 1.3). This pattern of interaction makes intuitive sense: if two entities A, B communicate with each other, and two entities B, C communicate with each other, it is likely that A, C communicate with each other as well. Thus, when generating graphs to be statistically similar to a real-world network or trying to predict unobserved subgraphs, it is vital to preserve the motif structures present in the network under consideration.

1.1.2 Hypergraphs

A graph consists of nodes and edges that represent a relationship between pairs of nodes. A *hypergraph* is a generalization of a graph that allows edges to exist between more than two nodes, a construction known as a *hyperedge*. Let $\mathcal{H} = (\mathcal{V}, \mathcal{E}_H)$ be a hypergraph. Then if $\mathcal{V} = \{1, \ldots, n\}$ denotes the nodes, $\mathcal{E}_H = \{(v_1, \ldots, v_k) \mid v_1, \ldots, v_k \in V,\ 2 \leq k \leq n\}$ denotes the set of hyperedges. The hyperedges in a hypergraph can be of varying sizes and they are useful for representing interactions between small groups of nodes. Figure 1.4 depicts such a hypergraph.

Note that network motifs and hyperedges are similar constructions. Both

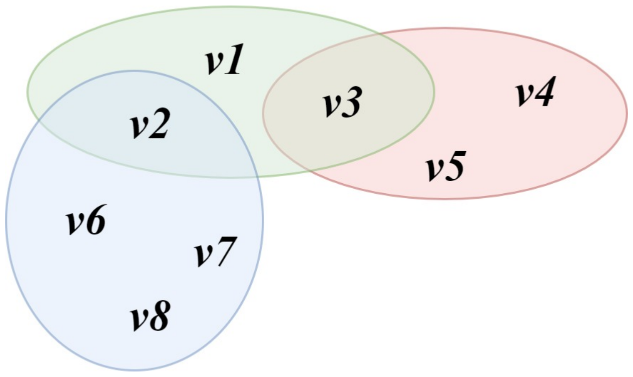


Figure 1.4: *A hypergraph of eight nodes and three hyperedges.*

represent relationships between multiple nodes. A network motif is a more specific type of hyperedge since it is requires a particular pairwise connectivity pattern between the nodes in the motif, whereas a hyperedge usually does not give us any information about pairwise connections. Thus, the different types of three-node network motifs in Figure 1.2 can all be considered hyperedges of size three, while all the different types of four-node motifs can be considered hyperedges of size four.

1.1.3 Generative Adversarial Networks (GAN)

Given the availability of large data sets and an increase in computational capabilities in recent years, neural networks have emerged as the standard tool for machine learning tasks. They are particularly suited for generative tasks since they are able to learn the implicit features of a data set far more efficiently than classical machine learning methods, which in turn leads to the output of a generative model resembling the original data more accurately. The generative adversarial network (GAN) [13] introduced in 2014 is one the of most popular and most effective neural network architectures used for generative modeling, and we base our motif-targeted graph generative model on a GAN architecture.

A GAN consists of a *generator* G and a *discriminator* D, which work against each other during the training phase to optimize each other's predictions. The generator maps a random vector $\mathbf{z}$, sampled from a prior noise distribution $p_{\mathbf{z}}$, to a sample $G(\mathbf{z})$. The discriminator maps an input data point x to a probability $D(x)$, which is the probability that x came from p_{data}, the distribution of the input data. The GAN then optimizes the objective:

$$\min_G \max_D V(D,G) = \mathbf{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbf{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]$$

Here, G is trained to output samples that are nearly indistinguishable from those from the real distribution p_{data}, and D is trained to accurately identify which samples are from p_{data} and which were generated by G. This objective is further equivalent to minimizing the Jensen-Shannon Divergence between the noise and real data distributions:

$$JS(p_{data}, p_m) = KL(p_{data}||p_m) + KL(p_m||p_{data}) \quad \text{where } p_m = \frac{p_{data}+p_z}{2}$$

While the GAN architecture is highly effective, it has some shortcomings in the generated output and is sometimes difficult to train. Mode collapse, or the generated output samples not having enough diversity, can occur with GANs. Further, it runs into the vanishing gradient problem, which makes it difficult to train the network with an approach like stochastic gradient descent, and it is sensitive to hyperparameter choice, which may result in the network not converging. An improved version of the GAN architecture, known as the *Wasserstein GAN* [14], was introduced in 2017 to address these issues.

The Wasserstein GAN essentially changes the metric for comparing the distributions of the noise and real data. The standard GAN minimizes the Jensen-Shannon divergence, while the Wasserstein GAN minimizes the Wasserstein-1 distance, also known as the Earth Mover distance:

$$W(p_{data}, p_z) = \inf_{\gamma\in\Pi(p_{data},p_z)} \mathbf{E}_{(x,y)\sim\gamma}\left[||x-y||\right]$$

where $\Pi(p_{data}, p_z)$ denotes the set of all joint distributions γ whose marginals are respectively p_{data} and p_z

This approach has been shown empirically to outperform the predictive abilities of the standard GAN architecture while also removing the difficulties in training and parameter tuning [14].

CHAPTER 2

REVIEW OF RELATED LITERATURE

Our goal is to create a graph generative model that, given an input graph, generates a realistic graph that resembles the input in terms of motif statistics. Further, we expect to use this generative model to predict missing network motifs in an incomplete input graph. We examine some existing methods for graph generation below.

2.1 Graph Generative Models

Generative models for graphs can be broadly classified into two types based on their how they are parameterized. The parametrization could be *explicit*, where we will need to provide the model with parameter values to determine the properties of the output graph, or *implicit*, where the model will learn these values autonomously given some training data.

2.1.1 Generative Models with Explicit Parametrization

These models use a predefined set of parameters selected by the user to generate graphs. Many classical stochastic models for graph generation fall under this category:

1. Erdős-Rényi Model

   This is a model with two parameters: n, the number of nodes in the graph, and p, the probability that given two nodes i and j there is an edge (i, j) between them. The produced graph is an instance of the random variable $G(n, p)$. The edges between any two nodes are generated completely independently of all other edges, which is not very realistic in real-world networks. Thus, this model is not sufficient for modeling many real-life network phenomena.

