FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Exploration of Graph Generation Techniques
Student:	Kirill Poligach
Supervisor:	Ing. Miroslav Čepek, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2022/2023

Instructions

Graphs are an important, powerful and common data model. Due to various reasons - like privacy or security concerns - it's not always possible to directly work with an original graph. The thesis aims to investigate graph-generating techniques so the analyst can fingerprint the original graph and generate a synthetic graph which preserves the statistical properties and can be used as a basis for graph machine learning projects with results directly usable even in the original graph.

Review, explore and demonstrate machine learning techniques for generating graphs similar to [1],[2] on selected datasets.

Focus on generating larger graphs, like social networks or bank-transaction graphs, for example, some of the graphs from [3] or [4] and assess limitations on the size and variability of the graph. Evaluate selected technique(s) and resulting synthetic graphs. In evaluation focus graph (statistical) properties and the predictive power of machine learning models. Also, demonstrate visually the similarity between synthetic and original graphs. The machine learning model is created on top of the synthetic graph and applied on the original graph. Demonstrate whether the ML model can be applied directly or whether fine-tuning of the model would be beneficial.

[1] You, Jiaxuan, et al. "Graph convolutional policy network for goal-directed molecular graph generation." Advances in neural information processing systems 31 (2018). https://arxiv.org/pdf/1806.02473.pdf

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 3 July 2022 in Prague.

Bachelor's thesis

EXPLORATION OF GRAPH GENERATION TECHNIQUES

Kirill Poligach

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: Ing. Miroslav Čepek Ph.D.
January 5, 2023

Contents

List of Figures

I would like to thank my supervisor for the help and advice to the work and for the help with finding materials. I would also like to thank my beloved family and friends for their endless support during the writing of this work, despite the fact that you are far away, your support is priceless. I couldn't have done it without all of you.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Praze on January 5, 2023 . . . .

Abstract

Due to various reasons – like privacy or security concerns – it's not always possible to work directly with an original graph, such as bank transactions or social network interaction graph. These graphs are necessary for ML projects such as bank security systems for the detection of abnormal transactions, social network recommendation systems, and many other similar projects. This thesis aims to investigate the current existing graph generation techniques and assess how much synthetic graphs statistically correspond to the properties of the original ones. It also evaluates the feasibility of using structural embedding models for classification problems using synthetic graphs.

Keywords graph theory, machine learning, synthetic graphs, neural networks, synthetic data generation, artificial neural networks

Abstrakt

Z různých důvodů - například z důvodu ochrany soukromí nebo bezpečnosti - není vždy možné pracovat přímo s původním grafem, například s grafem bankovních transakcí nebo interakcí na sociálních sítích. Tyto grafy jsou nezbytné pro ML projekty, jako jsou bankovní bezpečnostní systémy pro detekci abnormálních transakcí, doporučovací systémy sociálních sítí a mnoho dalších podobných projektů. Cílem této práce je prozkoumat současné existující techniky generování grafů a posoudit, nakolik syntetické grafy statisticky odpovídají vlastnostem grafů původních. Posuzuje také možnost využití strukturních embeddingových modelů pro klasifikační problémy s využitím syntetických grafů.

Klíčová slova teorie grafů, strojové učení, syntetické grafy, neuronové sítě, generování syntetických dat, umělé neuronové sítě

List of acronyms

CV	cross validation
BFS	breadth-first search
GAN	generative adversarial network
GRU	gated recurrent unit
RNN	recurrent neural network
ML	machine learning
MLP	multilayer perceptron
RL	reinforcement learning
VAE	variational autoencoder

Introduction

Graphs are the way of representing data that is used extensively in the modern sciences, both in the natural and social sciences. Graphs are most commonly used in computer science, where they can be used to store and structure large amounts of data, such as the graph of people's interactions in a social network. But the use of this data is not always possible and there is a need to create synthetic graphs that can be used as the real graphs.

In this thesis, I am going to focus on the problem of generating synthetic graphs that will statistically match the properties of the original data.

In today's world, it's hard for us to imagine our lives without apps. Watching videos, ordering food, communicating, finding new acquaintances, it's all become an integral part of our lives. Whether it's a social network or an online shop, data needs to be structured to be moderated and used, and graphs are the best way to do this.

Referral algorithms, banking security systems, and targeted advertising are all merits of machine learning models, which require large amounts of data from user interactions to create. But this data is not always available for legal or ethical reasons, such as violating the anonymity of social media users or using the banking transaction history of bank customers.

To create a working machine learning model, whether it is a social network recommendation system or a bank security system alerting a suspicious transaction, you need statistically correct user data, but that data must not threaten user anonymity or security. The solution to this problem is synthetic graphs of this data, which will preserve the statistical properties of the original data.

In this thesis, I will focus on investigating already existing techniques for generating synthetic graphs, and experiment on a few of them. The main goal of the experiments is to analyze how well the synthetic graphs retain the properties of the original ones, and how suitable they are for use in ML projects. I will also demonstrate how visually similar the generated graphs are to the original ones and as side goals, I will consider the possibility of generating large graphs, as well as the feasibility of using structural embeddings for graph's node classification task.

Chapter 1

Introduction to graph theory

In this chapter I will introduce the necessary basics of graph theory. More detailed information on these definitions can be found in [1],[2].

1.1 Fundamentals

Graph is a pair $G = (V, E)$ of sets such that $E \subseteq [V]^2$; thus, the elements of E are 2-element subsets of V. The elements of V are the vertices (or nodes, or points) of the graph G, the elements of E are it's edges (or lines).

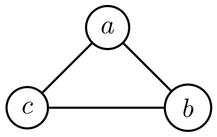


Figure 1.1 A graph.

If $V' \subseteq V$ and $E' \subseteq E$, then G' is a subgraph of G (and G a supergraph of G'), written as $G' \subseteq G$.

Directed graph is a pair (V, E) of disjoint sets (of vertices and edges) together with two maps init: $E \to V$ and ter: $E \to V$ assigning to every edge e an initial vertex and terminal vertex. The edge e is said to be directed from initial vertex to terminal vertex. Directed graphs can be used to demonstrate the directionality of relations between objects.

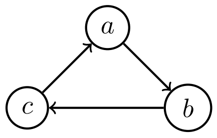


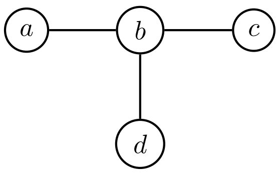
Figure 1.2 Directed graph, arrows are used to show the direction.

Path is a non-empty graph $P = (V, E)$ of the form

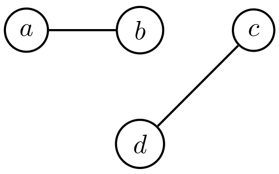
$$V = \{x_0, x_1, ..., x_k\} \; E = \{x_0x_1, x_1x_2, ..., x_{k-1}x_k\}$$

where the x_i are all distinct. On the figure 1.2 the path from a to c is $\{(a, b), (b, c)\}$. The number of edges in the path is it's length.

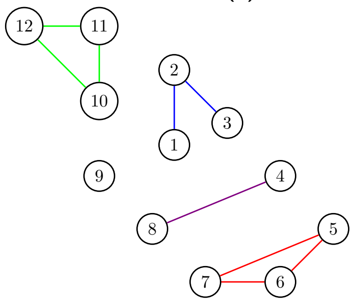
Graphs can also be viewed in terms of connectivity. A non-empty graph G is called **connected** (demonstrated on 1.3a) if any two of its vertices are linked by a path in G. If $U \subseteq V(G)$ and $G[U]$ is connected, U is also called itself connected (in G). If a graph is not connected, it is called **disconnected** (demonstrated on 1.3b).



(a) Connected graph.



(b) Disonnected graph.



(c) Graph with 5 components.

Figure 1.3 Graph connectivity

Let $G = (V, E)$ be a graph. A maximal connected subgraph of G is called a **component** (demonstrated on 1.3c) of G. Component, being connected, is always non-empty, therefore, if the graph is empty, then it cannot have a component.

1.2 Graph measures

Now I move on to the metrics (properties) that allow us to compare graphs with each other.

The degree (or valency) $d_G(v) = d(v)$ of a vertex v is the number $|E(v)|$ of edges at v. A vertex of degree 0 is isolated. The number

$$d(G) := \frac{1}{|V|} \sum_{v \in V} d(v)$$

is the average degree of G [1].

1.2.1 Shortest-path distance

The distance $d_G(x, y)$ between two vertices x, y in G is the lenght of a shortest path between these vertices in G, it's also called **geodesic distance** or **shortest-path distance**.
$d_G(x, y) = 0$ if $x = y$,
$d_G(x, y) = \infty$ if there is no path exists.

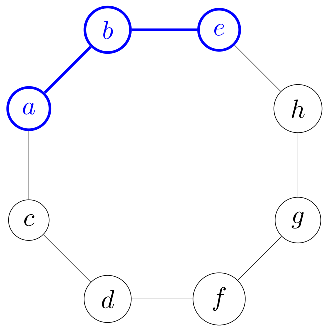


Figure 1.4 Shortest path between a and e is $\{(a,b),(b,e)\}$, so the geodesic distance is 2.

1.2.2 Centralities

Centrality is the concept of measuring the importance of a graph node. Centrality measure can be formulated as a function

$$\mathbf{c} : G(n) \to \mathbb{R}^n,$$

where $c_i(g)$ is the centrality of node i in graph g.

There are several types of centrality, each allows to assess the importance of a node in the graph, depending on a certain criterion.

For this thesis I will introduce the definitions of the following:

- Degree centrality
- Harmonic centrality
- Betweenness centrality

Degree centrality measures the number of edges of node $i, d_i(g)$. It can also be normalized by the maximal possible degree, $n-1$, to obtain a number between 0 and 1:

$$c_i^{deg}(g) = \frac{d_i(g)}{n-1}.$$

Degree centrality is an obvious measure, which describes 'popularity' of the node i, but does not give information about the other possibly informative aspects of the graph's architecture and node's position in the graph.

Harmonic centrality is an alternative measure of **closeness centrality**, which is based on the distance between a node and each other node in the graph. It extends degree centrality by considering the neighborhoods of all radii. As an input closeness centrality takes list of distances between node i and other nodes j in the graph, $\rho_g(i,j)$. There are different variations of closeness centrality based on the different functional forms. The measure proposed by [3] and [4], is based on distances between node i and all other nodes, $\sum_j \rho_g(i,j)$. The higher the score, the lower the centrality is, so there is an inversion and the distance between nodes belong to two different components becomes infinite. To solve this problem, another centrality measure was proposed by [4], it also can be normilized so that the highest possible centrality measure is equal to 1, to obtain the closeness centrality measure

$$c_i^{cls}(g) = \frac{n-1}{\sum_{j \neq i} \rho_g(i,j)}.$$

Harmonic centrality aggregates distances differently, it aggregates the sum of all inverses of distances $\sum_j \frac{1}{\rho_g(i,j)}$ ([5], [6]). So this centrality measure solves the problem with infinite or large distances between nodes. This measure also can be normalized:

$$c_i^{cl}(g) = \frac{1}{n-1} \sum_{j \neq i} \frac{1}{\rho_g(i,j)}.$$

Betweenness centrality describes the importance of a node in the graph, based on it's occurance in the connection of other nodes. From network's perspective, it describes the agent's (node's) role in the information transmissions between other agents (nodes) in the network (graph). The betweenness centrality measure proposed by [7] is

$$c_i^{bet}(g) = \frac{2}{(n-1)(n-2)} \sum_{(j,k), j \neq i, k \neq i} \frac{v_g(i:j,k)}{v_g(j,k)},$$

where $v_g(j,k)$ is geodesic (shortest-path distance) between two nodes j, k and $v_g(i:j,k)$ is the geodesic path between j,k passing through i.

Chapter 2

Machine Learning basics

The concept of machine learning is to create statistical models and algorithms that computer systems can use to perform tasks without direct instructions, relying on patterns derived from available data. In this chapter I will describe the basic concepts necessary to understand the concept of machine learning. The basic concepts in this chapter are taken from[8].

2.1 Data

One of the most important parts of machine learning is data. Data is considered as a collection of data points presented in a single format, where each data point has individual values describing the properties of a particular object. Data points can represent a variety of types of objects, depending on the application domain. Such a collection is commonly called a dataset, denoted by $\mathcal{D}$ and data point is denodetd by $\boldsymbol{z}$. We can represent dataset as $\mathcal{D} = \{\boldsymbol{z}^{(1)}, ..., \boldsymbol{z}^{(m)}\}$, where m is the number of data points in dataset. The object properties described by data point are low-level properties that can be easily calculated or measured, they are called features. Data points in dataset can be described by their feature vectors $\mathbf{x}$ with the same number n of individual features

$$\mathbf{x} = (x_1, ..., x_n)^T.$$

In addition to features, data points may have other properties representing higher-level facts related to the described object, called labels (or targer or output), usually are denoted by y or $\mathbf{y}$ if it is a vector of different values.

Label spaces can be divided into several categories. **Numeric** label space $\mathcal{Y}$ contains all possible y of data points. **Categorical** label space consists only of certain values, which are called classes or categories. **Ordinal** label space just like categorical contains finite set of values, but this set is ordered like a numeric.

Summing up, a data point can be represented as $\boldsymbol{z} = (\mathbf{x}, y)$. Label space $\mathcal{Y}$ is the set of all possible label values.

2.2 Model

The goal of ML model is to learn hypothesis map $h : \mathcal{X} \rightarrow \mathcal{Y}$ such that $y \approx h(\mathbf{x})$ for any data point, each characterized by features $\mathbf{x} \in \mathcal{X}$ and label $y \in \mathcal{Y}$. $h(\mathbf{x})$ commonly denoted by $\hat{y}$ and called as prediction. Hypothesis map h can also be called a predictor map since we use it to compute the prediction $\hat{y}$ of a (true) label y, or if a finite label space $\mathcal{Y}$ is used in the ML task (e..g, $\mathcal{Y} = \{-1, 1\}$), we refer to h also as a classifier.

So, we can conclude that the purpose of the model is to predict the result $\hat{y}$ based on the features of input data point $\mathbf{x}$ using the hypothesis map h.

2.3 Training

As already described above, for the model to work, it needs the hypothesis map h, which is a good predictor, i.e., predicted labels $\hat{y}$ are close to the true labels y (with small error $\hat{y} - y$). The process of finding the most accurate map is called training (or learning).

Based on how the models evaluate the quality of hypothesis maps, we can divide the learning into several types:

- Supervised learning.
- Unsupervised learning.
- Reinforcement learning.

Supervised and **unsupervised learning** differ in the presence of label (target) value in data points. Supervised ML model use training dataset with labeled data points (for which we know the correct label values). This model has a teacher (who labeled data points), therefore, during training, the model will try to find a hypothesis that will imitate the human annotator.

Unsupervised models do not need teacher who provides correct labels for data points, therefore, to find a good hypothesis map, they must rely only on the internal structure of data points in training dataset.

The main idea of **reinforcement learning** is the influence of the predictions obtained by the hypothesis on future data points generation. RL models involve data points that represents state of programmable system at different time instants. Therefore, data points labels represent an optimal action for the agent in a given state. As with unsupervised models, RL models are trained without access to any labeled data points.

2.4 Loss function

To evaluate the quality of the selected hypothesis, and compare them with each other, there is such a concept as loss function. **Loss function** is a map, which assigns a pair of a data point $((x), y)$ and a hypothesis h the non-negative real number

$$L : \mathcal{X} \times \mathcal{Y} \times \mathcal{H} \to \mathbb{R}_+ : (((x), y), h) \mapsto L(((x), y), h).$$

The loss value evaluates the divergence between the real (true) label value and the prediction. Small loss indicates low divergence.

The main principle of ML models can be formulated as:
Find hypothesis h out of a given space $\mathcal{H}$ that brings a minimum loss $L(((x), y), h)$ for any data point.

For an objective assessment of the accuracy of a trained model, usually the existing dataset is divided into training and test datasets, so a number of problems can be identified, such as an overfitting.

2.5 Overfitting

Overfitting is a phenomenon, when model find (learn) the best hypothesis map h for training dataset $\mathcal{D}_{training}$, which perfectly (almost) predicts the labels $\mathbf{y}$ of data points, but such a map might deliver much less accurate predictions $\hat{y}$ outside $\mathcal{D}_{training}$. This may be caused by too large hypothesis space $\mathcal{H}$, because the model can easily find such a hypothesis that will predict labels perfectly.

The main solution to this problem is to limit the hypothesis space $\mathcal{H}$. There are several methods we can use to avoid the overfitting.

2.6 Validation set

The basic idea is simple, first ML model learn the hypothesis h on training dataset $\mathcal{D}_{training}$, when we compute an average loss of h on data points outside the training dataset. We can split our dataset $\mathcal{D}$ into $\mathcal{D}_{training}$ and $\mathcal{D}_{val}$. Many extensions of this idea exist, one of the most popular is k-fold cross validation.

k-fold CV's main principle is to divide the entire dataset into k subsets (folds) repeatedly, each iteration, one subset is used as $\mathcal{D}_{val}$ and the other $k-1$ folds are used as $\mathcal{D}_{training}$. Then we average training error and average error obtained for each repetition.

2.7 Feature learning

One of the most important parts of model training is choosing the right features. Ideally, data points should contain only the most representative features that carry the relevant information needed to predict the label y of data point. The main problem is the right choice of such properties among all the available ones. The solution to this problem are feature learning methods that allow you to automate the selection or finding the required features. These methods find hypothesis map that reads in representation of data point and transforms it to the set of features. There are different methods of feature learning, some can help with reducing the number of features, others can create more new ones, based on the existing.

2.8 Dimensionality reduction

Dimensionality reduction is a feature learning method that reduces the number of features and thus helps to prevents overfitting. **dimensionality reduction**. The dimensionality reduction methods aims to find optimal mapping $h(\cdot) : \mathbb{R}^{n'} \to \mathbb{R}^{n}$ that transforms a long feature vector $\boldsymbol{z} \in \mathbb{R}^{n'}$ to a short feature vector

$$\boldsymbol{x} = (x_1, ..., x_n)^T := h(\boldsymbol{z}) \text{ (typically } n \ll n').$$

Feature learning can also be called representation learning, in the field of graphs, representation learning can be used to create properties based on the structure of graphs, on the basis of which you can build a classification model.

2.9 RiWalk

RiWalk [9] is algorithm for networks (graphs) feature learning, which is based on structural role identification of subgraphs. Let the $G = (V, E)$ is undirected graph, V is the set of nodes $\{v_1, v_2, ...v_{|V|}\}$, where $|V|$ is amount of nodes, and $E \subseteq V \times V$ is the set of $|E|$ edges. The main goal of algorithm is to learn a mapping function $f : V \to \mathbb{R}^d$, where $d \ll |V|$ is the number of dimensions of new representation. In the new latent space, structural similiar nodes are represented closely and nodes with different local structure are represented far apart.

The algorithm can be described as follows:

Algorithm 1: RiWalk algorithm [9]

```
  Input: Graph $G = (V, E)$, neighborhood size $k$, dimensions $d$, walks per node $\gamma$, walk
         length $\lambda$, window size $\omega$
1 for $i = 1, 2, ..., |V|$ do
2     $\tilde{G}(N^{k_i}) = \text{RoleIdentificationSubgraph}(G(N_i^k))$
3     $W_i = RandomWalkOnSubgraph(\tilde{G}(N^{k_i}), \gamma, \lambda)$
4     Add $W_i$ to $W$
5 end for
6 $f = \text{SkipGram}(W, d, \omega)$
7 return the learned node representation (embeddings) $f$
```

RoleIdentificationSubgraph is function, which can be defined as:

$$\psi_i(v_j) = h(\delta_i) \oplus h(\delta_j) \oplus s_{ij},$$

for each

$$v_j \in N_i^k \setminus \{v_i\},$$

where $\oplus$ is the concatenation operator. RandomWalkOnSubgraph is fixed length random walk from the anchor node v_i. Random walks can be performed in parallel. Then by merging all random walks together as a corpus, the Skip-Gram model with negative sampling is trained on it [10], [11].

Chapter 3

Artificial Neural Networks

In this chapter I will briefly explain what artificial neural networks are and present some of their types. The material in this chapter was taken from [12].

3.1 Neuron

Artificial neural networks get their name from mammalian brain cells - neurons, connected to each other. In artificial neural networks, each neuron (McCulloch-Pitts neuron)[13] is the computational unit (binary threshold unit), which has two states (outputs): active and inactive. For output computation, neuron sums the weighted inputs, and if the sum exceeds the given threshold, then the neuron is active, otherwise inactive. The state of neuron j at time t can be formulated as:

$$s_j(t) = \begin{cases} -1 & \text{inactive,} \\ 1 & \text{active.} \end{cases}$$

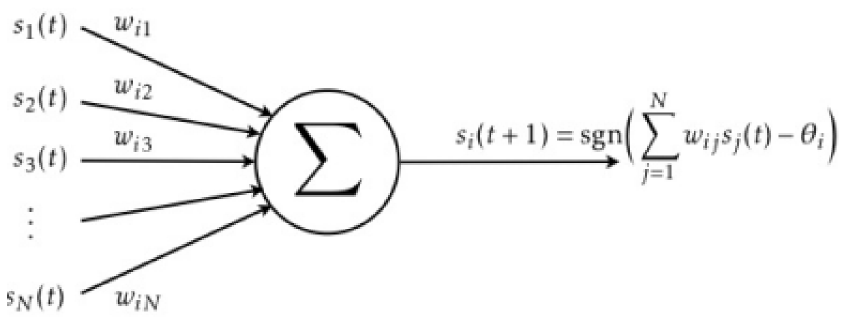

$s_1(t)$
w_{i1}
$s_2(t)$
w_{i2}
$s_3(t)$
w_{i3}
$s_N(t)$
w_{iN}
$\sum$
$s_i(t+1) = \text{sgn}\left(\sum_{j=1}^{N} w_{ij}s_j(t) - \theta_i\right)$

Figure 3.1 Schematic diagram of a McCulloh-Pitts neuron. i is the index of neuron, θ_i is the threshold value for neuron i, w_{ij} the input weights (strength) of the connection from neuron j to neuron i. t are the time sequence of computation steps, sgn(b) is the signum activation function [12].

For the given states $s_j(t)$, neuron with index i computes

$$s_i(t+1) = \text{sgn}(\sum_{j=1}^{N} w_{ij}s_j(t) - \theta_i) \equiv \text{sgn}[b_i(t)],$$

where $\mathrm{sgn}(b)$ is the signum function:

$$\mathrm{sgn}(b) = \begin{cases} -1, & b < 0, \\ +1, & b \geq 0. \end{cases}$$

It's argument

$$b_i(t) = \sum_{j=1}^{N} w_{ij} s_j(t) - \theta_i,$$

is called the local field, w_{ij} are called weights, indices i, j refer to the neuron that performs the calculation and to the neuron that is connected to it. Weights can be positive or negative, and when the $w_{ij} = 0$ there is no connection between these neurons. The threshold θ_i is also called bias, which is defined as the negative of θ_i, then we can formulate the neuron output as:

$$s_i(t+1) = \mathrm{sgn}(\sum_{j=1}^{N} w_{ij} s_j(t) + \mathrm{bias})$$

3.2 Perceptron

Perceptrons are the layered feed-forward networks of McCulloch-Pitts neurons, suggested by [14]. The perceptron consists of several layers, each containing the McCulloch-Pitts neurons described above. Each layer has its own task, the input layer, as the name implies, reads input and the output layer consists of resulting neurons. The middle layer is called hidden, the state of it's neurons are not read out. This network is called feed-forward, because there are no connections within the layers or connection to skip hidden layer, and there is no back connections. Neurons are connected strictly from input layer to the output.

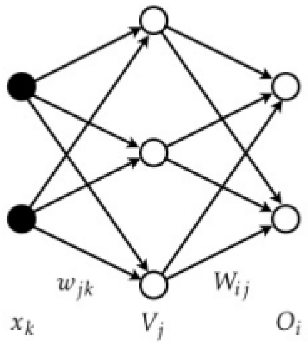


Figure 3.2 Feed-forward network with one hidden layer. x_k are the input terminals, V_j are the hidden neurons, O_j are the output neurons and $w_{jk}/W_{i,j}$ are the weights connecting to the hidden/output neurons [12].

The input of perceptron can be denoted as

$$\boldsymbol{x}^{(\mu)} = \begin{bmatrix} x_1^{(\mu)} \\ x_2^{(\mu)} \\ \vdots \\ x_N^{(\mu)} \end{bmatrix},$$

where $\mu = 1, ..., p$ are te different input patterns. The hidden's layer output is

$$V_j = g(b_j) \text{ with} b_j = \sum_k w_{jk} x_k - \theta_j,$$

where $g(b)$ is an activation function. The output layer performs similar calculations

$$O_i = g(B_i) \text{ with } B_i = \sum_j W_{ij} V_j - \theta_i$$

For classification problem, the target vectors corresponding to the input patterns are

$$\boldsymbol{t}^{(\mu)} = \begin{bmatrix} t_1^{(\mu)} \\ t_2^{(\mu)} \\ \vdots \\ t_M^{(\mu)} \end{bmatrix},$$

so the main idea is to find all weights and thresholds for the desired network's output:

$$O_i^{(\mu)} = t_i^{(\mu)} \text{ for all } i \text{ and } \mu.$$

Neurons with linear activation function are called linear unit, the formal solution for classification problem for them can be formulated as:

$$w_{ik} = \frac{1}{N} \sum_{\mu v} t_i^{(\mu)} (\mathbb{Q}^{-1})_{\mu v} x_k^{(v)}.$$

To find the solution iteratively, non-negative energy function exists:

$$H = \frac{1}{2} \sum_{i\mu} (t_i^{(\mu)} - O_i^{(\mu)})^2.$$

So now the goal is to find weights to minimize energy function H. To minimise H, the gradient descent is used:

$$w'_{mn} = w_{mn} + \delta w_{mn} \text{ with weight increments } \delta w_{mn} = -\eta \frac{\partial H}{\partial w_{mn}},$$

with learning rate $\eta > 0$. The derivatives are evaluated with the chain rule, and the weight increments are:

$$\delta w_{mn} = \eta \sum_{\mu} (t_m^{(\mu)} - O_m^{(\mu)}) x_n^{(\mu)}.$$

The states of hidden layer's neurons V_j do not depend on weights W_{mn}, so the increments for these weights are:

$$\delta W_{mn} = -\eta \frac{\partial H}{\partial W_{mn}} = \eta \sum_{\mu=1}^{p} (t_m^{(\mu)} - O_m^{(\mu)}) g'(B_m^{(\mu)}) V_n^{(\mu)} \equiv \eta \sum_{\mu=1}^{p} \Delta_m^{(\mu)} V_n^{(\mu)}.$$

The quantity

$$\Delta_m^{(\mu)} = (t_m^{(\mu)} - O_m^{(\mu)}) g'(B_m^{(\mu)})$$

is called as weighted output error, which is vanishes when $t_m^{(\mu)} = O_m^{(\mu)}$.

Weights for the hidden layer's neurons are adjusted by applying the chain rule 4 times, then the weighted errors for the hidden layers are:

$$\delta_m^{(\mu)} = \sum_i \Delta_i^{(\mu)} W_{im} g'(b^{(\mu)}),$$

where the $\delta_m^{(\mu)}$ vanishes when output errors $\Delta_i^{(\mu)}$ are zero. This is the main idea of backpropagation, the neurons are updated forward, and the errors are updated backwards.

3.3 Reccurent neural network

Reccurent neural networks are more general than perceptrons, they have feed-forward layout with feedbacks. Such neural networks use different connections, both between and within layers. In contrast to the perceptron, where the update rule for the i neuron from layer l may be represented as:

$$V_i^l = g(\sum_j w_{ij}^l V_j^{l-1} - \theta_i^l),$$

the RNN is used as the dynamical network, where layer l is replaced by iteration step t:

$$V_i(t) = g(\sum_j w^(vv)_{ij} V_j(t-1) + \sum_k w_{ik}^{vx} x_k - \theta_i^{(v)}) \text{ for } \ t = 1, 2, ...,$$

where w_{ij}^{vv} are the weights between neurons in the hidden layer and the w_{ik}^{vx} are the weights from the input neuron x_k to the neuron V_i.

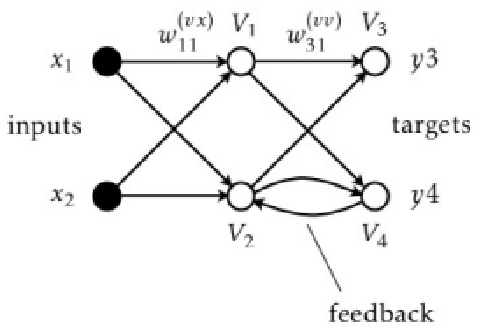


Figure 3.3 An example of network with feedback connection [12].

RNN can be trained using the stochastic gradient descent, but this algorithm suffers because of the vanishing-gradient problem, so usually in the hidden layer of RNN, neurons are not used, instead they are replaced by composite units, trained to act as different connections or non-linear units, that can learn correlations in the required form.

3.4 Autoencoder

Autoencoders are mainly used to eliminate noise in the data or to reduce the dimensionality of the input data is unseupervised manner. Autoencoder consists of two main parts: encoder and decoder. The encoder contains several fully connected layers, mapping the inputs to the smaller latent layer (low-dimensional space), $M \ll N$ where M is the number of neurons in latent layer, and N is the input dimension.

Encoder's mapping to latent space is $\mathbf{z} = f_e(\mathbf{x})$, and the decoder's mapping from latent space back to the input space (dimension) is $\mathbf{x} = f_d(\mathbf{z})$, where $\mathbf{x}$ is the input, and $\mathbf{z}$ is the latent variables. So the main goal of autoencoder is to learn approximation of the inputs as:

$$\mathbf{x} = f_d\left[f_e(\mathbf{x})\right].$$

The energy function for autoencoder can be defined as:

$$H = \frac{1}{2} \sum_\mu |\mathbf{x}^\mu - f_d\left[f_e(\mathbf{x}^{(\mu)})\right]|^2$$

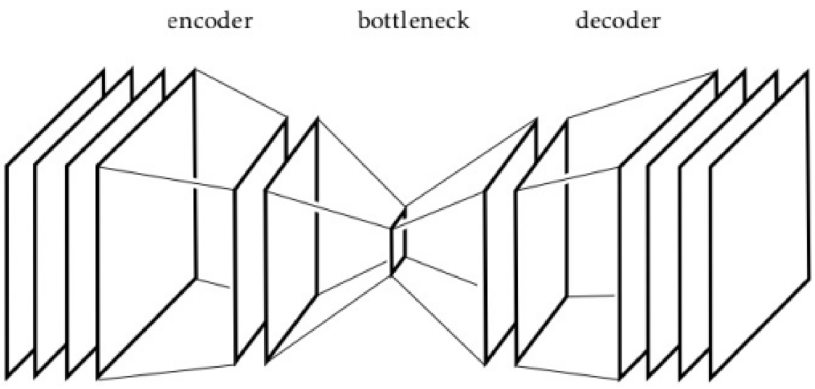


Figure 3.4 Autoencoder schematic struture [12].

3.5 Variational autoencoder

There are also generative autoencoder models, they are called variational [15], [16]. By manipulating the latent layer, it is possible to create new outputs from input variables. The main idea is to represent the data distribution in terms of a Gaussian distribution $P_L(\mathbf{z})$ of latent space variables $\mathbf{z}$. The training of VAE is also differs, the goal is to maximize log-likelihood:

$$\log P(\mathbf{x}^{(\mu)}) = \log \sum_{h_1=\pm1,\ldots,h_M=\pm1} P_B(\mathbf{v} = \mathbf{x}^{(\mu)}, \mathbf{h}),$$

for pattern $\mathbf{x}^{\mu}$. For VAE it is defined as:

$$\log P(\mathbf{x}) = \log \int d\mathbf{z} P(\mathbf{x}|\mathbf{z}) P_L(\mathbf{z}),$$

where $P(\mathbf{x}|\mathbf{z})$ is the probability of generating $\mathbf{x}$ for given $\mathbf{z}$.

3.6 Generative adversarial network

Generative adversarial network [17] is a generative model, which has similar to VAE structure, instead of an encoder-decoder structure, GAN contains two multilayer perceptrons called a generator and a discriminator. As the name implies, the generator creates new output data from the input data, while the discriminator's task is to determine whether the data are real or fake. Both networks are trained together, the main goal of generator is to adjust the weights so as to maximize the classification error of the discriminator, while the discriminator adjusts the weights so as to minimize it.

Chapter 4

Graph generation techniques

In this chapter I will introduce the existing graph generation techniques and their types, some of them will be used in experimental chapter. The presented techniques can mainly be divided into several types:

- Variational autoencoders
- Generative adversarial networks
- Auto-regressive models
- Diffusion models

4.1 Variational autoencoder

4.1.1 GraphVAE

GraphVAE [18] uses the idea of a variational autoencoder in the field of graph generation. Learning to generate graph is more difficult problem for gradient optimization based methods, unlike the text, graphs can have can have more random connectivity, and there is no best way exists, how to linearize graphs construction sequentially. GraphVAE models probabilities of nodes and edges existence as independent random variables.

The main idea of this method is to create probabilistic fully-connected graph, and then to align it to the ground truth using standart graph matching algorithm. Let $G = (A, E, F)$ be a graph with adjacency matrix A, edge attribute tensor E, and node attribute matrix F. The goal is to learn encoder and decoder to map between the graph's space G and graph's continuous latent representation $\mathbf{z} \in \mathbb{R}^c$. In the probabilistic setting of VAE, encoder is defined by variational posterior $q_\phi(\mathbf{z}|G)$ and the decoder by a generative distribution $p_\theta(G|\mathbf{z})$, where θ and ϕ are the learned parameters. The whole model is minimizing the upper bound on negative log-likelihood $\log p_\theta(G)$ [15]:

$$\mathcal{L}(\phi, \theta; G) = \mathbb{E}_{q\phi(\mathbf{z}|G)}\left[-\log p_\theta(G|\mathbf{z})\right] + \mathrm{KL}\left[q_\phi(\mathbf{z}|G)||p(\mathbf{z})\right],$$

where $\mathbb{E}_{q\phi(\mathbf{z}|G)}\left[-\log p_\theta(G|\mathbf{z})\right]$ is the reconstruction loss, which enforces high similarity of generated graphs to the input graph G, and $\mathrm{KL}\left[q_\phi(\mathbf{z}|G)||p(\mathbf{z})\right]$ is th KL-divergence, which regularizes the code space to allow $\mathbf{z}$ sampling directly from $p(\mathbf{z})$. Regularization is independent on the input space, but the reconstruction loss must be designed specifically:

$$-\log\, p(G|\mathbf{z}) = -\lambda_A \log\, p(A'|\mathbf{z}) - \lambda_F \log\, p(F|\mathbf{z}) - \lambda_E \log\, p(E|\mathbf{z}).$$

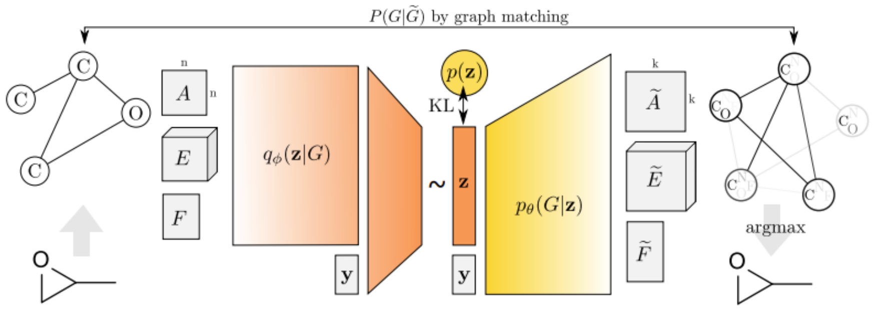


Figure 4.1 GraphVAE workflow schematic illustration. GraphVAE takes as an input discrete attributed graph $G = (A, E, F)$ on n nodes, stochastic encoder $q_\phi(\mathbf{z}|G)$ embeds graph into latent representation $\mathbf{z}$, then probabilistic decoder $p_\theta(G|\mathbf{z})$ outputs probabilistic fully-connected graph $\tilde{G} = (\tilde{A}, \tilde{E}, \tilde{F})$ on predefined $k \geq n$ nodes, from which discrete samples may be drawn. The process can be conditioned on label $\mathbf{y}$ for controlled sampling at the test time [18].

More particular:

$$\log\, p(A'|\mathbf{z}) = \frac{1}{k}\sum_a A'_{a,a} + (1-A'_{a,a})\log(1-A'_{a,a}) + \frac{1}{k(k-1)}\sum_{a\neq b}(A'_{a,b})\log(A'_{a,b}) + (1-A'_{a,b})\log(1-A'_{a,b})$$

for $A' = XAX^T$, and

$$\log\, p(F|\mathbf{z}) = \frac{1}{n}\sum_i \log F^T_{i,\cdot}\tilde{F}'_{i,\cdot},$$

$$\log\, p(E|\mathbf{z}) = \frac{1}{(||A||_1 - n)}\sum_{i\neq j}\log E^T_{i,j,\cdot}\tilde{E}'_{i,j,\cdot},$$

where $\tilde{F}' = X^T\tilde{F}$, $\tilde{E}'_{\cdot,\cdot,l} = X^T\tilde{E}_{\cdot,\cdot,l}X$. $X \in \{0,1\}^{k\times n}$ is binary assignment matrix, where $X_{a,i} = 1$ only if node $a \in \tilde{G}$ is assigned to $i \in G$, otherwise $X_{a,i} = 0$.

The graph matching is based on finding $X \in \{0,1\}^{k\times n}$ between nodes of graphs G and $\tilde{G}$ by their node pairs similarity $S : (i,j) \times (a,b) \to \mathbb{R}^+$ for $i,j \in G$ and $a,b \in \tilde{G}$. So the similarity function is defined as:

$$S((i,j),(a,b)) = (E^T_{i,j,\cdot}\tilde{E}_{a,b,\cdot}A_{i,j}\tilde{A}_{a,b}\tilde{A}_{a,a}\tilde{A}_{b,b})\,[i\neq j \wedge a\neq b] + (F^T_{i,\cdot}\tilde{F}_{a,\cdot})\tilde{A}_{a,a}\,[i=j\wedge a=b]\,.$$

Encoder of graphVAE is a feed forward network with edge-conditioned graph convolutions (ECC) [19], formulated as a probabilistic and enforces Gaussian distribution of $q_\phi(\mathbf{z}|G)$ by the outputs of last encoder layer, which are the mean and variance. This allows to sample $\mathbf{z}_l \sim N(\mu_l(G), \sigma_l(G))$ for $l \in 1, ..., c$ using the reparameterization trick [15].

Decoder is simple multilayer perceptron with three outputs in the last layer. Decoder's output is probabilistic fully-connected graph $\tilde{G} = (\tilde{A}, \tilde{E}, \tilde{F})$ on k nodes, where each tesor is a probabilistic interpretation:

$\tilde{A} \in [0,1]^{k\times k}$ contains node probabilities $\tilde{A}_{a,a}$ and edge probabilities $\tilde{A}_{a,b}$ for nodes $a \neq b$.

$\tilde{E} \in \mathbb{R}^{k\times k\times d_e}$ is an edge attribute tensor of class probabilities for edges, $\tilde{F} \in \mathbb{R}^{k\times d_n}$ is the similar tensor for nodes. To compute $\tilde{A}$, decoder uses sigmoid activation function, and to obtain $\tilde{E}$ and $\tilde{F}$ edge- and node-wise softmax is applied.

Proposed model is useful only for generating small graphs, when using a decoder on large graphs, it has difficulty capturing complex properties (in this case chemical interactions), also this model has limitations on graph's size due to increasing complexity and memory requirements.

4.2 Generative adversarial networks

4.2.1 MolGAN

MolGAN [20] is an implicit, likelihood-free generative model for small molecular graph. This method adapts GANs to operate directly on graph-strutured data. The main purpose of this model is to generate molecular graphs with desired requirements for their chemical properties.