2. Barabasi-Albert Model

   This graph generative model generates graphs that are scale-free, i.e. the distribution of node degrees (the number of edges each node is part of) follows a power law (the smallest degrees are far more likely than extremely large degrees). Many real-world networks are observed to be scale-free and this generative method attempts to model that evolution of such networks using a preferential attachment approach: new nodes created in the graph are more likely to form an edge with existing nodes that have a high degree. The model uses parameters n, the number of nodes in the graph, and $m \leq n$, the number of existing nodes in the graph to which each new node must attach.

3. Stochastic Block Model (Planted Partition Model)

   This model generates graphs that contain communities or clusters, which are groups of nodes which are highly connected to each other. It takes the following parameters: n, the number of nodes in the graph, k the number of communities, a partition of the nodes into different communities, p, the probability that two nodes within the same community have an edge between them, and q, the probability that two noes in different communities have an edge between them.

2.1.2 Generative Models with Implicit Parametrization

Modern approaches to graph generation tend to use neural network architectures as a way to obtain greater control on the properties of the generated graphs. These architectures are trained on a set of input graphs which have the properties we desire in the output. The models will learn these properties implicitly from the training data and replicate them in the output graphs. One danger of these methods is overfitting, which is when the model duplicates the input graph instead of generalizing to produce a new instance that resembles it. However, each of these generative architectures takes steps to prevent overfitting and promote generalization. While the implicit generative models are far superior to the explicit ones at capturing the properties of some observed graphs, they generally require large training data sets in order to generate graphs successfully, which is not a requirement for the classical approaches.

1. NetGAN [2]

   This is a graph generative architecture based on a Wasserstein GAN that learns the features of an input graph using random walks on the graph. The generator of the graph learns to generate random walks that are very similar to those performed on the input, and the discriminator learns to distinguish between these two types of random walks. After training is complete, the generator produces a set of random walks that could plausibly be from the input graph, which are then used to construct an output graph. The edges in the output graph are added based on the frequency with which they appeared in this set of random walks. NetGAN is shown empirically to outperform many existing implicit and explicit generative models [2]. Thus, we chose this generative model as the basis for our proposed motif-targeted generative model. A detailed explanation of this architecture is given in Section 2.2.

2. GraphGAN [3]

   This is also a GAN-based architecture that uses the edge connectivity distributions of an input graphs to generate new samples. The connectivity distribution is defined as the probabilities with which pairs of nodes form edges. The generator aims to learn a connectivity distribution as close as possible to that in the input graph. The discriminator learns to distinguish between the true connectivity distribution and those produced by the generator. Once trained, the connectivity distribution output by the generator can be used to construct a new output graph by adding edges between nodes according to the distribution.

3. GraphRNN [4]

   This architecture uses recurrent neural networks (RNN), a deep learning architecture used to model sequential data. GraphRNN generates an output graph as a sequence of nodes and edges. Each new node is added based on a probability distribution learned from already existing nodes by a “graph-level” RNN. Once a new node is added, it is connected to existing nodes based on a distribution learned from already existing edges by an “edge-level” RNN.

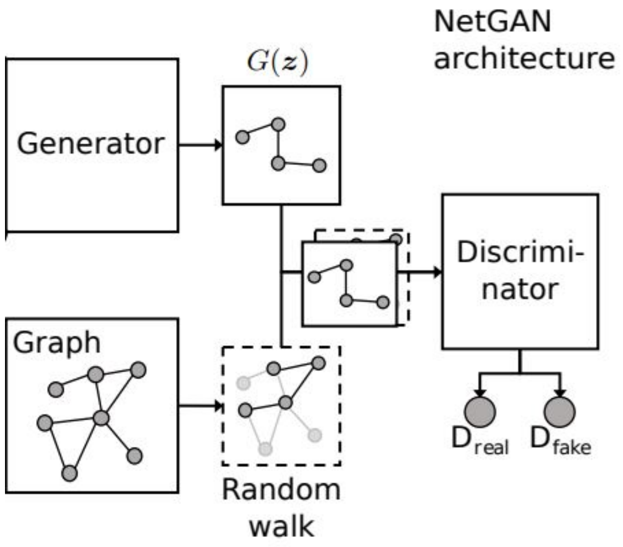


Figure 2.1: *The basic NetGAN architecture, adapted from [2].*

2.2 NetGAN

We base our motif-targeted generative model on an existing implicit graph generative architecture, NetGAN [2]. NetGAN is a generative adversarial network that uses random walks on a graph to generate realistic graphs that are statistically similar to a training graph. NetGAN consists of a generator G and discriminator D which are trained under the Wasserstein GAN objective [14] for increased stability. The generator G outputs sets of random walks that are similar to those sampled from an input graph that one wants to generate, while D learns to distinguish between random walks generated by G and those sampled from the input graph. Thus, NetGAN requires only one undirected graph as an input, from which it samples a set of random walks to act as a training data set. It is highly efficient for cases where one does not have a large set of similar graphs that can serve as the training set. Figure 2.1 depicts the basic NetGAN architecture.

Once G and D are trained, NetGAN generates a new graph as follows. It first generates a set of random walks using G. Then it constructs a score matrix S whose (i, j)th entry represents the number of times edge (i, j) appears in the generated set of random walks. The score matrix is normalized by the row sums so that for every node, one obtains a probability distribution over its neighboring nodes. To add an edge, a node is selected randomly and its neighbor is sampled according to the corresponding probability distribution constructed from the normalized score matrix. Subsequently, an edge between these two nodes is added in the output graph. The procedure continues until reaching the number of edges in the input graph.

NetGAN has been shown to outperform state-of-the-art graph generative models at preserving various topological features of the input graph (e.g. maximum degree, clustering coefficient, inter- and intra-community edge density) in its generated output. The method also exhibits competitive performance at link prediction on incomplete graphs, which indicates that it is capable of generalization, not just memorizing the input graph. Despite the efficacy of NetGAN in the above-mentioned tasks, we observe that the graphs generated by NetGAN (as well as other state-of-the-art generative models comparable to NetGAN) fail to approximate the network motif statistics of the input graph. For example, NetGAN systematically underestimates the number of triangles in social networks by 40-60% [2]. This is a major shortcoming for applications that aim to generate graphs that realistically mimic real-world networks or predict unobserved motif structures.

CHAPTER 3

DESIGN AND METHODOLOGY

The motif-targeted graph generative model proposed in this work, MMGAN, uses multiple techniques for learning on graphs and combines them into a motif-aware model. There exist many methods for link prediction in networks [15], but to the best of our knowledge, MMGAN is the only GAN-based generative and predictive model for *motifs*.

The proposed algorithm is a generalization of NetGAN, the GAN-based generative model which uses random walks to learn the structure of an input graph and predict missing links (pairwise edges) in the input. GANs are highly effective at learning implicit features of a data set and using these to generate realistic data samples. They are therefore a natural choice for both prediction tasks on incomplete data and sample generation. Conducting random walks on a graph is a technique widely used to learn the local and global topology of a graph [16–18]. Biased random walks in particular are used to characterize higher-order network structures like hyperedges and network motifs [8,19–23]. MMGAN extends NetGAN with the judicious use of biased random walks on a graph to learn higher-order motif structures and not only pairwise edges. With this approach, MMGAN achieves superior motif and link prediction compared to NetGAN as well as higher statistical resemblance to the input network in its output graphs.

3.1 Proposed Architecture

In this method we are particularly interested in predicting network motifs consisting of only three nodes, of which there are two types which we call V (two edges sharing a vertex) and T (three edges in a triangle) as pictured in Figure 3.1. We choose this size of network motif since it is a tractable size to begin extending existing graph generative methods and since they are signif-

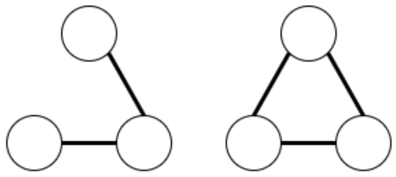


Figure 3.1: *Two types of three-node motifs targeted by MMGAN: V (pairs of edges sharing a vertex, pictured left) and T (triangles, pictured right).*

icant in social networks, which are a readily available and generally accurate source of data on which to test our architecture. Also, three-node motifs are likely to be contained in other higher-order motifs; this allows one to implicitly include information about higher-order interactions while limiting the complexity of the MMGAN architecture. However, it is straightforward to adapt MMGAN to account for motifs of larger sizes with adjustments in the weight calculation and score combination procedures outlined in Sections 3.1.2 and 3.1.3.

3.1.1 Overview

MMGAN consists of three GANs based on the basic NetGAN architecture, as shown in Figure 3.2:

1. (G_1, D_1): NetGAN with no modifications
2. (G_2, D_2): NetGAN biased towards predicting motif type V
3. (G_3, D_3): NetGAN biased towards predicting motif type T

Suppose we wish to generate graph instances that resemble some network of interest consisting of n nodes. We represent this network by an adjacency matrix $A \in \mathbb{R}^{n \times n}$ which shows the edge connectivity of the network. We assume that there are some missing edges (also known as links) and three-node motifs in the input network which we will try to predict using the graph output by the algorithm.

Each GAN (G_i, D_i) is first trained using the input graph A. A set of random walks are performed on A, which will be our training set. Using this training set, each generator G_i learns to produce random walks that resemble those in the training set, and each discriminator D_i learns to distinguish

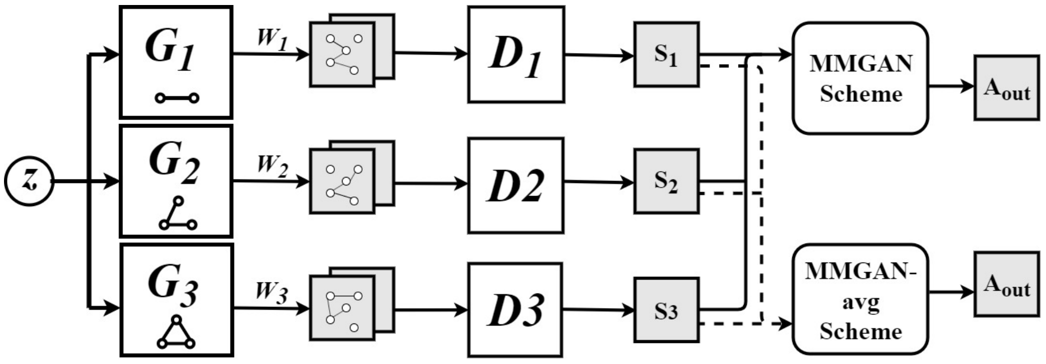


Figure 3.2: *The MMGAN architecture, consisting of NetGAN (G_1, D_1), and the two motif-biased GANs (G_2, D_2) and (G_3, D_3). Each G_i produces a set of random walks, while each D_i determines which are plausibly coming from the input graph and generates a score matrix. The score matrices are combined under two different schemes to obtain the output graph.*

between random walks coming from the training set and those generated by G_i. Note that the random walks are dependent on weights assigned to each edge, which are recorded in weight matrices $W_i \in \mathbb{R}^{n \times n}$. The three weight matrices are configured differently to accomplish different motif-biasing, as explained in Section 3.1.2.

Once this adversarial training is complete, each G_i can produce random walks that are very similar to those in the training set from a random noise sample z. A set of such random walks is produced by each GAN, which is then used to construct the score matrix S_i. This is done using the same approach as NetGAN: in the score for edge (j, k), recorded in $(S_i)_{jk}$, is the number of times that the edge appears in the set of random walks generated by the GAN (G_i, D_i). The three score matrices are then combined in two different ways to obtain optimal graph generation and link/motif prediction, and the combined score matrix is then used to construct the output graph A_{out} as explained in Section 3.1.3.