Most works in the field of molecular generation use the so-called SMILES [21] representation of molecules, but proposed GAN model is the first to address the generation of graph-structured data in molecular domain. MolGAN's generative model predicts discrete graph struture in non-sequencial way (at once), but sequintial variants are also possible, the discriminator is permutation-invariant and reward network (for RL-based optimization of desired chemical properties) based on graph convolution layers. The both discriminator and reward network operate directly on graph-structured representation.

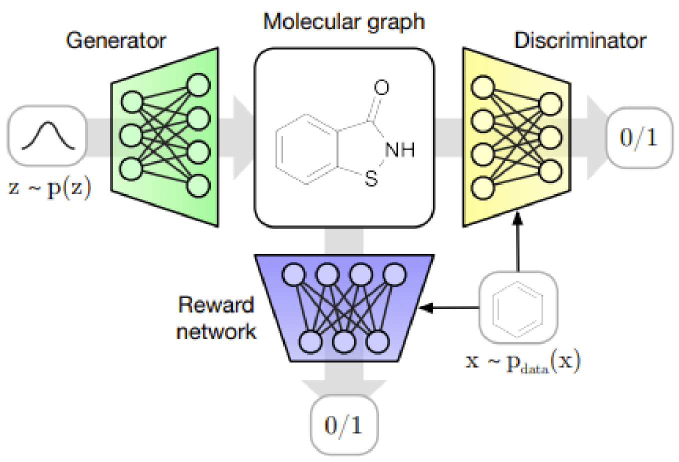


Figure 4.2 MolGAN's shematic struture. A vector **z** is sampled from prior and then passed to the generator, generator's result is graph-structure representation of molecule. The discriminator determines whether a given molecule is generated or not. Rewards network gives out a reward depending on the chemical properties of the molecule, determined by an external software [20].

MolGAN consists of three main components: a generator G_θ, a discriminator D_ϕ, and a reward network $\hat{R}_\psi$. The generator G_θ takes the sample from prior distribution and generates an annotated graph G, where edges represent molecular bonds, and nodes represent atoms. Discriminator D_ϕ takes this sample and one sample from dataset, then it learns to distiguish them. G_θ and D_ϕ are trained using improved WGAN.

WGANs [22] minimize Earth Moved distance defined between two porbability distributions p and q:

$$D_W\left[p||q\right] = \frac{1}{K} \sup_{||f||_L < K} \mathbb{E}_{x\sim p(x)}\left[f(x)\right] - \mathbb{E}_{x\sim q(x)}\left[f(x)\right],$$

where p is empirical distribution and q is the generator distribution.

The generator $G_\phi(\mathbf{z})$ takes D-dimensional vectors $\mathbf{z} \in \mathbb{R}^D$ sampled from standart distribution $\mathbf{z} \sim N(\mathbf{0}, \mathbf{I})$ and generates graphs. For each $\mathbf{z}$, G_θ outputs: $\mathbf{X} \in \mathbb{R}^{N\times T}$ that defines atoms and $\mathbf{A}^{N\times N\times Y}$ that defines bonds type. Both $\mathbf{A}$ and $\mathbf{X}$ are interpreted as probabilistic, so to generate molecule, sparse objects $\tilde{\mathbf{A}}$, $\tilde{\mathbf{X}}$ are obtained via categorical sampling from $\mathbf{A}$ and $\mathbf{X}$.

The discriminator D_ϕ is trained only using WGAN objective, but the generator G_θ uses linear combination of the WGAN loss and the RL loss:

$$L(\theta) = \lambda L_{WGAN}(\theta) + (1-\lambda) L_{RL}(\theta),$$

where $\lambda \in [0, 1]$ is a hyperparameter that regulates the tradeoff between these components.

The reward network $\hat{R}_\psi$ approximates the reward function of a sample and optimizes molecule generation metrics using reinforcement learning. $\hat{R}_\psi$ learns to assign reward to molecule to match the score provided by external software, when MolGAN's output is not a valid molecule, it is not possible to assign reward, because molecule is not even compund, so for such graphs zero reward is assigned.

Both the D_ϕ and $\hat{R}_\psi$ recieve graph as input, and output the scalar value. There is the same architecture for both networks RelationalGCN [23], but they do not share the weights between each other. A series of convlutional layers convolve node signals $\tilde{\mathbf{X}}$ from adjacency tensor $\tilde{\mathbf{A}}$.

At every layer feature representation of nodes are propogated according to:

$$\mathbf{h'}_i^{(l+1)} = f_s^{(l)}(\mathbf{h}_i^{(l)}, x_i) + \sum_{j=1}^{N}\sum_{y=1}^{Y} \frac{\tilde{\mathbf{A}}_{ijy}}{|N_i|} f_y^{(l)}(\mathbf{h}_j^{(l)}, x_j),$$

$$\mathbf{h}_i^{(l+1)} = \tanh(\mathbf{h'}_i^{(l+1)}),$$

where $\mathbf{h}_i^{(l)}$ is the signal of node i at layer l and $f_s^{(l)}$ is a linear transformation function that acts as self-connection between layers. Further an edge type-specific affine function $f_y^{(l)}$ is used for each layer. The normalization factor $\frac{1}{|N_i|}$ ensures that activations are on a similiar scale irrespective of the number of neighbors. After several propagations node embeddings are aggregated into a graph level representation vector as

$$\mathbf{h'}_G = \sum_{v\in V} \sigma(i(\mathbf{h}_v^{(L)}, x_v)) \odot \tanh(j(\mathbf{h}_v^{(L)}, x_v)),$$

$$\mathbf{h}_G = \tanh(\mathbf{h'}_G),$$

where $\sigma(x) = \frac{1}{(1+\exp(-x))}$ is the logistic sigmoid function, i and j are multilayer perceptrons with linear output layer and $\odot$ is an element-wise multiplication. $\mathbf{h}_G$ is a graph's vector representation, which is further processed by multilayer perceptron to produce scalar outputs for the discriminator and the reward network.

Compared to a recent SMILES-based sequential GAN model for molecular generation, MolGAN can achieve higher chemical property scores, while allowing for at least $\sim$5x faster training time. The central MolGAN's limitation is it's susceptibility to mode collapse, because both the

GAN and the RL objective do not encourage generation of diverse, non-unique outputs. This ultimately results in the generation of only a handful of different molecules if training is not stopped early. Current MolGAN's one-shot prediction of the adjacency tensor is feasible only for graphs of small size.

4.3 Auto-regressive models

4.3.1 GraphRNN

GraphRNN is represented by the authors [24] as a scalable framework for learning generative models of graphs, which models a graph in an autoregressive (or recurrent) manner. Generation is presented in the form of sequential additions of new nodes and edges to capture the complex joint probability of all nodes and edges in the graph.

This model, unlike other methods (such as VAE or GAN), represents graphs under different node orderings as sequences, so it doesn't suffer from serious drawback, and allows to generate variable-sized graphs without requirement to train all possible node permutations or specifying a cannonical permutaition, both of which require $O(n!)$ time in general.

G is an undirecteed graph $G = (V, E)$, $V = \{v_1, ..., v_n\}$ and $E = \{(v_i, v_j)|v_i, v_j \in V)\}$, under a node ordering π, G can be represented by the adjacency matrix $A^\pi \in \mathbb{R}^{n\times n}$, where $A^\pi_{i,j} = \mathbb{1}[\pi(v_i, \pi(v_j)) \in E]$. Adjacency matrix A requires a node ordering π that maps nodes to rows/columns of A, precisely, π is a permutation function over V, and the set of all $n!$ possible node permutations is defined as Π. Note that elements in the set of adjacency matrices A^Π all correspond to the same graph.

The goal is to learn distribution $p_{model}(G)$ over graphs, based on set of observed graphs $\mathbb{G} = \{G_1, ..., G_s\}$ sampled from data distribution $p(G)$, where each G_i may have different number of nodes and edges. The key idea of this approach is to represent graphs under different node orderings as a sequences, and then to build autoregressive model on these sequences. f_S is the defined mapping from graphs to sequences, where for a graph $G \sim p(G)$ with n nodes under node ordering π:

$$S^\pi = f_S(G, \pi) = (S^\pi_1, ..., S^\pi_n),$$

where each $S^\pi_i \in \{0,1\}^{i-1}, i \in \{1, ..., n\}$ is an adjacency vector representing the edges between node $\pi(v_i)$ and the previous nodes $\pi(v_j), j \in \{1, ..., i-1\}$ already in graph:

$$S^\pi_i = (A^\pi_{1,i}, ..., A^\pi_{i-1,i})^T, \forall i \in \{2, ..., n\}.$$

For undirected graps, S^π determines a unique graph G, so the mapping is denoted as $f_G(\cdot)$ where $f_G(S^\pi) = G$.

Summary, we can define $p(G)$ as the marginal distribution of the joint distribution $p(G, S^\pi)$:

$$p(G) = \sum_{S^\pi} p(S^\pi)\mathbb{1}[f_G(S^\pi) = G],$$

where $p(S^\pi)$ is the distribution that we want to learn using model. Further we decompose $p(S^\pi)$ as the product of conditional distributions over the elements:

$$p(S^\pi) = \prod_{i=1}^{n+1} \underbrace{p(S^\pi_1|S^\pi_1, ...S^\pi_{i-1})}_{p(S^\pi_i|S^\pi_{<i})},$$

where S^π_{n+1} is the end of sequence token EOS, using to represent sequences with variable lengths.

GraphRNN inference algorithm [24] can be described as follows:

Algorithm 2: GraphRNN inference algorithm

```
Input: RNN-based transition module $f_{trans}$, output module $f_out$, probability
       distribution $R_{\theta_i}$, parameterized by $\theta_i$, start token SOS, end token EOS and
       empty graph state $h'$
Output: Graph sequence $S^{\pi}$ $S^{\pi}_1$ = SOS, $h_1 = h', i = 1$
1 repeat
2     $i = i + 1$
3     $h_i = f_{trans}(h_{i-1}, S^{\pi}_{i-1})$ ;                // update graph state
4     $\theta_i = f_{out}(h_i)$
5     $S^{\pi}_i \sim R_{\theta_i}$ ;                 // sample node $i$'s edge connections
6 until $S^{\pi}_i$ is EOS;
7 return $S^{\pi} = (S^{\pi}_1, ..., S^{\pi}_i)$
```

This RNN consists of a state-transition function and an output function:

$$h_i = f_{trans}(h_{i-1}, S^{\pi}_{i-1}),\, \theta_i = f_{out}(h_i),$$

where $h_i \in \mathbb{R}^d$ is a vector that encodes the state of the current generated graph, S^{π}_{i-1} is the adjacency vector for the last generated node $i-1$, and θ_i represents next node's adjacency vector distribution.

Depending on the assumption about $p(S^{\pi}_i|S^{\pi}_{<i})$ there are different variants of GraphRNN. Authors proposed two variants, both of them implement f_{trans} as a Gated Recurrent Unit (GRU)[25], but differ in implementation of f_{out}. In simplified version, authors implement f_{out} as single multi-layer percetron (MLP) with sigmoid activation function, which shares weights across time steps. This version models $p(S^{\pi}_i|S^{\pi}_{<i})$ as a multivariate Bernoulli distribution, parameterized by the $\theta_i \in \mathbb{R}^{i-1}$ vector which is output of f_{out}. θ_i is a vector, it's element $\theta_i[j]$ represents a probability of edge (i, j). According to multivariate Bernoulli distribution, we can independently sample edges in S^{π}_i.

In the full version an another additional decomposition is used:

$$p(S^{\pi}_i|S^{\pi}_{<i}) = \prod_{j=1}^{i-1} p(S^{\pi}_{i,j}|S^{\pi}_{i,<j}, S^{\pi}_{<i}),$$

where $S^{\pi}_{i,j}$ is a binary scalar denoting the junction of node $\pi(v_{i+1})$ with node $\pi(v_j)$. There are two RNNs in this variant:

- Node sequence generator (graph-level RNN)
- Edge per new node generator (edge-level RNN)

Graph-level RNN generates the nodes and maintains graph's state, while the edge-level RNN is a GRU model, which generates the edges for a given node, more precisely, it's hidden state is initialized via the graph-level RNN hiden state h_i, and the output at each step is mapped by MLP to a scalar that indicates the probability of the edge existence. $S^{\pi}_{i,j}$ is sampled from distribution specified by the jth output of ith edge-level RNN, and then is fed into $j+1$th input of the same RNN. All edge-level RNNs share the same parameters.

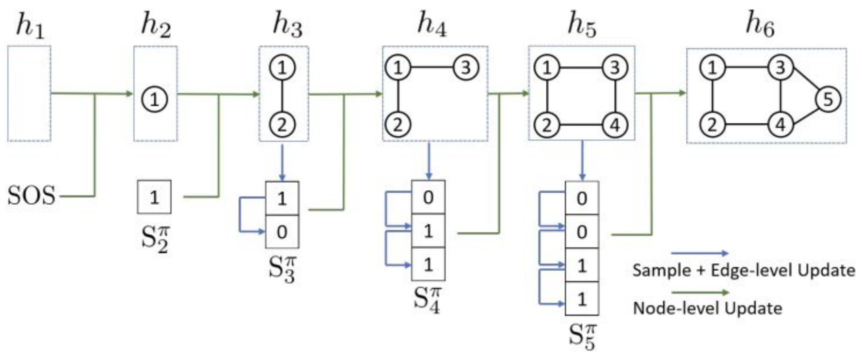


Figure 4.3 GraphRNN at inference time. Green arrows denote the graph-level RNN that encodes the "graph state" vector h_i in its hidden state, updated by the predicted adjacency vector S_i^π for node $\pi(v_i)$. Blue arrows represent the edge-level RNN, whose hidden state is initialized by the graph-level RNN, that is used to predict the adjacency vector S_i^π for node $\pi(v_i)$ [24].

The key method presented by the authors is the use of the BFS algorithm for node orderings. Formally:

$$S^\pi = f_S(G, \mathbf{BFS}(G, \pi)),$$

where $\mathbf{BFS}(\cdot)$ is deterministic BFS function. This method avoids learning to generate graphs under all possible node permutation without a loss of generality, we only need to train on all possible BFS orderings. According to the authors, this approach reduces the number of required operations from $O(n^2)$ on worst-case to sub-quadratic complexity in many cases.

Authors proposed GraphRNN, an autoregressive generative model for graph-structured data, along with a comprehensive evaluation suite for the graph generation problem, which is used to show that GraphRNN achieves significantly better performance compared to previous state-of-the-art models, while being scalable and robust to noise.

4.3.2 MolecularRNN

MolecularRNN [26] is graph reccurent model for direct generation of molecular graph structure with high validity and novelty,this model extends GraphRNN with capability to hold graph's nodes and edges type. For molecule representation, atoms are mapped to nodes, and edges are represent categorical bond types as: $S_{i,j}^\pi \in \{0,1,2,3\}$ corresponding to no, single, double, and triple. Also for each node there are categorical mappings: $C_i^\pi \in \{1,2,3,...,K\}$, representing atoms. So for the MolecularRNN, likelihood from GraphRNN is rewritten as:

$$p(S^\pi, C^\pi) = \prod_{i=1}^{n+1} p(C_i^\pi | S_{<i}^\pi, C_{<i}^\pi) p(S_i^\pi | C_i^\pi, S_{<i}^\pi, C_{<i}^\pi),$$

where $p(C_{n+1}|S_{<n+1}, C_{<n+1}) \equiv 1$ for terminal node $n+1$.

In this model, after the sub-graph on the first $i-1$ nodes under permutation π is completed, NodeRNN can instantly determine atom type of the following node i, as well the model switches to EdgeRNN and links generated node to the set $\{1,...,i-1\}$. Model's structure can be formulated as:

$$\text{input}_{i-1} = \left[emb(S_{i-1}^\pi), emb(C_{i-1}^\pi)\right]$$

$$h_i^{\text{node}} = \text{NodeRNN}(h_{i-1}^{\text{node}}, \text{input}_{i-1}), \quad h_0^{\text{node}} = 0$$

$$\psi_i = \text{NodeMLP}(h_i^{\text{node}})$$

$$h_{i,j}^{\text{edge}} = \text{EdgeRNN}(h_{i,j-1}^{\text{edge}}, emb(S_{i,j-1}^{\pi})), \quad h_{i,0}^{\text{edge}} = h_i^{\text{node}}$$

$$\phi_{i,j} = \text{EdgeMLP}(h_{i,j}^{\text{edge}}),$$

where *emb* is the embedding for categorical inputs, EdgeMLP and NodeMLP are two-layer MLP with softmax activation on top of hidden states h^{node} and h^{edge} for categorical predictions.

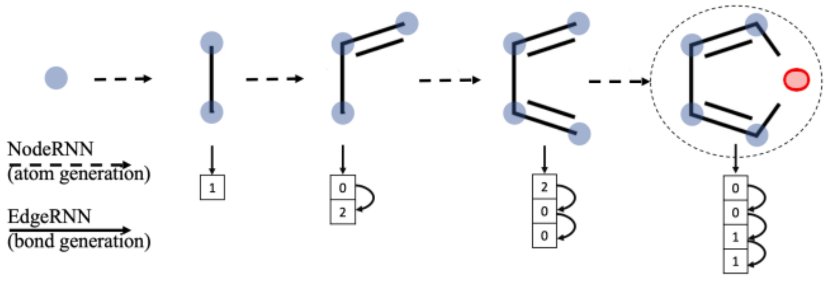


Figure 4.4 MolecularRNN model. NodeRNN unrolls across atoms and predicts type of the next atom. EdgeRNN is initialized for every atom with NodeRNN hidden state, unrolls across preceding atoms for bond types prediction [26].

MolecularRNN utilizes valency-based rejection sampling, so in each step can be ensured, that the sum of sampled molecule's bonds does not exceed allowed valency. This controlling process is formulated as:

$$\sum_j A_{i,j}^{\pi} + k \leq \text{valency}_{C_i^{\pi}} \text{ and } \sum_i A_{i,j}^{\pi} + k \leq \text{valency}_{C_j^{\pi}},$$

for bond of order k between i and j atoms. If the atoms are not filled up, their valences are complemented with hydrogens.

For the molecule property optimization, MolecularRNN acts as a policy network and outputs probability of the next action based on the current state. The set of states is defined as all possible sub-graphs of graphs with fixed number of N nodes. In MolecularRNN BFS ordering, initial state s_0 is a graph with singe carbon atom (node). Final states are defined as all graphs that correspond to a valid molecule with up no N heavy atoms. The training represents the policy gradient optimization algorithm(CITE), so the loss function is formulated as:

$$L(\theta) = -\sum_{i=1}^{N} r(s_N) \cdot \gamma^i \cdot \log p(s_i|s_{i-1};\theta),$$

where $r(s_N)$ is a reward for state s_N.

MolecularRNN model for generating valid molecules with desired properties, which learns diverse distribution through unsupervised pretraining and utilizes policy gradient optimization.

4.3.3 GraphGen

GraphGen [27] is a domain-agnostic scalable generative model with a graphRNN - like structure, also using DFS code to transform graph to sequences, but it is capable to handle node and edge lables.

Instead of using adjacency matrices, GraphGen uses graph canonization, and works with DFS codes sequences, which results in faster and better modeling.

GraphGen utilizes custom LSTMs [28], because they have proven themselves well in the problem of RNN training. This custom LSTM, proposed by authors, consists of a state transition function f_{trans}, embedding function f_{emb} and five separate output functions for each of the components $s_i = (t_u, t_v, L_u, L_e, L_v)$:

$$h_i = f_{\text{trans}}(h_{i-1}, f_{\text{emb}}(s_{i-1}))$$

$$t_u \sim M\theta_{t_u} = f_{t_u}(h_i)$$

$$t_v \sim M\theta_{t_v} = f_{t_v}(h_i)$$

$$L_u \sim M\theta_{L_u} = f_{L_u}(h_i)$$

$$L_e \sim M\theta_{L_e} = f_{L_e}(h_i)$$

$$L_v \sim M\theta_{L_u} = f_{L_v}(h_i)$$

$$s_i = \text{concat}(t_u, t_v, L_u, L_e, L_v)$$

Here $\sim M$ represents sampling from multinomial distribution, $s_i \in \{0,1\}^k$ is the concatenated component wise one-hot encoding of the real edge, and $h_i \in \mathbb{R}^d$ is LSTM hidden state vector, which encodes the state of generated graph. Embedding function f_{emb} compresses sparse edge representation (s_i) to a small vector of real numbers. Each function f represents the multinomial distribution over possibilities of five components of new edge tuple, that after are concatenated to from the final new edge.

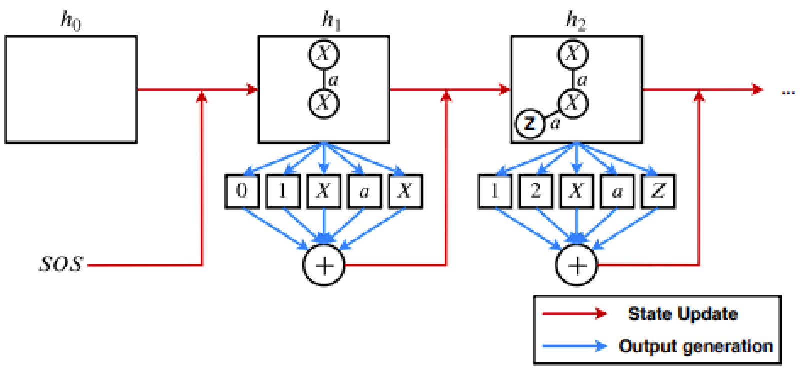


Figure 4.5 The GraphGen's architecture. Red arrows represent data flow in the RNN with hidden state h_i, which captures the state of currently generated graph. Blue arrows indicate information flow to generate edge tuple s_i.

The training goal is to learn functions $f_{t_u}, f_{t_v}, f_{L_u}, f_{L_e}, f_{L_v}$ from a set of training graphs.

Algorithm 3: Graph modelling algorithm

```
Input: Graphs training dataset $\mathbb{G} = \{G_1, ..., G_n\}$
Output: Learned function $f_{t_u}, f_{t_v}, f_{L_u}, f_{L_e}, f_{L_v}$ and embedding function $f_{\text{emb}}$
1  $\mathbb{S} = \{S = \mathcal{F}(G) | \forall G \in \mathbb{G}\}$
2  Initialize $f_{t_u}, f_{t_v}, f_{L_u}, f_{L_e}, f_{L_v}, f_{\text{emb}}$
3  repeat
4      for $\forall S = [s_1, ..., s_m] \in \mathbb{S}$ do
5          $s_0 \leftarrow$ SOS; Initialize $h_0$
6          loss $\leftarrow 0$
7          for $i$ from 1 to $m+1$ ;                  // $s_{m+1}$ for EOS tokens
8           do
9              $h_i \leftarrow f_{\text{trans}}(h_{i-1}, f_{\text{emb}}(s_{i-1}))$
10             $\tilde{s}_i \leftarrow \phi$ ; // $\tilde{s}_i$ will contain component-wise probability distribution
                vectors of $\hat{s}_i$
11             for $c \in \{t_u, t_v, L_u, L_e, L_v\}$ do
12                 $\theta_c \leftarrow f_c(h_i)$
13                 $\tilde{s}_i \leftarrow \text{concat}(\tilde{s}_i, \theta_c)$
14             end for
15             loss $\leftarrow$ loss + BCE($\tilde{s}_i, s_i$)
16         end for
17         Back-propagate loss and update weights
18     end for
19 until stopping criteria ;       // Typically when validation loss is minimized
20 ;
```

Accuracy of the model is optimized using binary cross-entropy loss function:

$$\text{BCE}(\tilde{s}_i, s_i) = -\sum_c (s_i\,[c]^T \log\tilde{s}_i\,[c] + (1 - s_i\,[c])^T \log(1 - \tilde{s}_i\,[c])),$$

where $s\,[c]$ represents component $c \in \{t_u, t_v, L_u, L_e, L_v\}$, and log is taken elementwise on vector.

Algorithm 4: Graph generation algorithm

```
Input: LSTM based state transition function $f_{\text{trans}}$, embedding function $f_{\text{emb}}$, output
       functions $f_{t_u}, f_{t_v}, f_{L_u}, f_{L_e}, f_{L_v}$, empty graph state $h_0$, SOS and EOS tokens
Output: Graph $G$
1  $S \leftarrow \emptyset$
2  $\hat{s}_0 \leftarrow$ SOS
3  $i \leftarrow 0$
4  repeat
5      $i \leftarrow i + 1$
6      $h_i \leftarrow f_{\text{trans}(h_{i-1}, f_{\text{emb}}(\hat{s}_{i-1}))}$
7      $\hat{s}_i \leftarrow \emptyset$
8      for $c \in \{t_u, t_v, L_u, L_e, L_v\}$ do
9          $\theta_c \leftarrow f_c(h_i)$ ;  // sample $c_i$ from multinomial distribution parameterized
            by $\theta_c$
10         $c_i \sim_M \theta_c$
11         $\hat{s}_i \leftarrow \text{concat}(\hat{s}_i, c_i)$
12     end for
13     $S \leftarrow S || \langle \hat{s}_i \rangle$
14 until $\exists c_i, c_i = EOS$;
15 return $\mathcal{F}^{-1}(S)$ ;          // $\mathcal{F}^{-1}$ is mapping from minimum DFS code to graph
```

GraphGen, using minimum DSF codes of graphs, provides more scalability within certain limits and quality,but still as the size of the graphs increases, the quality also decreases, also the DFS's complexity is non-polynomial, which is a disadvantage of RNN family.

4.4 Diffusion models

4.4.1 Graph diffusion via the system of stochastic differential equations

Graph diffusion via the system of stochastic differential equations (GDSS) [29] as the name implies, models the joint distribution of graph nodes and edges using the system of stochastic differential equations (SDEs). The goal of the model is to generate graphs, that will closely follow the distribution of training graphs, to avoid direct representation of the distribution, there is a continuous-time score-based generative framework, which transforms graphs to toise, while modelling dependencies between nodes and edges.

Let the G be a graph with N nodes, defined by node features $X \in mathbbR^{N\times F}$ and weighted adjacency matrix $A \in \mathbb{R}^{N\times N}$, where F is the node features dimension. The diffusion process is used to model dependency between X and A, by transforming both to a simple noise distribution. It can be represented as a trajectory of random variables $\{G_t = (X_t, A_t)\}_{t\in[0,T]}$ in a fixed time horizon $[0, T]$:

$$d\mathbf{G}_t = \mathbf{f}_t(\mathbf{G}_t)dt + \mathbf{g}_t(\mathbf{G}_t)d\mathbf{w}, \quad \mathbf{G}_0 \sim p_{data},$$

where $\mathbf{f}_t(\cdot) : \mathcal{G} \to \mathcal{G}$ is the linear drift coefficient, $\mathbf{g}_t(\cdot) : \mathcal{G} \to \mathcal{G} \times \mathcal{G}$ is the diffusion coefficient and $\mathbf{w}$ is the standard Wiener process. The reverse-time diffusion process proposed by authors is novel and modeled by the following system of SDEs:

$$\begin{cases} d\mathbf{X}_t = \left[\mathbf{f}_{1,t}(\mathbf{X}_t) - g_{1,t}^2 \nabla_{\mathbf{X}_t} \log p_t(\mathbf{X}_t, \mathbf{A}_t)\right] d\bar{t} + g_{1,t} d\bar{\mathbf{w}}_1 \\ d\mathbf{A}_t = \left[\mathbf{f}_{2,t}(\mathbf{A}_t) - g_{2,t}^2 \nabla_{\mathbf{A}_t} \log p_t(\mathbf{X}_t, \mathbf{A}_t)\right] d\bar{t} + g_{2,t} d\bar{\mathbf{w}}_2 \end{cases},$$

where $\mathbf{f}_{1,t}$ and $\mathbf{f}_{2,t}$ are linear drift coefficients satisfying $\mathbf{f}(\mathbf{X}, \mathbf{A}) = (\mathbf{f}_{1,t}(\mathbf{X}), \mathbf{f}_{2,t}(\mathbf{A}))$, $g_{1,t}$ and $g_{2,t}$ are scalar diffusion coefficients, and $\bar{\mathbf{w}}_1, \bar{\mathbf{w}}_2$ are reverse-time standard Wiener processes. These forward and reverse diffusions processes of graphs are the main idea of GDSS. The key property of GDSS is that these diffusion processes are dependent on each other, related by the joint log-density gradients $\nabla_{\mathbf{X}_t} \log p_t(\mathbf{X}_t, \mathbf{A}_t)$ and $\nabla_{\mathbf{A}_t} \log p_t(\mathbf{X}_t, \mathbf{A}_t)$, that are the partial score functions. Using partial scores to model the dependency between components over time, GDSS can represent the process of diffusion of the entire graph, consisting of nodes and edges.

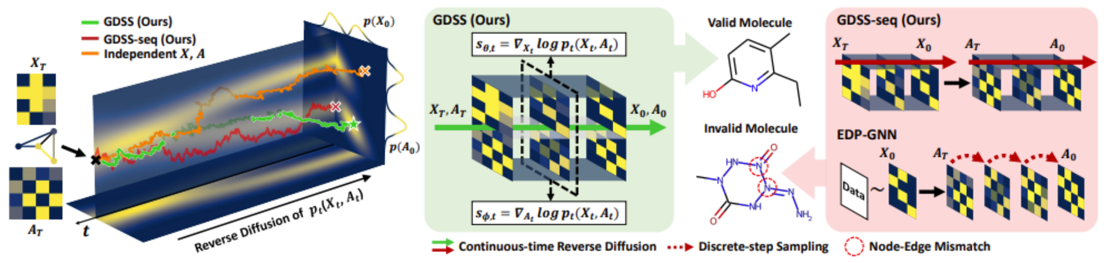


Figure 4.6 (Left) Graph generation through the reverse-time diffusion process. Colored trajectories denote different types of diffusion processes in the joint probability space of node features **X** and adjacency **A(Right) Score-based graph generation framework.** GDSS generates **X** and **A** simultaneously by modeling dependency through the time. [29]

One of the variants of this method is the continuous-time version of EDP-GNN [30], by ignoring the diffusion process of $\mathbf{X}$ with $\mathbf{f}_{1,t} = g_{1,t} = 0$ and choosing prior distribution of $\mathbf{X}$ as the data distribution, so the diffusion process of $\mathbf{A}$ generalizes discrete-step pertrubation with a finite noise scales. Another proposed variant generates $\mathbf{X}$ and $\mathbf{A}$ sequentially. So by neglecting part of the dependency by the assumptions $\nabla_{\mathbf{X}_t} \log p_t(\mathbf{X}_t, \mathbf{A}_t) \approx \nabla_{\mathbf{X}_t} \log p_t(\mathbf{X}_t)$ and $\nabla_{\mathbf{A}_t} \log p_t(\mathbf{X}_t, \mathbf{A}_t) \approx \nabla_{\mathbf{A}_t} \log p_t(\mathbf{X}_0, \mathbf{A}_t)$ SDEs for the diffusion process can be formulated as:

$$\begin{cases} d\mathbf{X}_t = \left[\mathbf{f}_{1,t}(\mathbf{X}_t) - g_{1,t}^2 \nabla_{\mathbf{X}_t} \log p_t(\mathbf{X}_t) d\right] \bar{t} + g_{1,t} d\bar{\mathbf{w}}_1 \\ d\mathbf{A}_t = \left[\mathbf{f}_{2,t}(\mathbf{A}_t) - g_{2,t}^2 \nabla_{\mathbf{A}_t} \log p_t(\mathbf{X}_0, \mathbf{A}_t)\right] d\bar{t} + g_{2,t} d\bar{\mathbf{w}}_2 \end{cases},$$

that are sequential, because reverse function of $\mathbf{A}$ is determined by $\mathbf{X}_0$.

For the training score-based model should be trained to minimize the distance to the ground-truth partial scores. Transition distribution $p_{0t}(\mathbf{G}_t|\mathbf{G}_0)$ can be separated for $\mathbf{X}_t$ and $\mathbf{A}_t$ as:

$$p_{0t}(\mathbf{G}_t|\mathbf{G}_0) = p_{0t}(\mathbf{X}_t|\mathbf{X}_0) p_{0t}(\mathbf{A}_t|\mathbf{A}_0).$$

By leveraging the idea of denoising score matching to the partial scores and since the drift coefficient of the forward diffusion process is linear and as $p_{0t}(\mathbf{X}_t|\mathbf{X}_0)$ and $p_{0t}(\mathbf{A}_t|\mathbf{A}_0)$ are Gaussian distributions where mean and variance are determined by the coefficients of the forward diffusion process [31], authors proposed new training objectives:

$$\begin{cases} \min_\theta \mathbb{E}_t\{\lambda_1(t) \mathbb{E}_{\mathbf{G}_0} \mathbb{E}_{\mathbf{G}_t|\mathbf{G}_0} ||\mathbf{s}_{\theta,t}(\mathbf{G}_t) - \nabla_{\mathbf{X}_t} \log p_{0t}(\mathbf{X}_t|\mathbf{X}_0)||_2^2\} \\ \min_\phi \mathbb{E}_t\{\lambda_2(t) \mathbb{E}_{\mathbf{G}_0} \mathbb{E}_{\mathbf{G}_t|\mathbf{G}_0} ||\mathbf{s}_{\phi,t}(\mathbf{G}_t) - \nabla_{\mathbf{A}_t} \log p_{0t}(\mathbf{A}_t|\mathbf{A}_0)||_2^2\} \end{cases}$$

To estimate $\nabla \mathbf{A}_t \log p_t(\mathbf{X}_t, \mathbf{A}_t)$ score-based model $\mathbf{s}_{\phi,t}$ is presented as:

$$\mathbf{s}_{\phi,t}(\mathbf{G}_t) = \text{MLP}(\left[\{\text{GMH}(\mathbf{H}_i, \mathbf{A}_t^p)\}_{i=0,p=1}^{K,P}\right]),$$

where $\mathbf{A}_t^p$ are the high-order adjacency matrices, $\mathbf{H}_{i+1} = \text{GNN}(\mathbf{H}_i, \mathbf{A}_t)$ with given $\mathbf{H}_0 = \mathbf{X}_t$, $[\cdot]$ is the concatenation operation, and the GMH is graph multi-head attention block [32], K is the number of GMH layers. Another score-based model with the same dimensionality as $\mathbf{X}_t$ uses multiple layers of GNNs to learn partial scores from node representations:

$$\mathbf{s}_{\theta,t}(\mathbf{G}_t) = \text{MLP}(\left[\{\mathbf{H}_i\}_{i=0}^L\right]),$$

where $H_{i+1} = \text{GNN}(\mathbf{H}_i, \mathbf{A}_t)$ with given $\mathbf{H}_0 = \mathbf{X}_t$ and L is the number of GNN layer.

To use the reverse-time diffusion process as a generative model, it's needed to solve the system of reverse-time SDEs, approximated using trained score-based models $\mathbf{s}_{\theta,t}$ and $\mathbf{s}_{\phi,t}$:

$$\begin{cases} d\mathbf{X}_t = \mathbf{f}_{1,t}(\mathbf{X}_t) d\bar{t} + g_{1,t} d\bar{\mathbf{w}}_1 - g_{1,t}^2 \mathbf{s}_{\theta,t}(\mathbf{X}_t, \mathbf{A}_t) d\bar{t} \\ d\mathbf{A}_t = \mathbf{f}_{2,t}(\mathbf{A}_t) d\bar{t} + g_{2,t} d\bar{\mathbf{w}}_2 - g_{2,t}^2 \mathbf{s}_{\phi,t}(\mathbf{X}_t, \mathbf{A}_t) d\bar{t} \end{cases}$$

To solve these processes, authors proposed Symmetric Splitting for System of SDEs (S4) inspired by SSCS [33] and Predictor-Corrector Samlper [32]. For each discretized time step t, S4 solver first computes the estimation of partial scores using $\mathbf{s}_{\theta,t}$ and $\mathbf{s}_{\phi,t}$ models, then it uses them for correction and prediction. Correction step is performed by leveraging score-based MCMC method [34] to obtain corrected $\mathbf{G}'_t$ from $\mathbf{G}_t$. The prediction of the state at time t' follows the $p_{t'}$ marginal distribution. The propogation of the corrected step $\mathbf{G}'_t$ from time t to $t - \delta t$ can be approximated as:

$$e^{\frac{\delta t}{2} \hat{\mathcal{L}}_F^*} \mathbf{G} = \tilde{\mathbf{G}} \sim p_{t, t-\frac{\delta t}{2}}(\tilde{\mathbf{G}}|\mathbf{G}).$$

Chapter 5

Experiments

5.1 Experiments design

The main objectives of the experimental part are to compare the statistical properties of synthetic and original graphs, as well as to evaluate the possibility of using synthetic graphs to build a classification model. Therefore, the experiment can be divided into two parts: statistical tests and classification model tests.

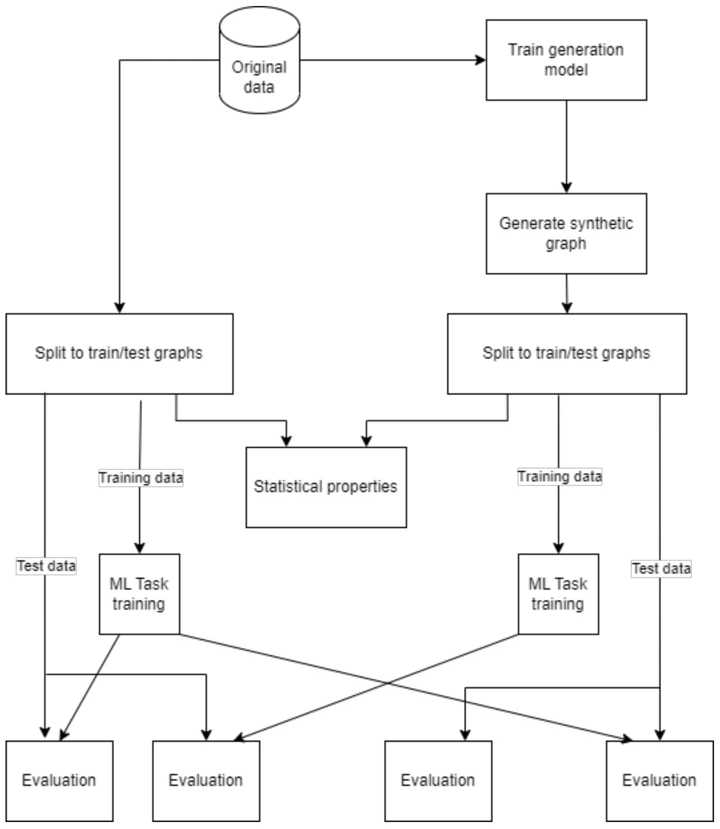


Figure 5.1 Experiment design diagram

First, the model is trained on dataset so that it can generate synthetic graphs. Then the statistical tests follows.

The statistical test involves different measurements. To be objective and to test the consistency of the graph generation model, graphs are re-generated 10 times and compared with the original. Degree, harmonic and betweenness centralities were chosen for graphs comparison. For each centrality histogram and boxplot are built. Each graph represents dataset, which contains graphs as components.

To train the classification model, it is necessary to translate the graph into a suitable representation, for this the RiWalk algorithm is used, which is a feature learning method, allowing to build feature vectors based on the structural specificity of the graph.

Most of the currently existing techniques are narrowly focused on the generation of chemical compounds, or structurally similar graphs without labeled nodes. The techniques presented in this part allow generating graphs with node labels, which makes it possible to evaluate the classification capabilities of models trained on the synthetic graphs, however, the structure of graphs is not always directly related to labels of graph vertices, which is why it is not possible to build the necessary dependence when training classification models. In this regard, with imbalanced data in the dataset, the classifier will be trained so that in most cases it will predict the value of the majority class, which makes it biased to use such a metric as accuracy score, therefore, F1-score[35] is used for experiments with the classifier.

The weighted Random Forest [36] model is used as a classifier, since this model is one of the most successful when training on imbalanced data.

In classification tests, graphs also are generated repeatedly. The main goal of this part is to compare classification model's F1-score trained on synthetic data with the same model trained on the original data. Train data is divided into training and test datasets in the proportion of 75/25. Hyperparameters of the classifictaion model are selected at the first iteration, and then used in further.

5.2 Labeled Graph RNN

Labeled Graph RNN is an extended version of GraphRNN, which uses the idea of MolecularRNN and also capable of storing node labels and edge labels as well.