3.1.2 Motif-biasing through Random Walks

The three different GANs in MMGAN are optimized to retain the statistics of different types of edge structures in the graph. (G_1, D_1), which is plain NetGAN, is optimized to retain the pairwise edge connectivity patterns in the graph. (G_2, D_2) is optimized to retain information about the three-node motif type V, while (G_3, D_3) is optimized for motif type V, the triangles. This motif-targeted optimization in the latter two is done by biasing the random walks conducted by each GAN to run more often across edges that participate more in the type of motif targeted by the GAN. Thus, these edges will show up more often in the generated set of random walks, resulting in a higher edge score in the corresponding score matrix, which increases the likelihood of the edge being present in the output graph A_{out}.

To generate the training set of random walks, NetGAN employs a second-order random walk, node2vec, which captures the local and global structure of the graph effectively via a two-step weighting scheme [17]: given an edge (v, x), suppose that the previous transition of the random walk was from some node t to v. The second order bias α is chosen as

$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0, \\ 1 & \text{if } d_{tx} = 1, \\ \frac{1}{q} & \text{if } d_{tx} = 2, \end{cases}$$

where $p, q \in \mathbb{R}$ and d_{tx} is the shortest-length path between t and x. The unnormalized transition probability from v to x equals

$$\pi_{vx} = \alpha_{pq}(t, x) \cdot w_{vx},$$

where w_{vx} is the weight of edge (v, x) (equal to 1 for unweighted graphs). In MMGAN, we change this weight to incorporate the three-node motif statistics of the graph and bias the random walk towards edges that are more likely to be part of a particular network motif. This bias is different from the bias α_{pq} introduced to control the extent of exploration in the graph.

To find the correct biases, we first count the motifs in the graph of interest. While a complete enumeration of the motifs present in large-scale network is computationally prohibitive, a number of efficient motif-sampling algorithms exist that approximate the frequencies of different motifs in a network [24,25]. In our analysis, we use *FANMOD*, a fast network motif detection algorithm that can handle both directed and undirected networks and finds motifs containing up to eight nodes [25].

Using *FANMOD*, we first estimate the total number M of three-node motifs which are of types V and T listed in Figure 3.1. We then calculate the edge weights as follows. Let the *concentration* of a motif be $C^{(X)} = M^{(X)}/M$, where $X \in \{V, T\}$ and $M^{(X)}$ denotes the number of motifs of type X in the graph. For an edge (i, j), we define $N_{ij}^{(X)}$ to be the number of motifs of type X in which the edge participates. Then, the *motif-biased weight* of the edge (i, j) equals

$$w_{ij} = \frac{\beta N_{ij}^{(V)} + (1-\beta) N_{ij}^{(T)}}{N_{ij}^{(V)} + N_{ij}^{(T)}},$$

where $\beta = \begin{cases} \max\left\{C^{(V)}, C^{(T)}\right\} & (\text{ biasing towards } V) \\ 1 - \max\left\{C^{(V)}, C^{(T)}\right\} & (\text{ biasing towards } T). \end{cases}$

Thus, w_{ij} is a weighted average of the motif counts, weighted by an appropriate function of the concentration. The chosen bias will lead to a higher frequency of the particular motif in the output graph compared to the input graph. In order to obtain motif counts that reflect the counts in the input, we combine the output score matrices of three GANs with random walks biased as follows: without using motif weights as in NetGAN (S_1), using weights that bias towards V (S_2), and using weights biased towards T (S_3). The matrix S_1 leads to a good characterization of the input edge set. From S_2, we obtain a better characterization of the V motifs in the graphs when compared to S_1, but with a frequency that is higher. Similarly, S_3 ensures a good characterization of triangles, albeit once again with a higher count than observed in the input. These three 'views' of the graph provide a close approximation of the actual motif frequencies and concentrations once properly combined.

3.1.3 Combining Score Matrices to Generate A_{out}

To handle different tasks such as motif generation and link prediction, we propose two different ways of combining the score matrices:

I. Multi-view combination for link prediction (MMGAN-Avg)

We combine the three score matrices via averaging, resulting in $S = \frac{S_1+S_2+S_3}{3}$. Edges are sampled in the same manner as in NetGAN by first normalizing S to produce a transition probability matrix, then selecting a node at random and choosing one of its neighbors according to the above distribution. We add an edge between the corresponding nodes in the output graph and continue adding edges similarly until the same number of edges as in the input graph is reached.

II. **Multi-view combination for graph generation (MMGAN)**

In this scheme, we sample both edges and motifs from the three views at random and add them to the output graph directly as follows. We first randomly choose one view S_i of S_1, S_2, S_3 with probabilities p_1, p_2, p_3 respectively. Then, we choose one of two sampling methods, sampling by maximum score or random sampling, with probabilities p_s and $1 - p_s$ respectively, where p_s is small to avoid overfitting.

If we choose sampling by maximum score, we first select the edge e_i with the highest score in S_i. Then, we add the corresponding subgraph structure to the output graph. In more detail, if $S_i = S_1$ we add e_i to the output graph. If $S_i = S_2$, we find all possible V motifs containing e_i and compute the average score for each possible motif. Then, we select the motif with the highest average score and add the two edges of the motif to the output graph. Similarly, if $S_i = S_3$, we compute the average scores for all T motifs (triangles) containing e_i and add the three edges of the highest scoring motif to the output. After adding the edge(s), we remove the corresponding scores from S_i to enforce sampling without replacement. We repeat this procedure with the next highest score in the score matrix and continue until the output graph has the same number of edges as the input.

If we choose random sampling, we first select a node n_i uniformly at random. Then, similar to the previous combination method, we randomly sample two other nodes n_2, n_3 with the probability distribution defined by the normalized score matrix. Finally, if $S_i = S_1$, we add edge (n_1, n_2) to the output graph. If $S_i = S_2$, we add the V motif with edges $\{(n_1, n_2), (n_1, n_3)\}$, and if $S_i = S_3$, we add the T triangle with all three nodes. We continue this until the output contains the same number of edges as the input.