5.2.1 Datasets

Experiments for this technique were carried out with citation and yeast datasets. In citation dataset, graph represents citation network, where each node's label represents the topic of cited work, and edges represent a citation. Yeast dataset consists of different yeast structures, where each node represents atom and edges represent bonds.

5.2.2 Citation dataset statistics

The original graph is represented by a hundred combined randomly selected graphs from the original dataset, synthetic graphs are generated by the trained model, each of them also represented by a hundred merged graphs. As a result, after 10 generations of graphs, the comparison of their statistical indices was made.

Let's start with the centrality measures. Degree centrality allows us to consider the importance of vertices in terms of the number of their neighbors. Consider the indicators of the original graph, as we can see on 5.2, the degree centrality of the greatest number of nodes is at about 0.0003, then as the centrality measure increases, the number of nodes begins to decrease. This indicates that most nodes in the graph are connected to few other nodes, in other words, they have few neighbors.

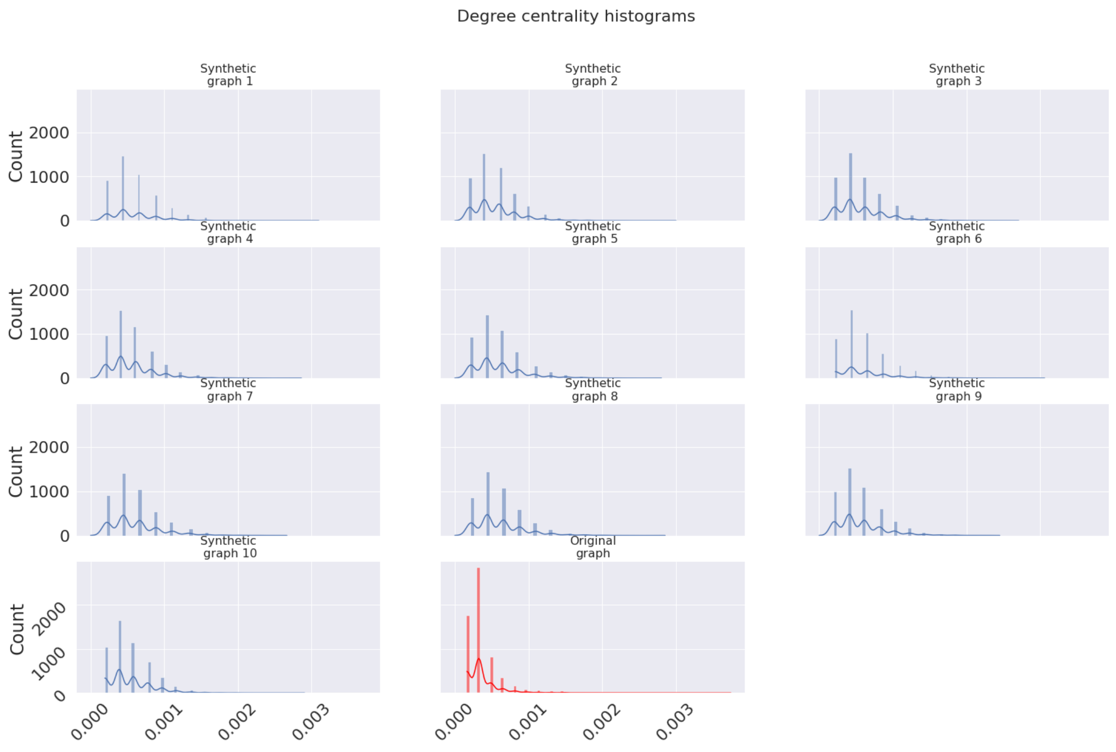


Figure 5.2 Graph Labeled RNN degree centrality histograms (citation dataset).

Now let's pay attention to synthetic graphs, in comparison with each other, indicators have insignificant differences, and have a general tendency, as in the original graph, most of the nodes of graphs have a small degree centrality, with the increase of which the number of nodes decreases. Hence, we can conclude that the generation model works consistently.

If we compare the values of synthetic and original graphs, we can see that they have a similar distribution, the main difference being the number of nodes with minimal values, in the original graph, most of the nodes have minimal values, and then their number decreases rapidly, while in the synthetic, the number of these nodes is much smaller, and it decreases more slowly.

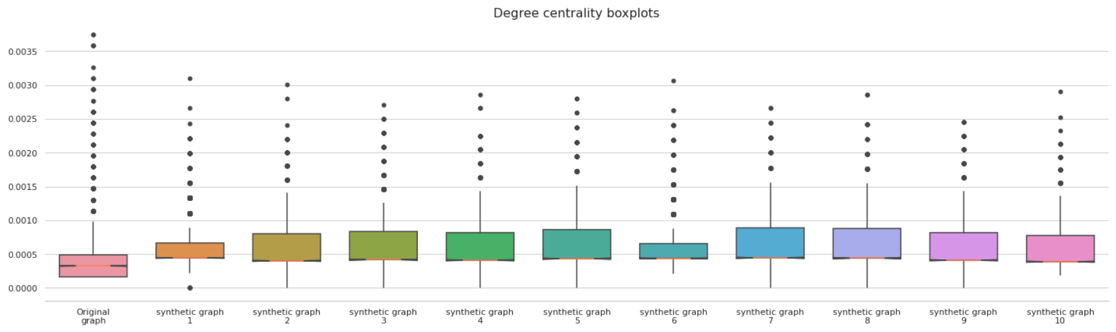


Figure 5.3 Graph Labeled RNN degree centrality boxplots (citation dataset).

We can see this if we look at the boxplots 5.3. The medians of the synthetic graphs are slightly larger than those of the original graph, their values are closer to 0.0005 than 0.0003 of the original graph. The number of vertices less than the median is greater in the original graph, while in synthetic graphs significantly more vertices have values above the median.

This is an evidence of slower decrease of the number of vertices depending on degree centrality. Also we can observe the difference in extreme values, the minimum of some graphs reaches zero, while the maximum value of most synthetic graphs is greater than that of the original. After reviewing the degree centrality indicators, we can conclude that in synthetic graphs, nodes on average have more neighbors than the nodes of the original graph.

Using harmonic centrality, we will be able to find out more detailed information about the neighbors of nodes in graphs, how far are they from each other.

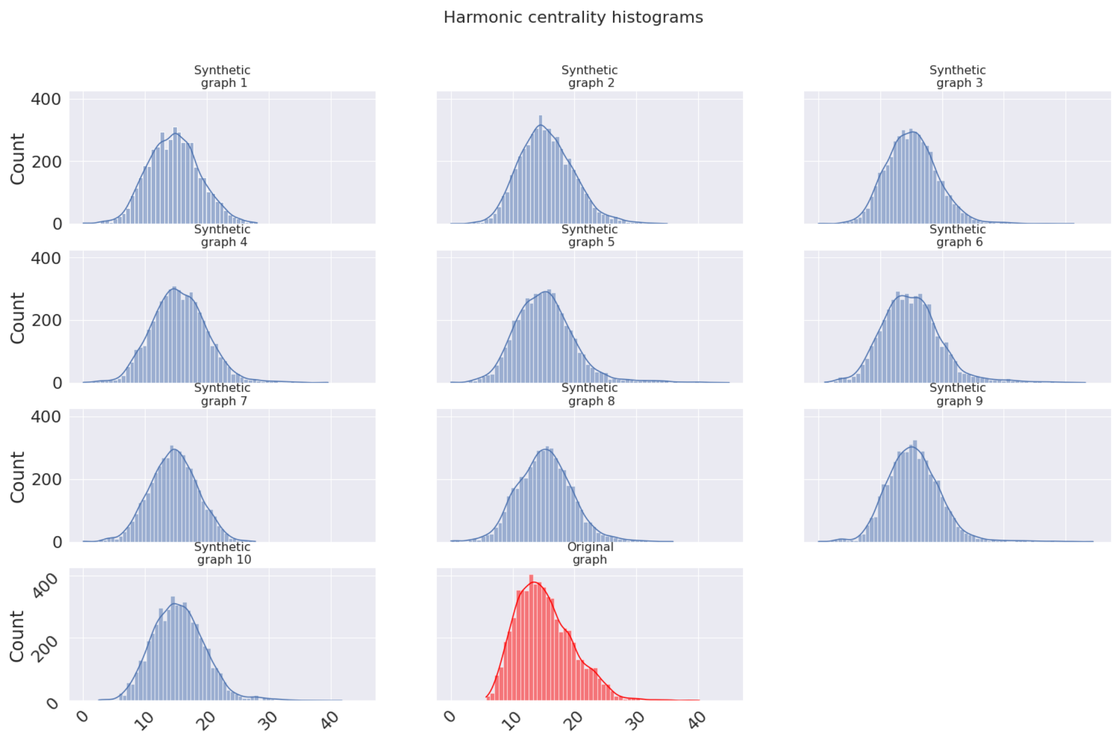


Figure 5.4 Graph Labeled RNN harmonic centrality histograms (citation dataset).

Here 5.4 we can observe a similar situation as with degree centrality, in synthetic graphs there are more minimum values of harmonic centrality, as well as the decrease in indicators is slower than in the original graph.

If we look at boxplot 5.5, we will see that, unlike degree centrality, the harmonic centrality indicators of synthetic graphs are very similar to those of the original graph, but as already noted above, synthetic graphs have a larger lower bound, which means a large number of nodes located at a far distance from other nodes of the graph.

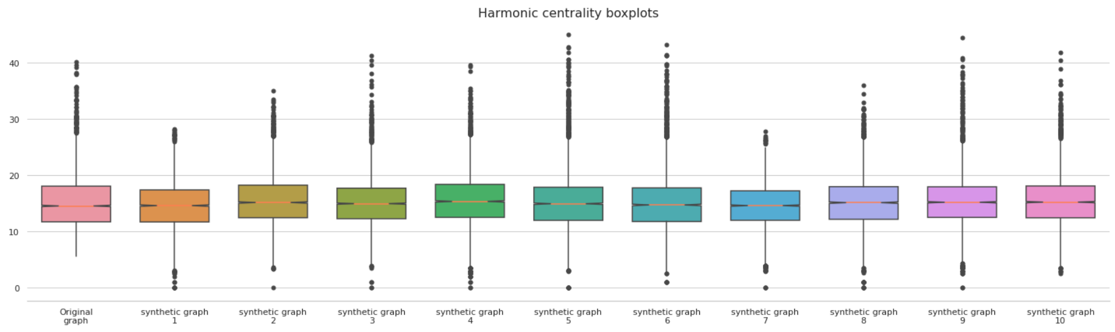


Figure 5.5 Graph Labeled RNN harmonic centrality boxplots (citation dataset).

Betweenness centrality indicates how much a node is involved in the paths between all nodes of the graph. Paying attention to the previous metrics, here 5.6 we also see the similarity between the indicators of synthetic and original graphs, the difference again is the rate of decrease in betweenness centrality indicators, but in this case, we also observe that synthetic graphs have a maximum value greater than the original graph.

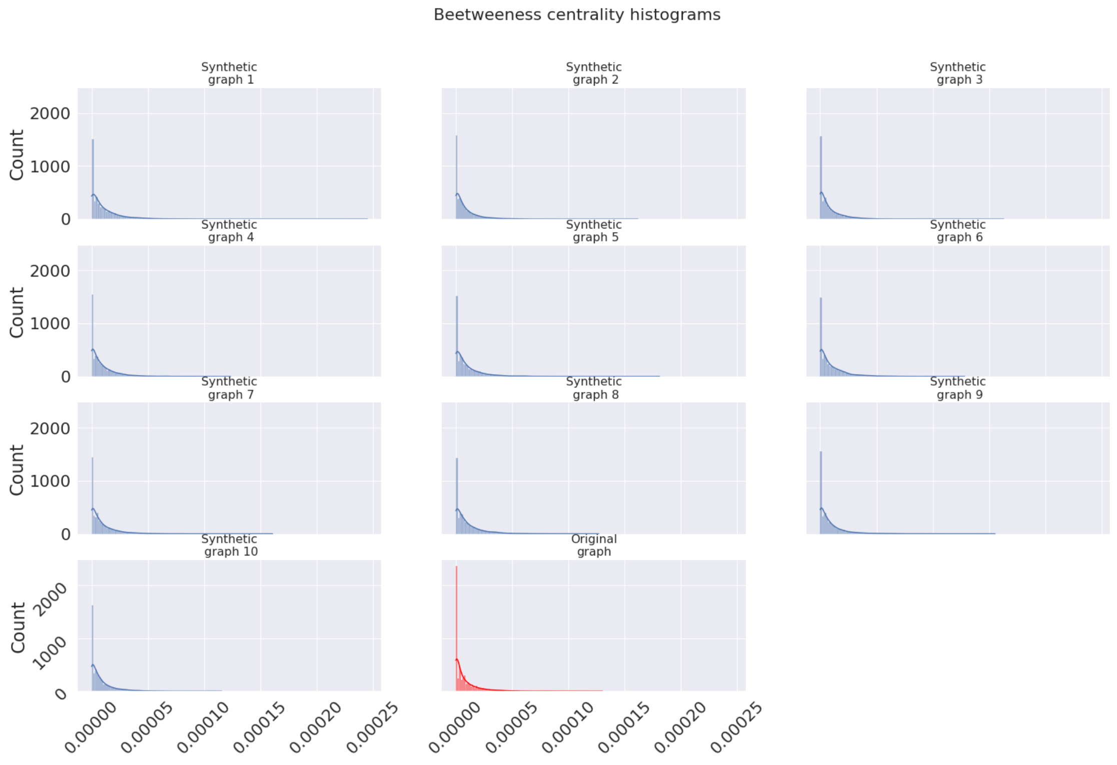


Figure 5.6 Graph Labeled RNN betweenness centrality histograms (citation dataset).

The difference in medians and upper values is clearly visible on the boxplot 5.7, the indicators of synthetic graphs are greater than those of the original graph.

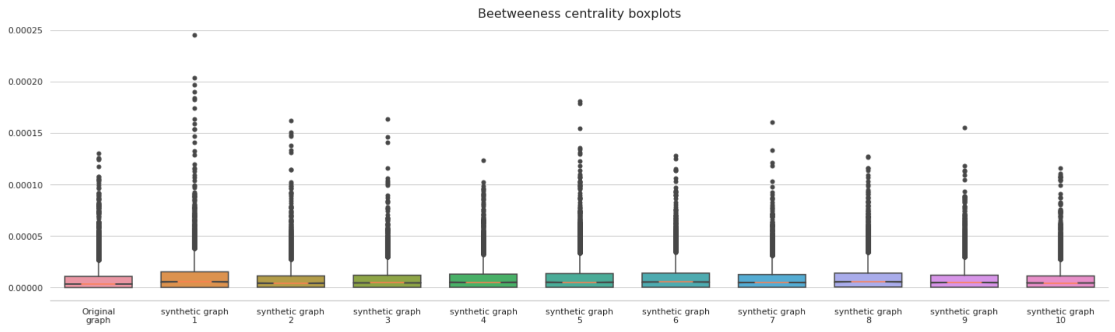


Figure 5.7 Graph Labeled RNN betweenness centrality boxplots (citation dataset).

Now let's consider the sizes of the graphs, for this we just need to compare the sizes of their components, since each graph is a disconnected graph of smaller graphs.

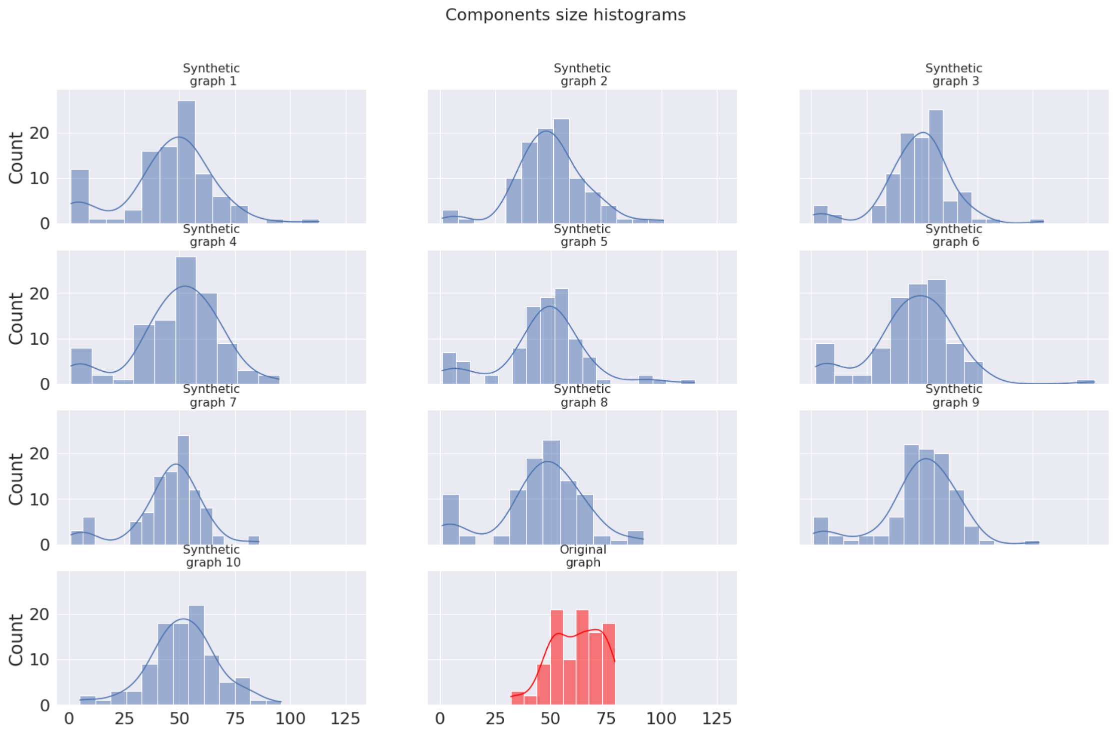


Figure 5.8 Graph Labeled RNN graph component sizes (citation dataset).

According to 5.8, synthetic graphs are more diverse in size, while the original graphs contain from 30 to 80 nodes, synthetic graphs can be both very small and quite large, this explains the difference in the indicators of previous measurements.

5.2.3 Citation dataset classification tests

In this part of the experiment were performed two repeated tests, in the first one the original data was used, in the second one the synthetic data, and then the obtained results were compared.

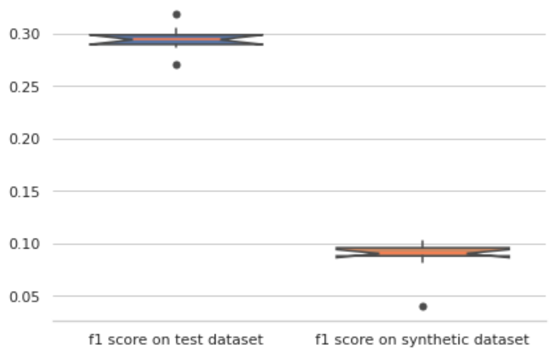


Figure 5.9 Graph Labeled RNN citation dataset, model trained on original training dataset.

As you can see, the model trained on the original graphs almost always has the same score, which is a consequence of the problem of unbalanced data, and this affects synthetic data similarly.

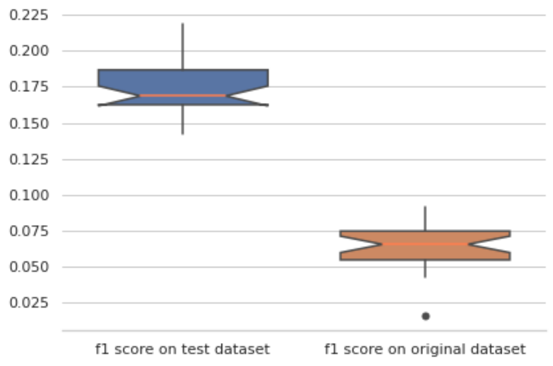


Figure 5.10 Graph Labeled RNN citation dataset, model trained on synthetic training dataset.

The classifier trained on the synthetic graphs shows the worse score, but we can also notice that the score now varies more, and this affects the variability of the score on the original data and the difference between datasets in both cases is proportional.

5.2.4 Yeast dataset statistics

In this dataset, we can notice, that the degree centrality of synthetical graphs closer to the original.There is still a tendency to increase the values, but the median is closer to the original ones, this may be due to the nature of graphs, since the compounds of atoms have a more systematic structure.

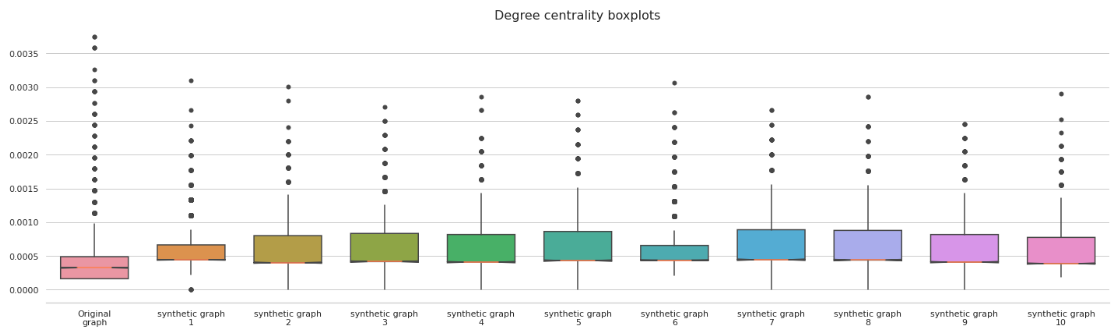


Figure 5.11 Graph Labeled RNN degree centrality boxplots (yeast dataset).

The same can be said about betweenness centrality indicators.

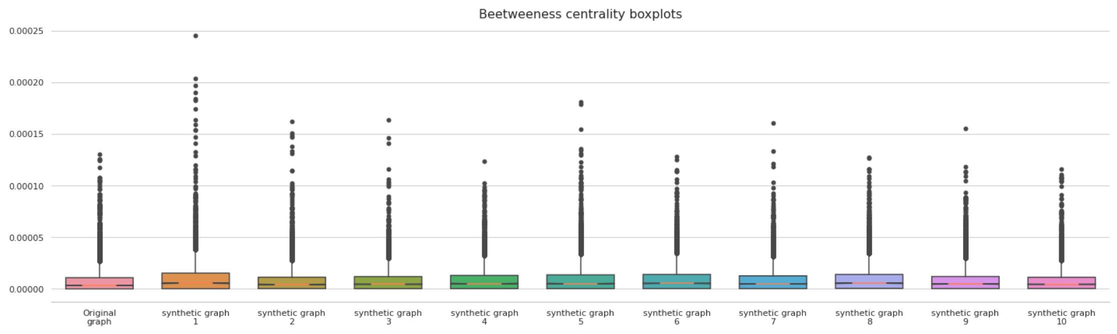


Figure 5.12 Graph Labeled RNN beetweeness centrality boxplots (yeast dataset).

The sizes of synthetic graphs vary quite a lot, but on average correspond to the sizes of the original ones.

The other indicators remained about the same.

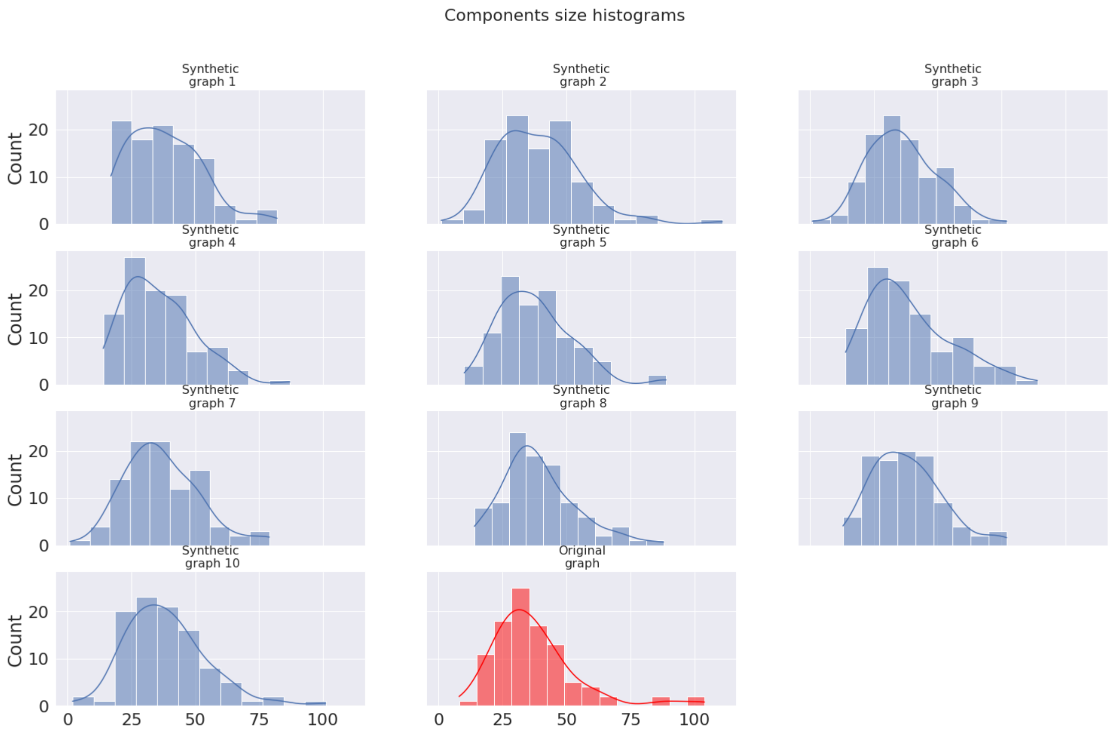


Figure 5.13 Graph Labeled RNN graph component sizes (yeast dataset).

5.2.5 Yeast dataset classification tests

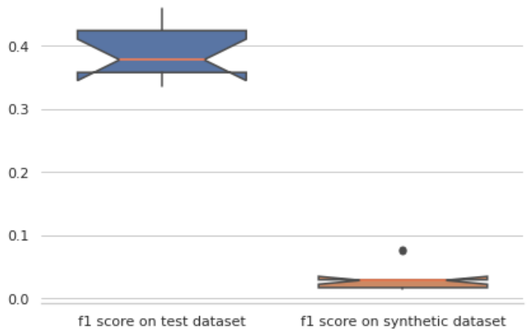


Figure 5.14 Graph Labeled RNN yeast dataset, model trained on original training dataset.

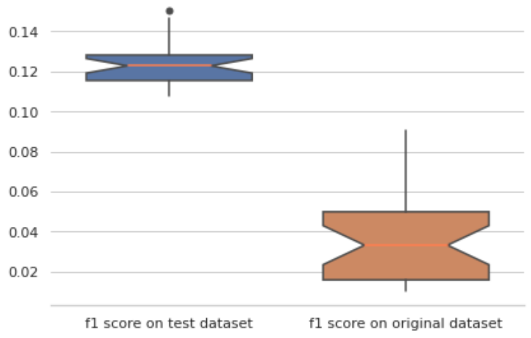


Figure 5.15 Graph Labeled RNN yeast dataset, model trained on synthetic training dataset.

Here it can be noted that in both cases, the score is low, which is again a consequence of the nature of the graph, since organic compounds have frequently occurring atoms that are bound to a small number of other atoms, which is the reason for the imbalance of classes in dataset. The classifier trained on the original graphs shows the best result on test data, but on synthetic data the score is almost zero.

5.3 GDSS

5.3.1 Datasets

For this technique QM9 molecular dataset is used.

5.3.2 QM9 statistics

Degree centrality of synthetical graphs are similar to original, but maximal values are bigger.

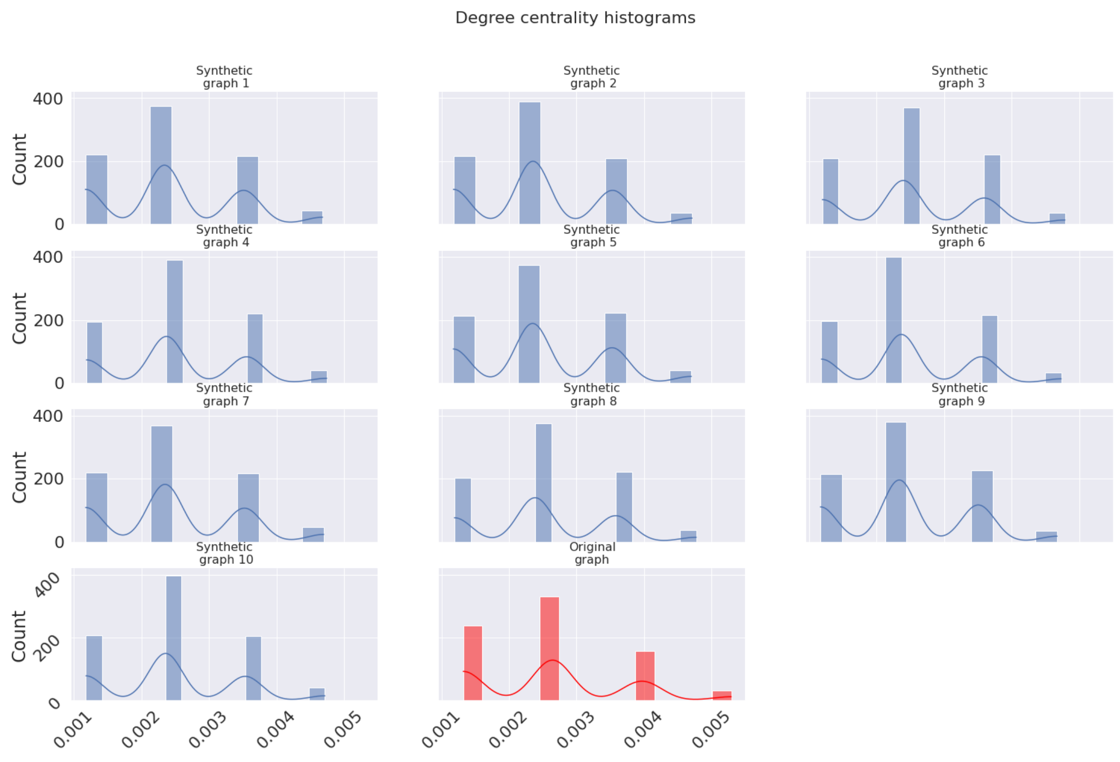


Figure 5.16 GDSS degree centrality histogram.

This can be seen especially on boxplot, the median of synthetic graphs is close to the original ones, but it can be noticed that the upper bound of synthetic graphs is higher, it is also noticeable that in half of the cases the lower bound is also significantly higher than that of the original graphs.

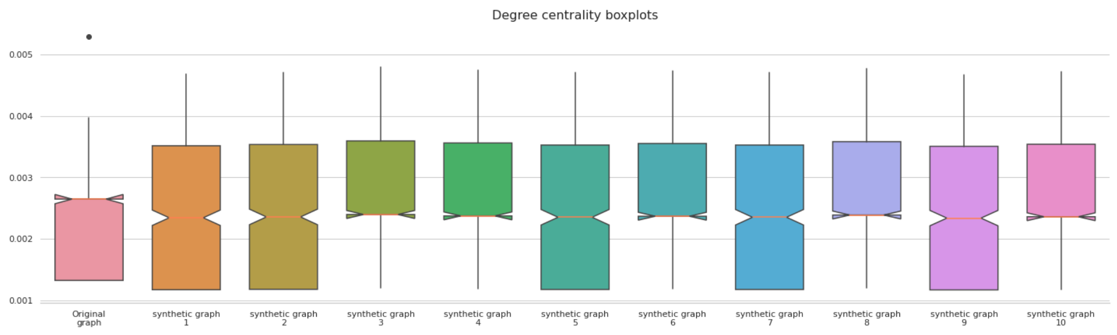


Figure 5.17 GDSS degree centrality boxplot.

Harmonic centrality is quite close to the original, but the values of synthetic graphs are higher, both in maximum values and in median.

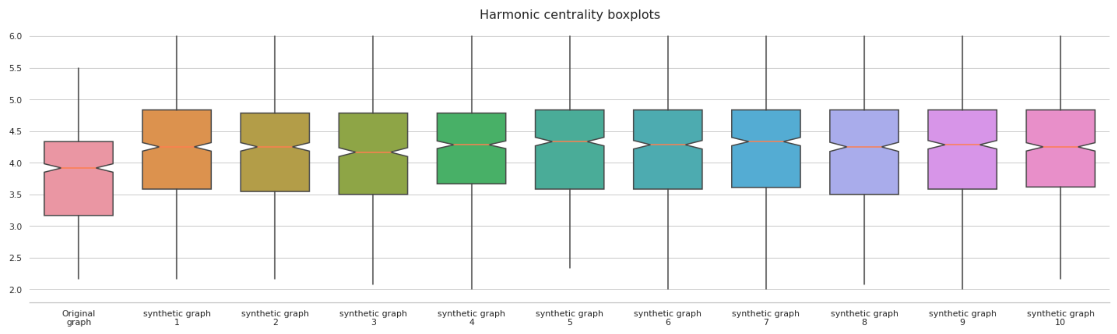


Figure 5.18 GDSS harmonic centrality boxplot.

At the same time, the betweenness centrality of synthetic graphs is very close to the original ones, which may be the result of the fact that the sizes of the molecules are quite small and all atoms are strongly connected in the graphs.

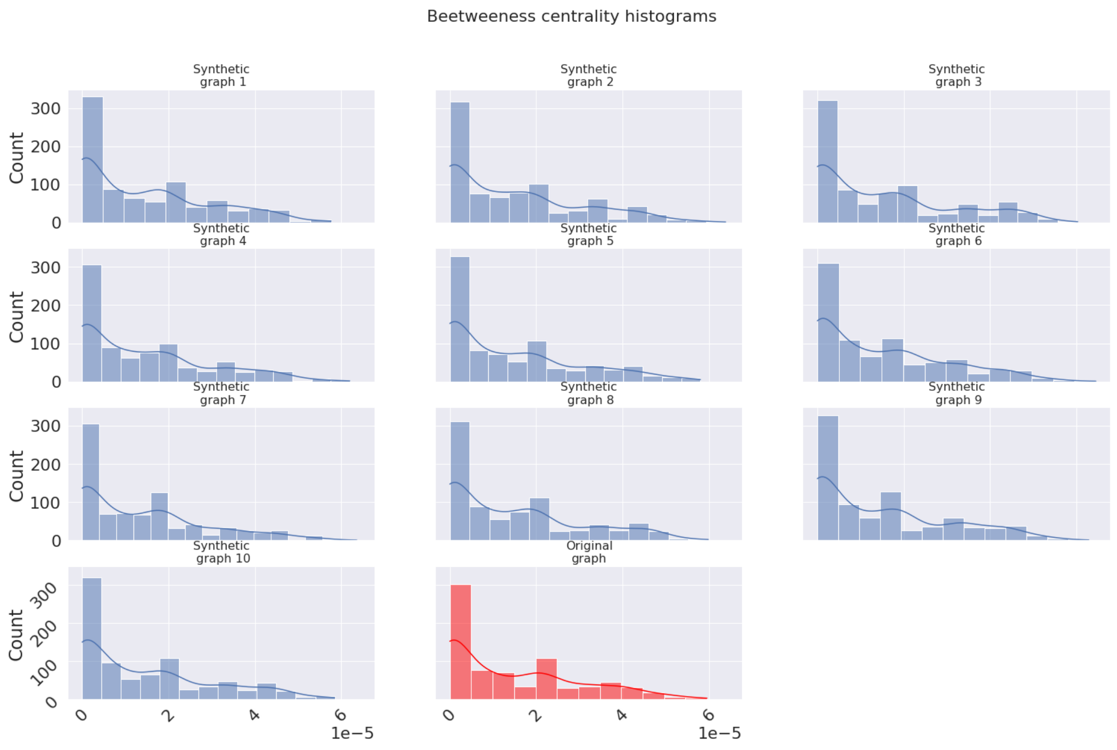


Figure 5.19 GDSS betweeness centrality histogram.

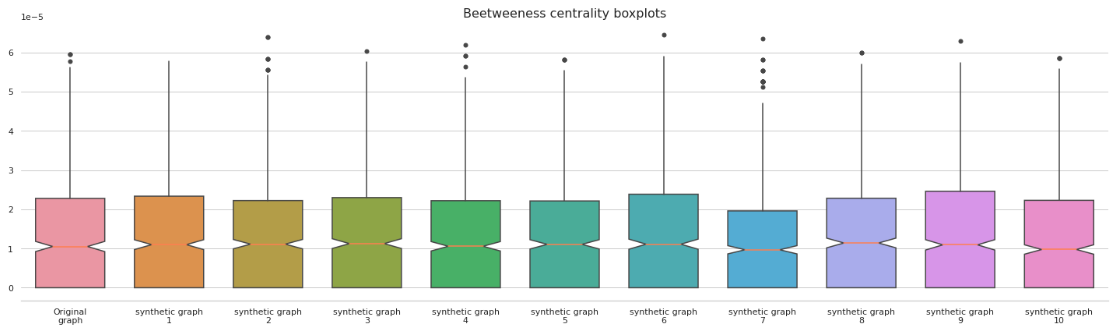


Figure 5.20 GDSS betweeness centrality boxplot.

As expected, the graph sizes are quite small, but the model generates slightly more diverse graphs, although most graphs still have a size of 8-9 nodes.

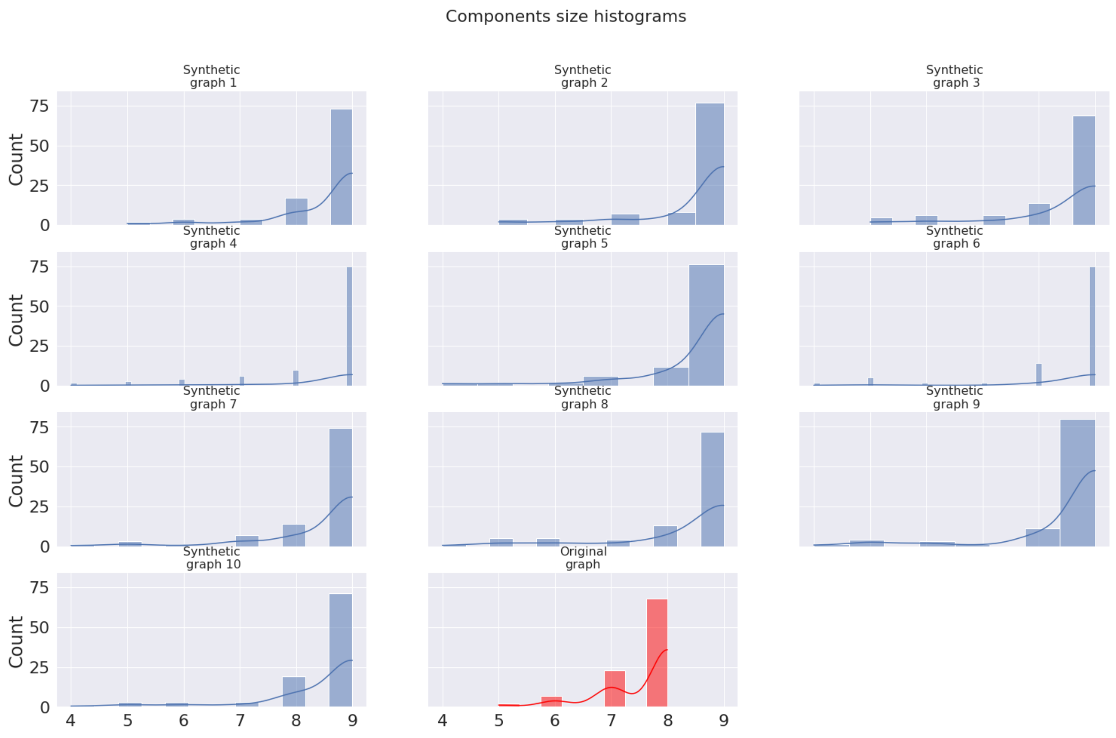


Figure 5.21 GDSS component sizes histogram.

5.3.3 QM9 classification tests

In the classification part we can notice, that in both cases, score difference is proportional, classificator trained on the original data has better score and score distribution, then the model trained on synthetic dataset, which in some cases shows a 7 % drop in the score.