Choosing some of the maximum scoring edges and motifs ensures that the key edges that appear repeatedly in the sampled set of random walks are included in the output graph. The repeated appearances indicate that the edge has a high weight and therefore is a part of a larger number of motifs. Every time we sample from these heavily weighted edges, we add an entire motif to the output graph. Thus, adding a small sample of these will lead to a higher frequency of motifs in the output. Furthermore, by adjusting p_1, p_2, p_3, we can control the frequency of the different motif types as needed. This approach leads to a closer approximation of the motif counts in the original graph compared to MMGAN-avg, at the potential expense of link and motif prediction accuracy.

3.2 Extension to Hypergraphs

The MMGAN architecture described above is designed for motif prediction on networks represented as graphs with pairwise interactions, modeled by the adjacency matrix $A \in \mathbb{R}^{n \times n}$. However, some real-world networks are best represented as *hypergraphs*, which consist of nodes and *hyperedges*, i.e. edges between more than just two nodes. A network motif with k nodes can be considered a size-k hyperedge. For networks represented in this way, we can no longer use the simple random walk, motif-biased or otherwise, used in NetGAN and MMGAN since the hyperedges do not give us information about pairwise connections between nodes.

For networks represented as hypergraphs, we introduce an extension of our architecture called MMGAN-H. In this extension, we use the same principle as in MMGAN of combining the output of three different GANs with the goal of predicting hyperedges of different sizes, which can be thought of as network motifs. We are particularly interested in predicting missing hyperedges of size two and size three as in the case with MMGAN, where we predicted missing edges and three-node motifs (note that hyperedges of size two are essentially edges). Since hyperedges do not allow a differentiation between those of the same size (like V and T in the case of three-node motifs), we bias the GANs towards hyperedges of size two and three instead.

The input to the MMGAN-H architecture is a hypergraph with n nodes and m hyperedges of sizes two and above. We introduce a *hyperedge-biased random walk* in order for the generator to learn the features of the input hypergraph. This random walk transitions first between two hyperedges and then between nodes within the two hyperedges according to probabilities calculated based on the hyperedge statistics of the graph.

The hyperedge transitions probabilities are calculated based on the number of incident hyperedges (hyperedges that share nodes with the current hyperedge we are in within the random walk) of different types using a method known as *additive smoothing*. This accounts for any missing hyperedges in the graph, since the input graph in incomplete. The next node to transition to is chosen depending on a *hyperedge-biased weight* computed as follows. Suppose the two nodes in question are $i, j \in \mathcal{V}$. Let $C^{(x)}$ be the percentage of size-x hyperedges in the hypergraph. Then the hyperedge-biased weight for transitioning between them is:

$$w_{ij} = \frac{\beta N_{ij}^{(2)} + (1-\beta) N_{ij}^{(3)}}{N_{ij}^{(2)} + N_{ij}^{(3)}},$$

$$\text{where } \beta = \begin{cases} \max\left\{C^{(2)}, C^{(3)}\right\} & (\text{ biasing towards hyperedges of size 2}) \\ 1 - \max\left\{C^{(2)}, C^{(3)}\right\} & (\text{ biasing towards hyperedges of size 3}). \end{cases}$$

Note that this is almost identical to the biasing scheme in MMGAN, with the two three-node motif types replaced by two differently sized hyperedges. The full algorithm for the hyperedge-biased random walk is described below. Let L be the length of the random walk:

1. Initialize a starting hyperedge h_0 uniformly at random.
2. Initialize a starting node n_0 from the nodes in h_0 uniformly at random.
3. For $k = 0, \ldots, L-1:$
   (a) Find all hyperedges incident to (sharing one or more nodes with) h_k and split into two categories:
   H_1 = incident hyperedges with only one overlapping node with h_k
   H_2 = incident hyperedges with two or more overlapping nodes with h_k
   (b) Compute the following hyperedge counts:
   N_1 = number of hyperedges in H_1
   N_2 = number of hyperedges in H_2
   (c) Compute probability p of choosing the next hyperedge from H_1 using additive smoothing with $\alpha = 1$ and $d = 2$:

$$p = \frac{N_1 + \alpha}{N_1 + N_2 + d\alpha}$$

(d) Choose to pick the next hyperedge h_{k+1} from H_1 with probability p and from H_2 with probability $(1 - p)$

(e) Once either H_1 or H_2 is selected, choose a hyperedge h_{k+1} from it uniformly at random.

(f) Choose the next node n_{k+1} from h_{k+1} based on the hyperedge-biased weights as follows. Suppose the next hyperedge h_{k+1} has nodes $(m_1, \ldots, m_t)$. Then choose $n_{k+1} = m_i, i \in \{1, \ldots, t\}$ with the probability:

$$q_i = \frac{w_{n_k m_i}}{\sum_{j=1}^{t} w_{n_k m_j}}$$

Given an incomplete hypergraph as input, MMGAN-H combines the output score matrices of three GANs as with MMGAN. The first is NetGAN with no hyperedge biasing. Since the input is a hypergraph, we first convert it to a clique expansion graph which consists on only pairwise edges and use this as the input to NetGAN. Suppose S_1 is the output score matrix from this method. The score matrix S_2 is obtained by running the GAN with the above described hyperedge-biased random walk, biased towards size-two hyperedges. S_3 is obtained by doing the same but biased towards size-three hyperedges instead. The combined output score matrix is obtained by averaging the three score matrices as in MMGAN-avg, so $S = (S_1 + S_2 + S_3)/3$. The output graph A_{out} is obtained from S similarly to MMGAN-avg.