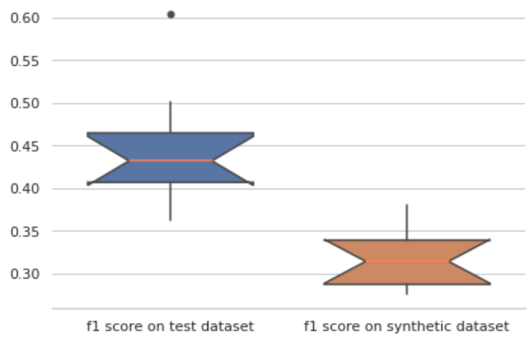


Figure 5.22 GDSS model trained on original training dataset.

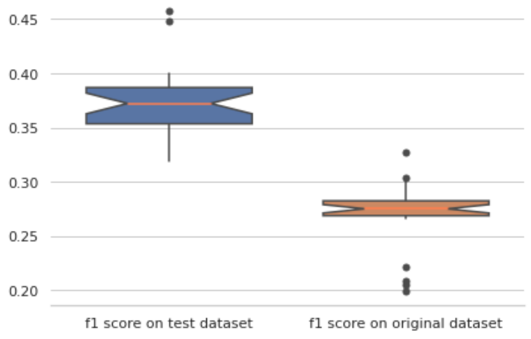


Figure 5.23 GDSS model trained on synthetic training dataset.

5.4 GraphGen

5.4.1 Datasets

For this model there are CiteSeer and ENZYMES datasets. CiteSeer dataset consists of papers with one of the following classes:

- Agents
- AI
- DB
- IR
- ML
- HCI

And the ENZYMES dataset represents enzymes molecular dataset. For an additional task, there is Cora dataset, which was used to create bigger training graphs by RandomWalk algorithm. There are 7 classes in this dataset:

- Case Based
- Genetic Algorithms
- Neural Networks
- Probabilistic Methods
- Reinforcement Learning
- Rule Learning
- Theory

5.4.2 CiteSeer statistic tests

For this dataset, to see the impact of training, this model was tested in two types, trained on 1000 epochs and on 20. The graphs in current original dataset were created by random walk procedure, from one CiteSeer graph.

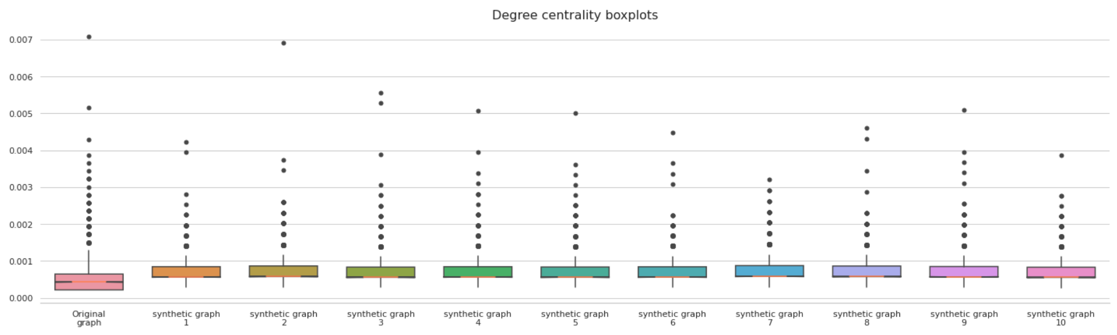


Figure 5.24 Graphgen high-trained degree centrality boxplots.

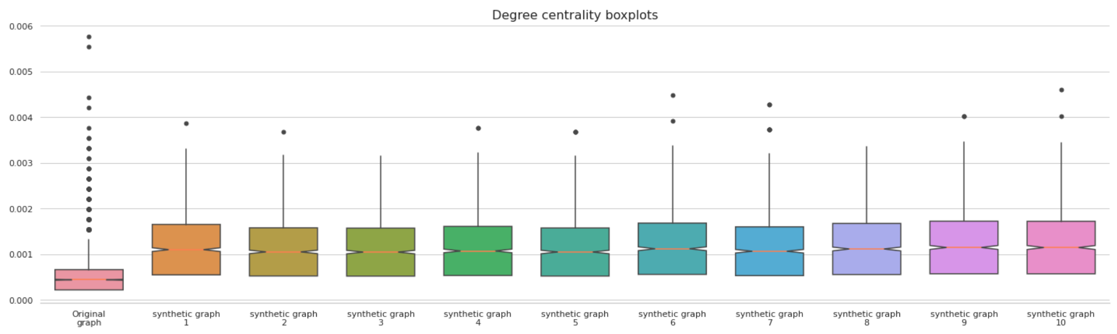


Figure 5.25 Graphgen low-trained degree centrality boxplots.

These boxplots show that a high-trained model generates graphs much closer in values to original graphs, while synthetic graphs of a low-trained model have indicators with a much larger spread and with large values.

Let's see if this trend will continue for the rest of the metrics.

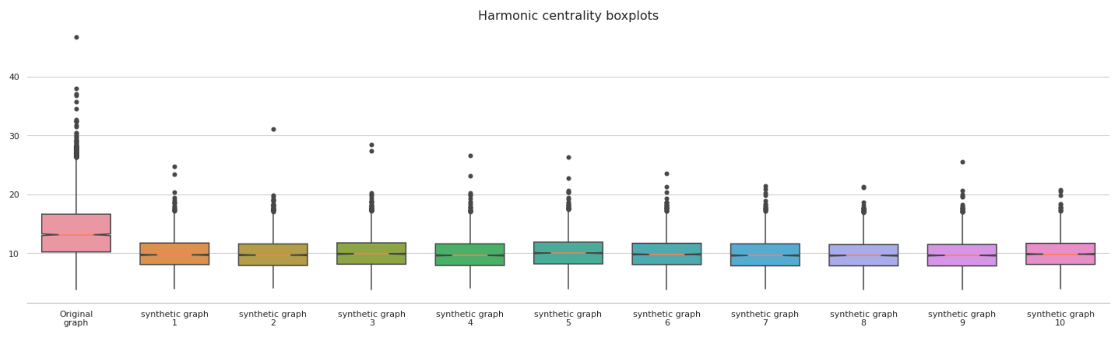


Figure 5.26 Graphgen high-trained harmonic centrality boxplots.

For the harmonic centrality measure there is the same situation, after training the indicators have become 5 units closer to the indicators of the original graphs. The synthetic graphs themselves, in any case, have lower indices, which indicates that there are fewer nodes in them that have a large number of neighbors.

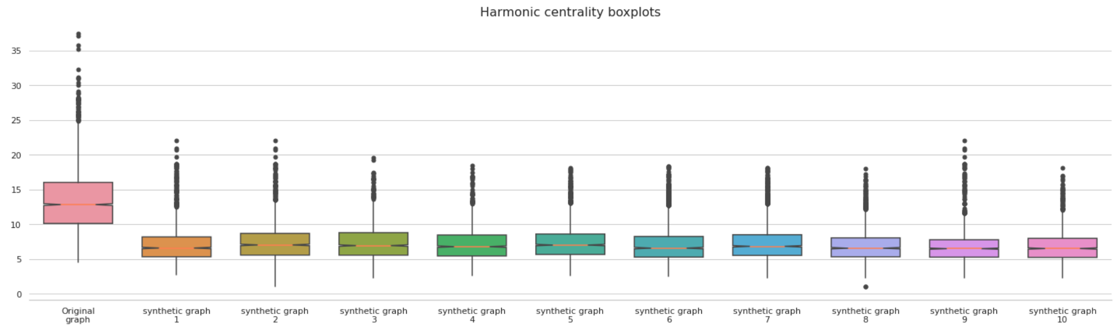


Figure 5.27 Graphgen low-trained harmonic centrality boxplots.

And if we look at betweenness centrality, there is no much difference between models.

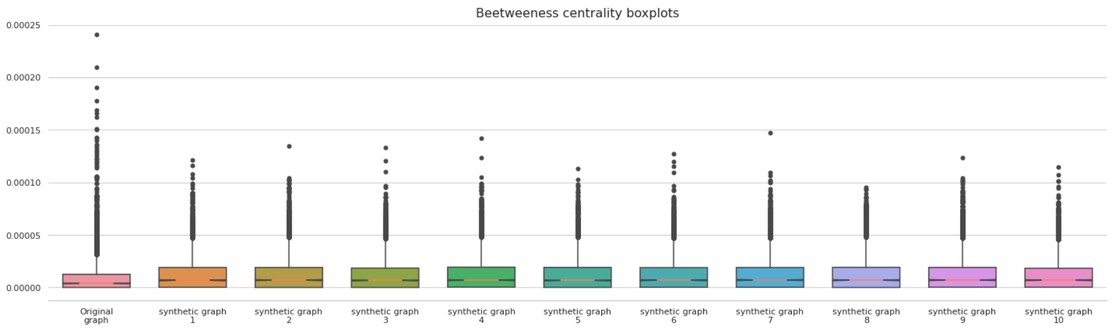


Figure 5.28 Graphgen high-trained betweenness centrality boxplots.

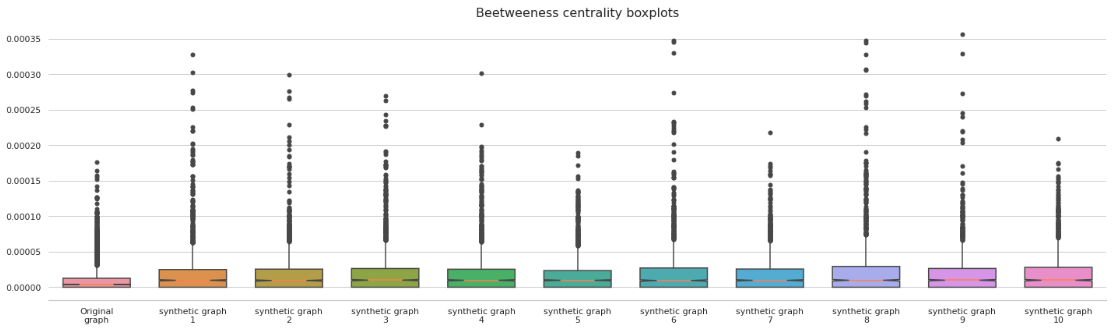


Figure 5.29 Graphgen low-trained betweenness centrality boxplots.

Finally, let's consider how much the sizes of synthetic graphs vary.

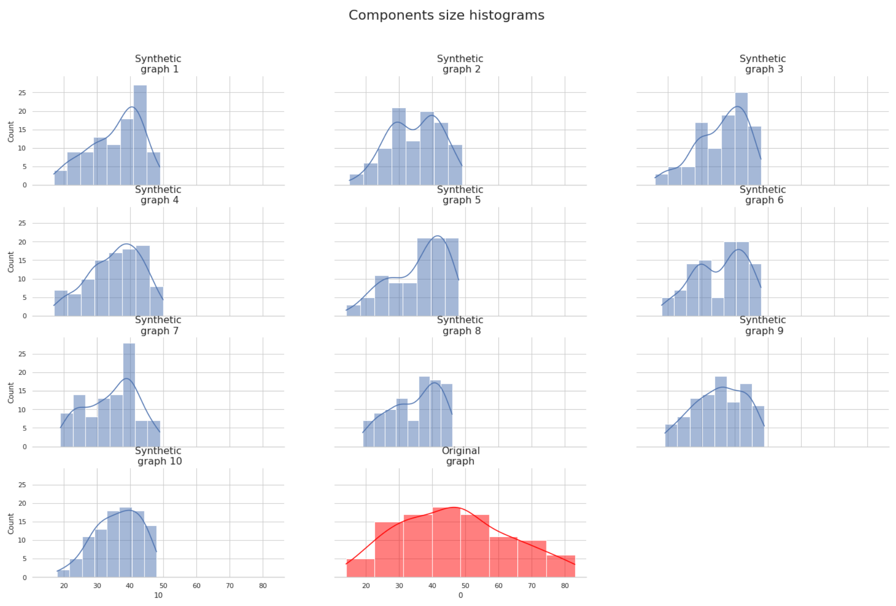


Figure 5.30 Graphgen high-trained component sizes hists.

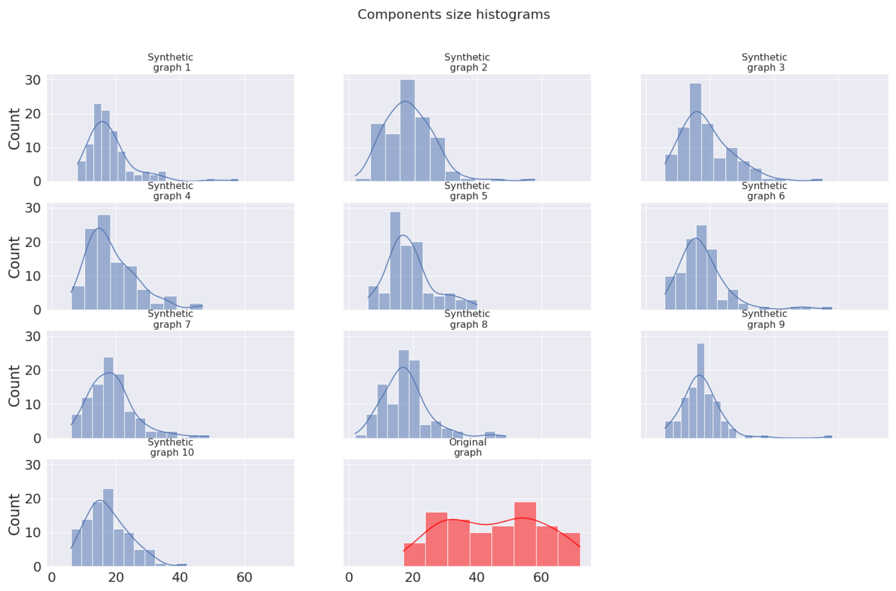


Figure 5.31 Graphgen low-trained component sizes hists.

Here we can see, that low-trained model generates more diverse graphs, while a high-trained model has a large number of medium-sized graphs, and interestingly, the sizes of these graphs do not exceed 50 nodes, what is the result of the conditions of the technique that allows you to limit the number of nodes and edges of synthetic graphs.

5.4.3 CiteSeer classification tests

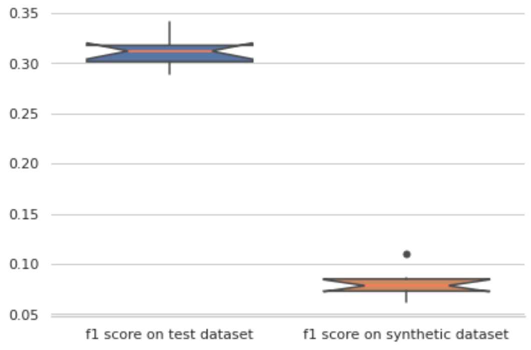


Figure 5.32 Graphgen high-trained model, classificator trained on original training dataset.

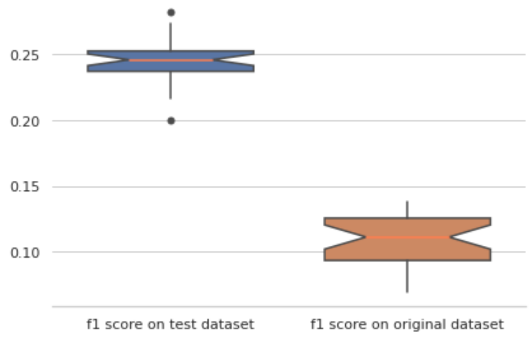


Figure 5.33 Graphgen high-trained model, classificator trained on synthetic training dataset.

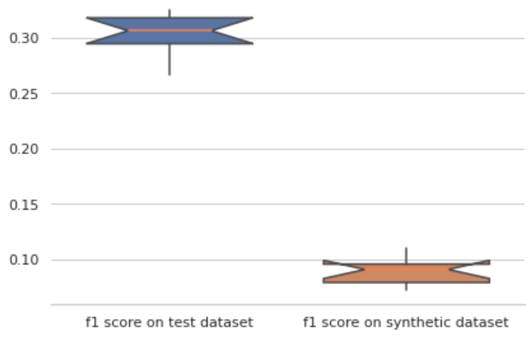


Figure 5.34 Graphgen hlow-trained model, classificator trained on original training dataset.

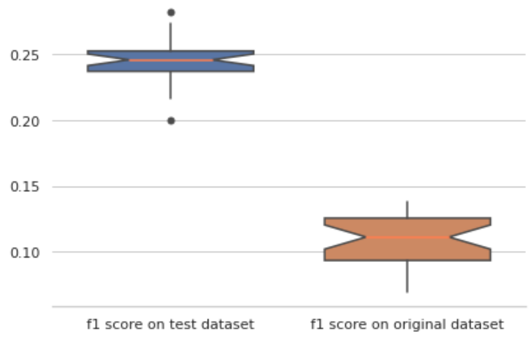


Figure 5.35 Graphgen low-trained model, classificator trained on synthetic training dataset.

In the classification task, we can notice that since embedding has a direct impact on the training of the classifier, a lot depends on what kind of structure the graph has and how it is related to node labels. That is why we can notice that when training the classifier on the original graphs, the result is different, while when training on synthetic data, the result remains the same, which indicates that in synthetic graphs, the graph structure cannot be linked by the embedding model with node labels.

5.4.4 ENZYMES dataset

Statistically, the tests on this dataset did not differ in anything special, so I will describe only the classification part.

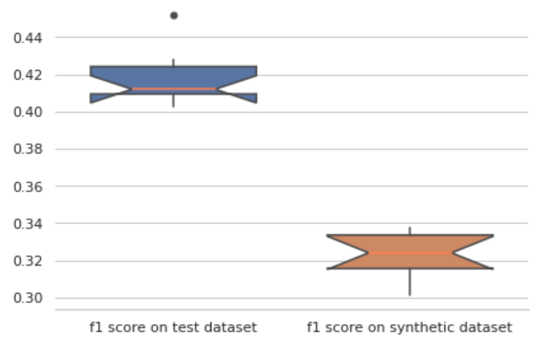


Figure 5.36 Graphgen enzymes dataset model, classificator trained on original training dataset.

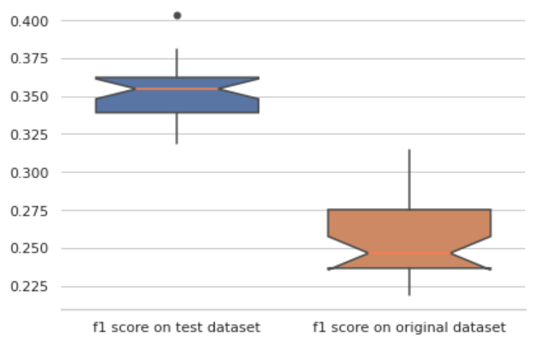


Figure 5.37 Graphgen enzymes dataset model, classificator trained on synthetic training dataset.

In this classification task, you can see that in both cases the score is about the same, for a model trained on synthetic graphs it is lower by about 7 %, it is also clear that in the case of this dataset, embedding copes better, and the relationship between the structure and node labels of the graph is more explicit.

5.4.5 Cora large graphs dataset

For this task for each test, datasets consist of 50 graphs instead of 100.

Degree histograms shows, that synthetic graphs has bigger degree than original, but have the similiar indicator distribution.