CHAPTER 4

EXPERIMENTAL FINDINGS

4.1 Testing on Social Networks

We test the performance of MMGAN against NetGAN, which is shown to outperform a number of other benchmark graph generative models in terms of preserving the input graph statistics [2]. For data, we use three real-world social networks, Cora [26], Citeseer [27], and Facebook [28] with the characteristics described in Table 4.1. Note that in all of these networks, triangles (T) are statistically significant (occur with higher frequency in the real network compared to randomized networks). Thus, we are generally interested in keeping a comparable triangle count to the input network in our generated output.

Table 4.1: *Statistics of the real-world network used for testing.* |N| *and* |E| *are the number of nodes and edges in the largest connected component of the graph respectively.* $C^{(V)}$ *and* $C^{(T)}$ *represent the concentration of each motif (proportion of motifs of each type in the total set of three-node motifs).* $R^{(V)}$ *and* $R^{(T)}$ *show the average concentration of each motif type in a set of graphs drawn from the random graph model* $\mathcal{G}_r$.

Network	|N|	|E|	$C^{(V)}$	$C^{(T)}$	$R^{(V)}$	$R^{(T)}$
Cora	2485	10,138	96.81	3.19	99.97	0.03
Citeseer	2118	7358	95.45	4.55	99.94	0.06
Facebook	1034	53,498	74.66	25.34	96.68	3.32

In each of the experiments described, we train NetGAN, MMGAN, and MMGAN-Avg to 60% edge overlap (one of the methods of early stopping in NetGAN) and average results over five runs. We use an 80-20% training and testing split of the total three-node motifs in the original graph.

Table 4.2: *Motif counts in the generated graphs, normalized with respect to input count for easier comparison. We use the dark shade to denote the best result (least error) over all methods and the light shade for any our methods that outperforms NetGAN.*

Dataset	Motif	Normalized Motif Count		
		NetGAN	MMGAN	MMGAN-avg
Citeseer	V	0.8069 (-0.1931)	1.0227 ($+0.0227$)	0.7672 (-0.2328)
	T	0.5830 (-0.4170)	1.1854 ($+0.1854$)	0.6661 (-0.3339)
Cora	V	0.8340 (-0.1660)	1.2426 ($+0.2426$)	0.7546(-0.2454)
	T	0.5110 (-0.4890)	1.1675 ($+0.1675$)	0.6457(-0.3543)
Facebook	V	1.0803 ($+0.0803$)	0.9723(-0.0277)	1.0735($+0.0735$)
	T	0.5557 (-0.4443)	0.4011(-0.5989)	0.5618(-0.4382)

4.1.1 Motif-targeted Graph Generation

We evaluate the ability of MMGAN and MMGAN-Avg to preserve the motif structures in the graph by comparing motif counts and motif concentrations in the output. For this, we combine the multiple score matrices using the combination schemes described in I and II. For II, we set $p_1 = \frac{1}{6}$, $p_2 = \frac{1}{3}$ and $p_3 = \frac{1}{2}$, emphasizing triangles, and $p_s = 0.25$ for every experiment. The choice of the probabilities is governed by the number of edges in the motifs being 1, 2 and 3, respectively. The results for both combination schemes I and II are shown for comparison in Tables 4.2 and 4.3.

4.1.2 Link and Motif Prediction

We evaluate the predictive ability of the MMGAN and MMGAN-Avg as follows. For motif prediction, we use the test set of motifs held out during training and construct an equally sized set of test non-motifs. For link prediction, we use the corresponding edges as test edges and non-edges. For each motif and non-motif in the test set, we assign a motif score, which is the average of the scores of the nodes involved in the motif. For link prediction, we use the edges corresponding to the motifs in the test motifs individually and use the score assigned to each edge in the score matrix output by each algorithm.

We use the average scores of these test motifs and edges to compute two

Table 4.3: *Motif distributions in generated graphs and comparison using Küllback-Leibler divergence with respect to the input distribution.*

Dataset	Motif	Motif Concentration			
		Input	NetGAN	MMGAN	MMGAN-avg
Citeseer	V	95.45 %	96.68%	94.75%	96.03%
	T	4.55%	3.32%	5.25%	3.97%
		KL	0.2764	0.0777	0.0583
Cora	V	96.81%	98.02%	97.00%	97.25%
	T	3.19%	1.98%	3.00%	2.75%
		KL	0.3942	0.0086	0.0474
Facebook	V	74.66%	85.14%	87.72%	84.92%
	T	25.34%	14.86%	12.28%	15.08%
		KL	4.6922	7.5672	4.4839

Table 4.4: *Link and motif prediction quality measured using area under the ROC curve (AUC).*

Dataset	Type	NetGAN	MMGAN	MMGAN-avg
Citeseer	Link	0.9599	0.9265	0.9675
	Motif	0.9974	0.9958	0.9982
Cora	Link	0.9159	0.8947	0.9340
	Motif	0.9961	0.9907	0.9977
Facebook	Link	0.9779	0.9751	0.9981
	Motif	0.9733	0.9585	0.9770

metrics: AUC (area under the curve of the receiver operating characteristic) and AP (average precision), which are standard metrics for link prediction evaluation [2]. Tables 4.4 and 4.5 show the results under each metric.

4.1.3 Discussion

While all three algorithms are quite successful, in motif prediction MMGAN-Avg outperforms all the other methods in every data set under all metrics and should be the method of choice for motif prediction. The two different GAN-combining schemes essentially trade off between exploration and exploitation in different manners. MMGAN targets edges that are more likely to produce motifs and adds them to the output, thus ensuring that we obtain close to

Table 4.5: *Link and motif prediction quality measured using average precision (AP).*

Dataset	Type	NetGAN	MMGAN	MMGAN-avg
Citeseer	Link	0.9655	0.9391	0.9730
	Motif	0.9962	0.9950	0.9970
Cora	Link	0.9223	0.9010	0.9429
	Motif	0.9959	0.9902	0.9969
Facebook	Link	0.9735	0.9743	0.9816
	Motif	0.9578	0.9337	0.9632

the input counts. MMGAN-Avg on the other hand incorporates information from all three views equally, resulting in a graph that better reflects the edge connectivity of the input network. Nevertheless, it appears plausible that large-scale tuning of the motif sampling probabilities and the proportions of the maximum and random score selection in MMGAN may lead to improved performance compared to MMGAN-Avg.

We further note that even without explicitly incorporating statistics of four-node motifs in the input network, MMGAN approximates their counts better than NetGAN. For example, we compare the square (four-node cycle) counts in the output when they were trained on Citeseer. NetGAN generates graphs that have a normalized count of 0.1204 on average, while MMGAN has a normalized count of 0.3012 on average in its output graphs. This supports our assumption that since three-node motifs are likely to be contained in other higher-order motifs, using only the three-node motif statistics still allows us to implicitly include information about the higher-order motifs.

While MotifGAN-avg has relatively poor performance compared to MotifGAN in terms of motif counts, it is still competitive and could be a good alternative to MotifGAN if maintaining a balance between the exploration and exploitation is important for the target application.

4.2 Testing on Biological Networks

We test the extension of MMGAN to hypergraphs, MMGAN-H, against NetGAN on a genomic hypergraph data set known as Chia-Drop [29]. This data

Table 4.6: *Distribution of hyperedge sizes in BioChr4.*

Hyperedge Size	2	3	4	5	6	7	8	9	11	14
Number of hyperedges	652	168	71	16	16	9	4	1	2	1

set captures how the genomes of multicellular organisms fold into themselves in 3D space in a cell. The folding results in different regions of a chromosome interacting with each other, known as a chromatin interaction. If each of these regions is represented by a node, the multi-way interactions between the regions can be represented as hyperedges. Since only partial information about such interactions is available, we would like to infer the missing interactions by converting this into a missing hyperedge prediction problem.

The data set used for our experiment, which we will call BioChr4, captures the chromatin interactions within the fourth chromosome of *Drosophila* (fruit flies). The resulting hypergraph consists of $n = 818$ nodes and $m = 940$ hyperedges with the hyperedge size distribution given in Table 4.6.

We test the performance of three algorithms. First we test NetGAN with the clique expansion graph of the BioChr4 hypergraph as input. Next we test MMGAN with the clique expansion graph as input, but with the motif-biasing replaced with the hyperedge-biased weighting scheme described in Section 3.2, biasing towards hyperedges of size two and three. Finally we test MMGAN-H with the hypergraph as input directly. We test the prediction of missing hyperedge of sizes two to four under the same metrics as before. The results are given in Table 4.7.

Table 4.7: *Prediction results on test hyperedges of different sizes under two metrics, AUC (area under ROC curve) and AP (average precision).*

Metric	Algorithm	Hyperedge Size		
		2	3	4
AUC	NetGAN	0.7860	1.0000	1.0000
	MMGAN	0.8389	1.0000	1.0000
	MMGAN-H	1.0000	1.0000	1.0000
AP	NetGAN	0.7733	1.0000	1.0000
	MMGAN	0.8158	1.0000	1.0000
	MMGAN-H	1.0000	1.0000	1.0000

4.2.1 Discussion

We note that both MMGAN and MMGAN-H outperform NetGAN at predicting size-two hyperedges (edges) under both metrics. MMGAN-H particularly achieves perfect prediction across the board for all hyperedge sizes. The perfect prediction in the case of the size three and four hyperedges for all algorithms is likely due to the small number of test hyperedges used (16 and 11 respectively) compared to the test hyperedges of size two (30). While MMGAN by itself is sufficient to get good prediction on the BioChr4 dataset, MMGAN-H could be used for much better prediction. For a more detailed comparison, future work will include testing on the other five chromosomes in the Chia-Drop dataset which are much larger compared to chromosome four used here.

CHAPTER 5

CONCLUSION

This thesis examined the use of graph generative models for predicting higher-order network structures in graph. Graph generative models have long been studied, with many classical and modern approaches developed to generate new graph instances with certain desired properties. Modern graph generative models based on neural network architectures implicitly learn the characteristics of an input graph and replicate them in the output, with sufficient generalization to ensure that the graphs are not duplicated. This is a great advantage over classical approaches, which rely on explicit parameters input by the user to determine the characteristics of the graph, making it difficult to replicate real-world networks whose properties are not always well understood.

One such property is the presence of higher-order connectivity patterns such as network motifs. Most existing generative methods excel at retaining the pairwise connectivity patterns of the original graph in its output but do not account for motifs, hyperedges, and similar higher-order structures. Thus, such architectures can be used successfully to predict missing pairwise links in the input graph, but they do not predict higher-order structures well. Network motifs in particular contain valuable information about fundamental functional properties of networks, and they have been observed over a wide range of real-world networks. Thus, in order to generate realistic real-world graphs and to use them for predicting such structures, the graph generative architectures must account for network motifs.

The architecture proposed in this work, MMGAN, addresses this issue and we show empirically that it outperforms the current benchmark at both link and motif prediction. MMGAN generalizes the NetGAN generative model, which uses random walks on an input graph to learn its properties and replicate them in the output. MMGAN extends this architecture to predicting network motifs by using motif-biased random walks to account for different three-node motif types present in the graph. We show its efficacy at both retaining the motif statistics of the original graph and predicting missing links and motifs by testing on real-world social network data. The MMGAN architecture, while designed with only three-node network motifs in mind, can be easily extended to predict higher-order network motifs as well.

We also introduce an extension to MMGAN for hyperedges named MMGAN-H, and test its performance at predicting small hyperedges of size up to four (which can be thought of as network motifs) on a real-world biological data set consisting of hypergraphs. We show that both MMGAN-H and MMGAN with some modifications outperform NetGAN on this hypergraph data set. Future work will include more comprehensive testing on larger chromosomal data sets and extensions to larger motifs as well as hyperedges.

REFERENCES

[1] D. Easley and J. Kleinberg, *Networks, crowds, and markets: Reasoning about a highly connected world.* New York, NY, USA: Cambridge University Press, 2010.

[2] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, "NetGAN: Generating graphs via random walks," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018. [Online]. Available: http://proceedings.mlr.press/v80/bojchevski18a.html pp. 610–619.

[3] 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 Conference on Artificial Intelligence (AAAI-18)*, 2018. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16611/15969

[4] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec, "GraphRNN: Generating realistic graphs with deep auto-regressive models," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018. [Online]. Available: http://proceedings.mlr.press/v80/you18a.html pp. 5708–5717.

[5] D. Chakrabarti and C. Faloutsos, "Graph mining: Laws, generators, and algorithms," *ACM Computing Surveys (CSUR)*, vol. 38, no. 1, p. 2, 2006.

[6] X. Yu, X. Ren, Y. Sun, Q. Gu, B. Sturt, U. Khandelwal, B. Norick, and J. Han, "Personalized entity recommendation: A heterogeneous information network approach," in *Proceedings of the 7th ACM International Conference on Web Search and Data Mining.* ACM, 2014, pp. 283–292.

[7] S. Gao, L. Denoyer, and P. Gallinari, "Temporal link prediction by integrating content and structure information," in *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*. ACM, 2011, pp. 1169–1174.

[8] D. Zhou, J. Huang, and B. Schölkopf, "Learning with hypergraphs: Clustering, classification, and embedding," in *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. C. Platt, and T. Hoffman, Eds. MIT Press, 2007, pp. 1601–1608. [Online]. Available: http://papers.nips.cc/paper/3128-learning-with-hypergraphs-clustering-classification-and-embedding.pdf

[9] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: Simple building blocks of complex networks," *Science*, vol. 298, no. 5594, pp. 824–827, 2002. [Online]. Available: https://science.sciencemag.org/content/298/5594/824

[10] S. Shen-Orr, R. Milo, S. Mangan, and U. Alon, "Network motifs in the transcriptional regulation network of escherichia coli," *Nature Genetics*, vol. 31, no. 1, pp. 64–68, 2002. [Online]. Available: https://doi.org/10.1038/ng881

[11] J. Ugander, L. Backstrom, and J. Kleinberg, "Subgraph frequencies: Mapping the empirical and extremal geography of large graph collections," in *Proceedings of the 22nd International Conference on World Wide Web*, ser. WWW '13. New York, NY, USA: ACM, 2013. [Online]. Available: http://doi.acm.org/10.1145/2488388.2488502 pp. 1307–1318.

[12] A. K. Dey, Y. R. Gel, and H. V. Poor, "Motif-based analysis of power grid robustness under attacks," in *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, Nov 2017, pp. 1015–1019.

[13] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'14. Cambridge, MA, USA: MIT Press, 2014, p. 2672–2680.

[14] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017. [Online]. Available: http://proceedings.mlr.press/v70/arjovsky17a.html pp. 214–223.

[15] D. Liben-Nowell and J. Kleinberg, “The link-prediction problem for social networks,” *Journal of the American Society for Information Science and Technology*, vol. 58, no. 7, pp. 1019–1031, 2007.

[16] B. Perozzi, R. Al-Rfou, and S. Skiena, “DeepWalk: Online learning of social representations,” in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’14. New York, NY, USA: ACM, 2014. [Online]. Available: http://doi.acm.org/10.1145/2623330.2623732 pp. 701–710.

[17] A. Grover and J. Leskovec, “Node2Vec: Scalable feature learning for networks,” in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’16. New York, NY, USA: ACM, 2016. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939754 pp. 855–864.

[18] P. Li, I. Chien, and O. Milenkovic, “Optimizing generalized PageRank methods for seed-expansion community detection,” *CoRR*, vol. abs/1905.10881, 2019. [Online]. Available: http://arxiv.org/abs/1905.10881

[19] J. Lee, M. Cho, and K. M. Lee, “Hyper-graph matching via reweighted random walks,” in *CVPR 2011*, June 2011, pp. 1633–1640.

[20] C. E. Tsourakakis, J. Pachocki, and M. Mitzenmacher, “Scalable motif-aware graph clustering,” in *Proceedings of the 26th International Conference on World Wide Web*, ser. WWW ’17. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2017. [Online]. Available: https://doi.org/10.1145/3038912.3052653 pp. 1451–1460.

[21] L. Backstrom and J. Kleinberg, “Network bucket testing,” in *Proceedings of the 20th International Conference on World Wide Web*, ser. WWW ’11. New York, NY, USA: ACM, 2011. [Online]. Available: http://doi.acm.org/10.1145/1963405.1963492 pp. 615–624.

[22] G. Han and H. Sethu, “Waddling random walk: Fast and accurate mining of motif statistics in large graphs,” in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, Dec 2016, pp. 181–190.

[23] M. El Dayeh and M. Hahsler, “Biological pathway completion using network motifs and random walks on graphs,” in *2012 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, May 2012, pp. 229–236.

[24] A. Masoudi-Nejad, F. Schreiber, and Z. R. M. Kashani, “Building blocks of biological networks: A review on major network motif discovery algorithms,” *IET Systems Biology*, vol. 6, no. 5, pp. 164–174, Oct 2012.

[25] S. Wernicke and F. Rasche, "FANMOD: A tool for fast network motif detection," *Bioinformatics*, vol. 22, no. 9, pp. 1152–1153, May 2006. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/btl038

[26] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[27] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, "Automating the construction of internet portals with machine learning," *Information Retrieval*, vol. 3, no. 2, pp. 127–163, 2000.

[28] J. Leskovec and J. Mcauley, "Learning to discover social circles in ego networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 539–547.

[29] M. Zheng, S. Tian, D. Capurso, M. Kim, R. Maurya, B. Lee, E. Piecuch, L. Gong, J. Zhu, Z. Li, C.-H. Wong, C. Ngan, P. Wang, X. Ruan, L. Chia Wei, and Y. Ruan, "Multiplex chromatin interactions with single-molecule precision," *Nature*, vol. 566, 02 2019.