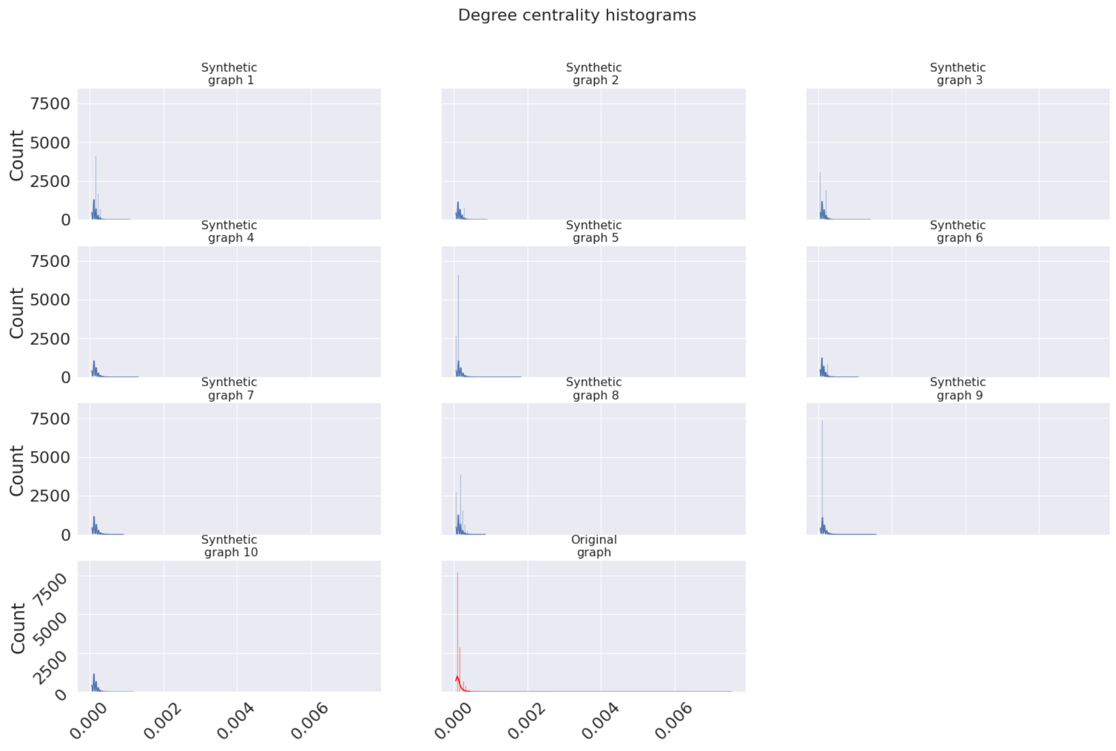


Figure 5.38 Graphgen cora big graphs dataset degree centrality histograms.

Harmonic centrality of synthetic graphs has lower values on average than that of the original graphs.

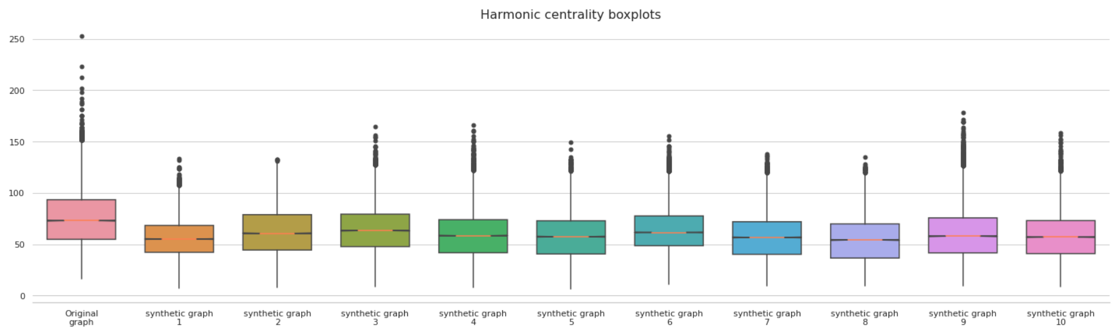


Figure 5.39 Graphgen cora big graphs dataset harmonic centrality boxplots.

At the same time betweenness centrality differs slightly.

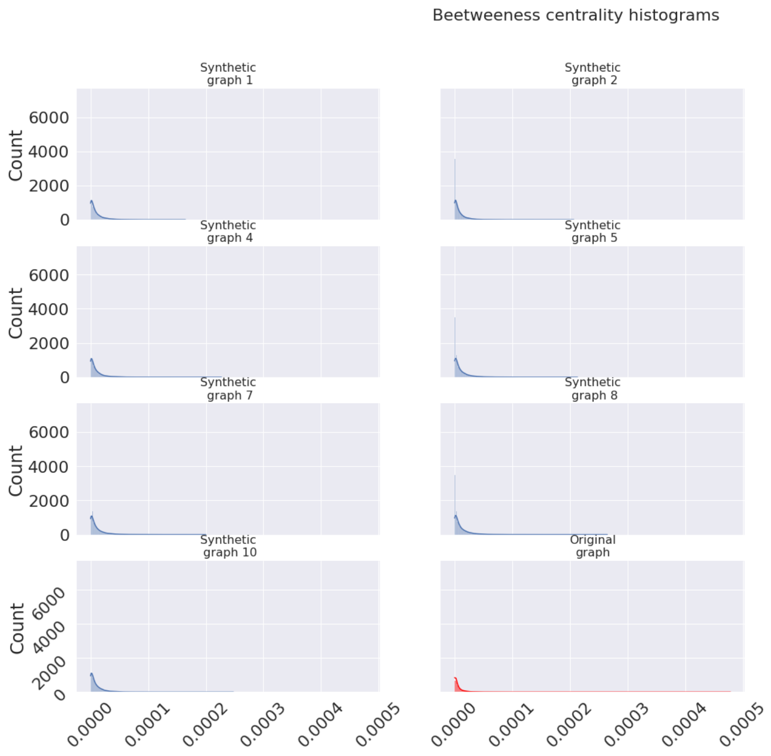


Figure 5.40 Graphgen cora big graphs dataset beetweeness centrality histograms.

Most synthetic graphs have size equal to the minimum or maximum possible, because they cannot exceed the size of the graphs on which the model was trained.

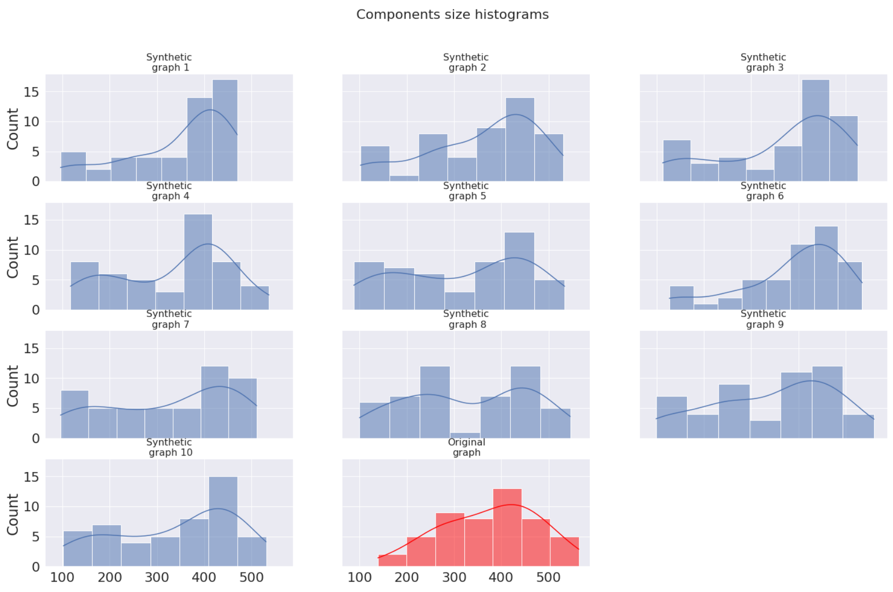


Figure 5.41 Graphgen cora big graphs dataset component sizes histograms.

5.4.6 Summary

Statistically, synthetic graphs are close to the properties of the original ones, using the GraphGen technique as an example, the influence of learning on the generative model was demonstrated, and the influence of the size of the original graphs on the statistical properties of synthetic ones was also demonstrated. Unfortunately, these techniques do not able to generate graph features and are limited only by the generation of node labels, but the graph structure does not always have a direct relationship with node labels, what complicates the task of training the classifier. Some of the non-received materials can be found in the appendix. Implementation of the experiments can be found on github `https://github.com/kirking/fit_ctu_thesis/tree/main`

Chapter 6 Conclusion

In this thesis, I have considered several representative techniques in the field of synthetic graph generation. I have conducted statistical tests to assess how well synthetic graphs preserve the properties of graphs on which the generative model was developed. Experiments were also conducted with the possibility of using classification models in conjunction with structural embedding models.

The choice of techniques was based on the ability of the model to generate graphs with labeled nodes, in order to test how possible it is to use the classifier on synthetic graphs. Since most of the currently existing models are aimed at generating small graphs, such as molecular structures, the choice of techniques that could generate large graphs and at the same time be able to generate their node labels is quite modest. The biggest problem in the task of evaluating the capabilities of classifiers trained on synthetic graphs is the weak dependence of the values of node labels on the graph structure, which affects the ability of the classifier to learn. This problem does not allow using such methods of dealing with data imbalance as SMOTE [37], since features obtained using embedding models cannot sufficiently represent node labels. Also the possibilities of models to preserve the statistical properties of the original graphs were evaluated. The used models, as experiments show, cope with this task quite well, although they are still relatively far from ideal.

This thesis can serve as a basic introduction to the problems of generating synthetic graphs through artificial neural networks, as well as an experiment in using structural embedding models for classification problems with graphs.

In the future, this work can be expanded by using other metrics for evaluating statistical properties of graphs, as well as using other techniques and evaluating classifiers, the main purpose of which is to determine the graph label based on it's structure.

Appendix A

Graph statistics

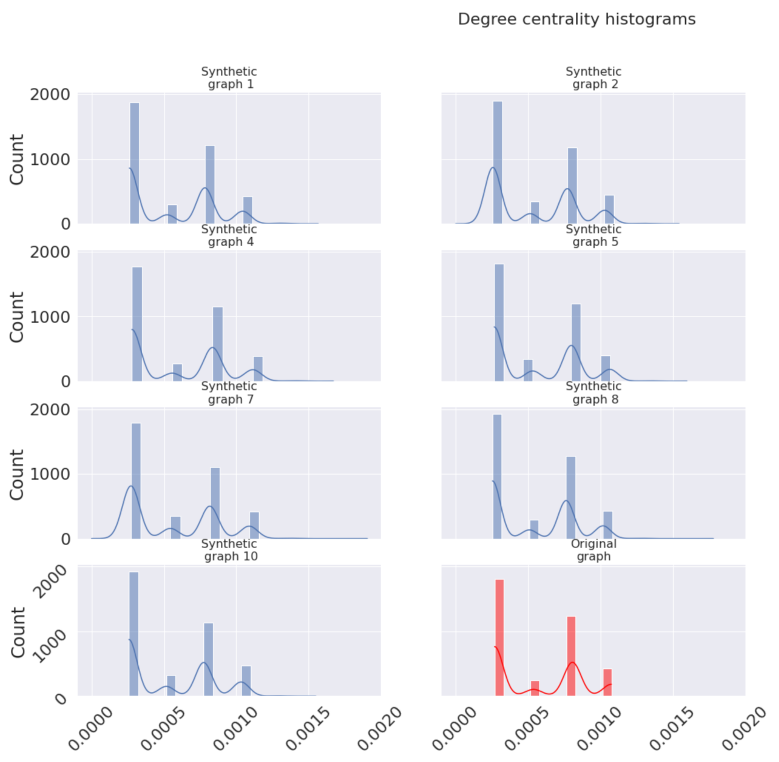


Figure A.1 Labeled RNN yeast dataset degree centrality histograms.

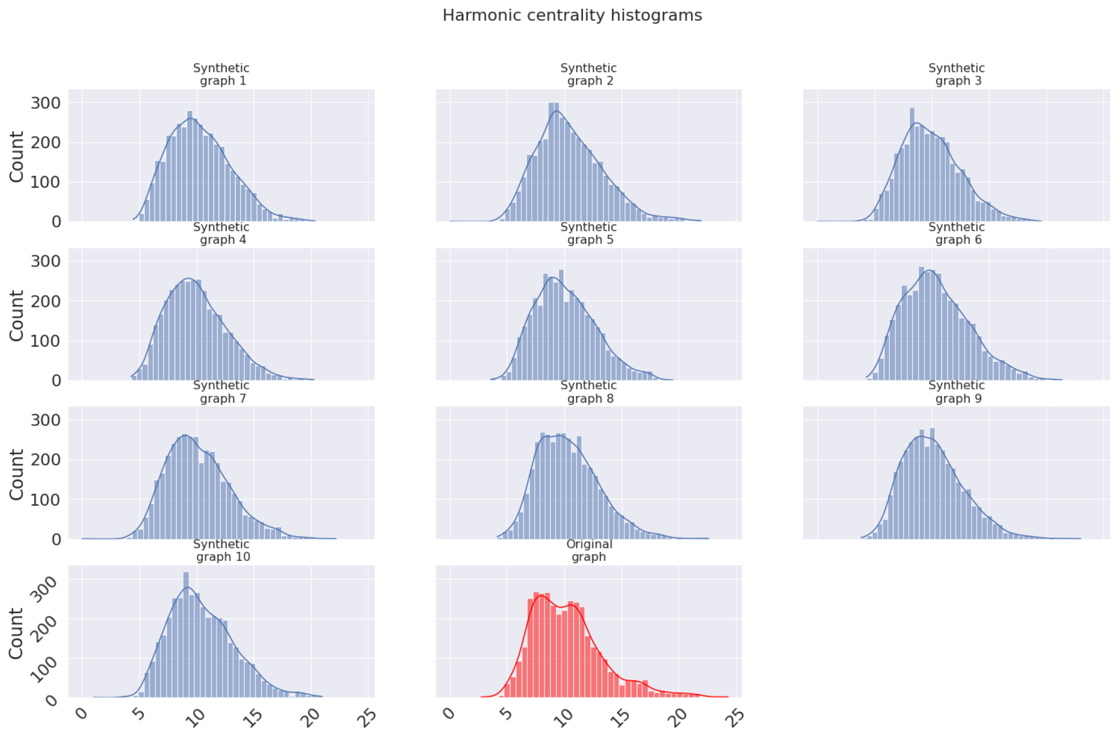


Figure A.2 Labeled RNN yeast dataset harmonic centrality histograms.

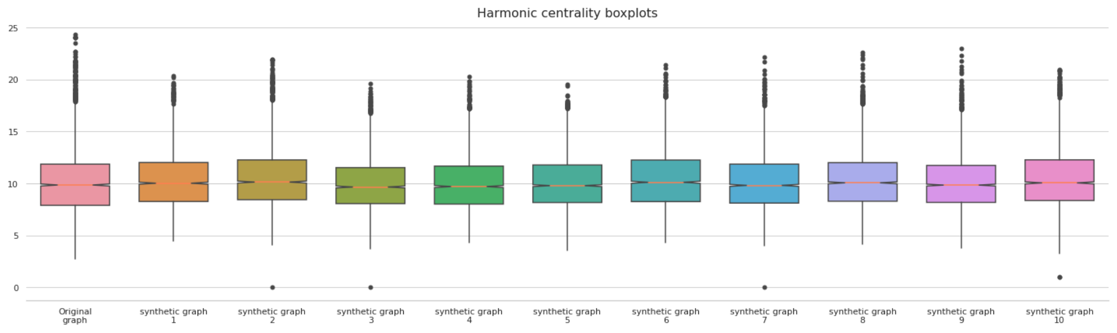


Figure A.3 Labeled RNN yeast dataset harmonic centrality boxplot.

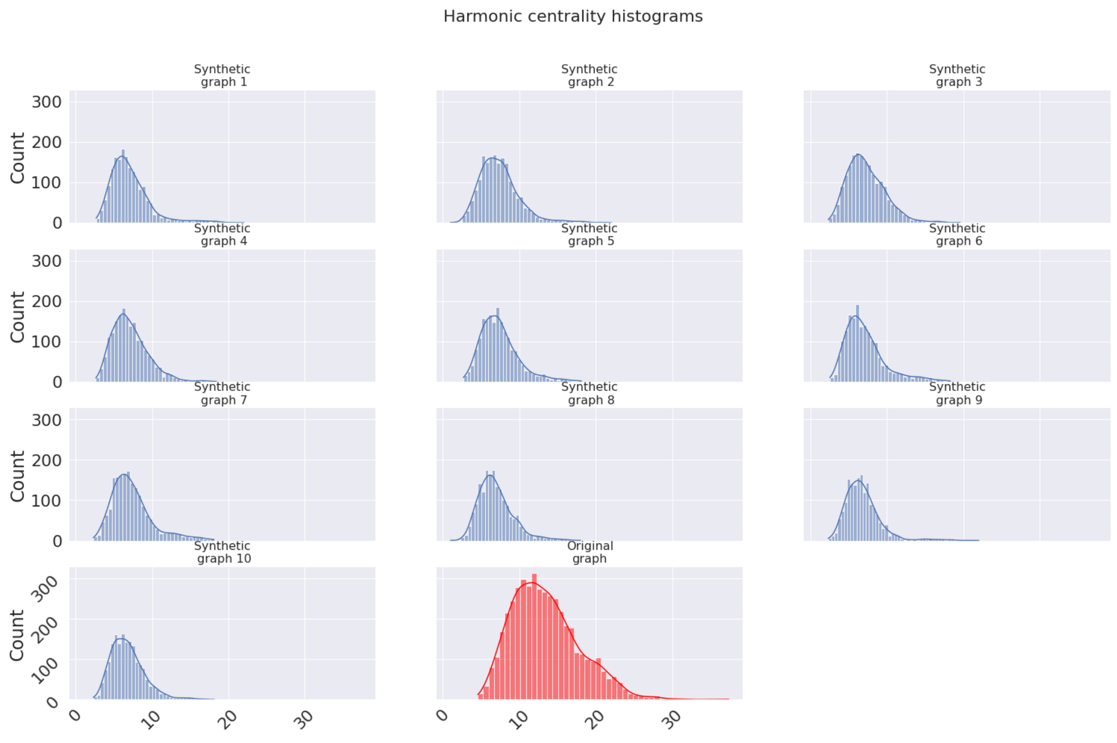


Figure A.4 low-trained graphgen citation dataset harmonic centrality histograms.

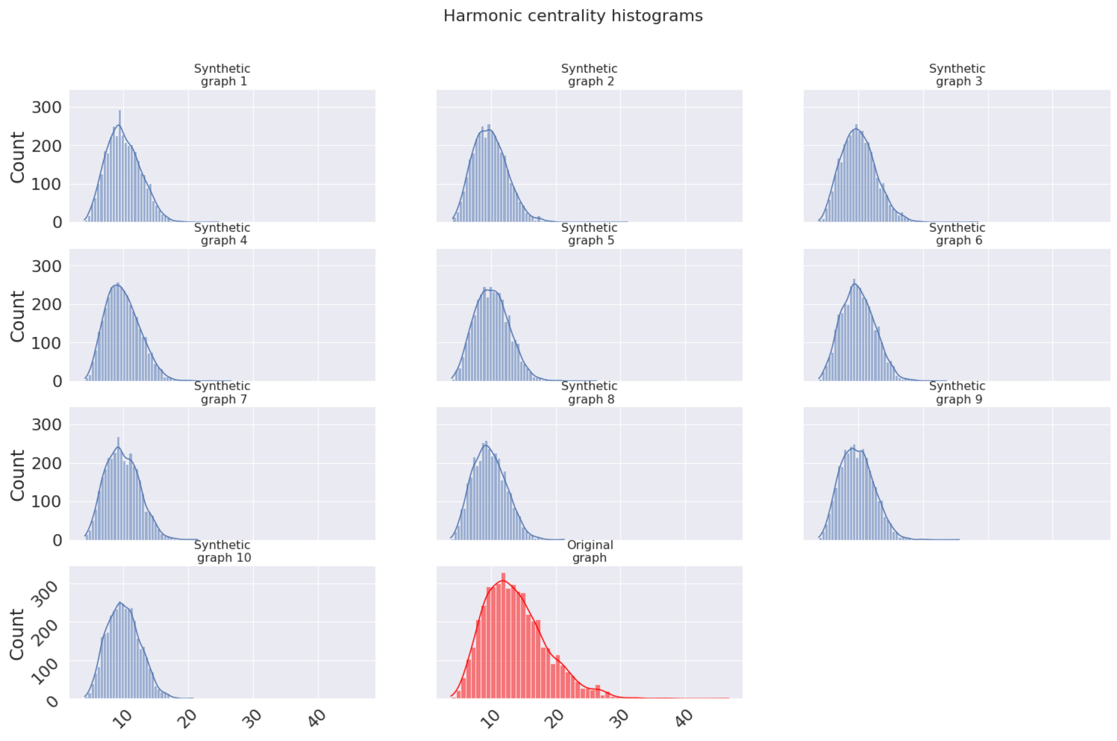


Figure A.5 high-trained graphgen citation dataset harmonic centrality histograms.

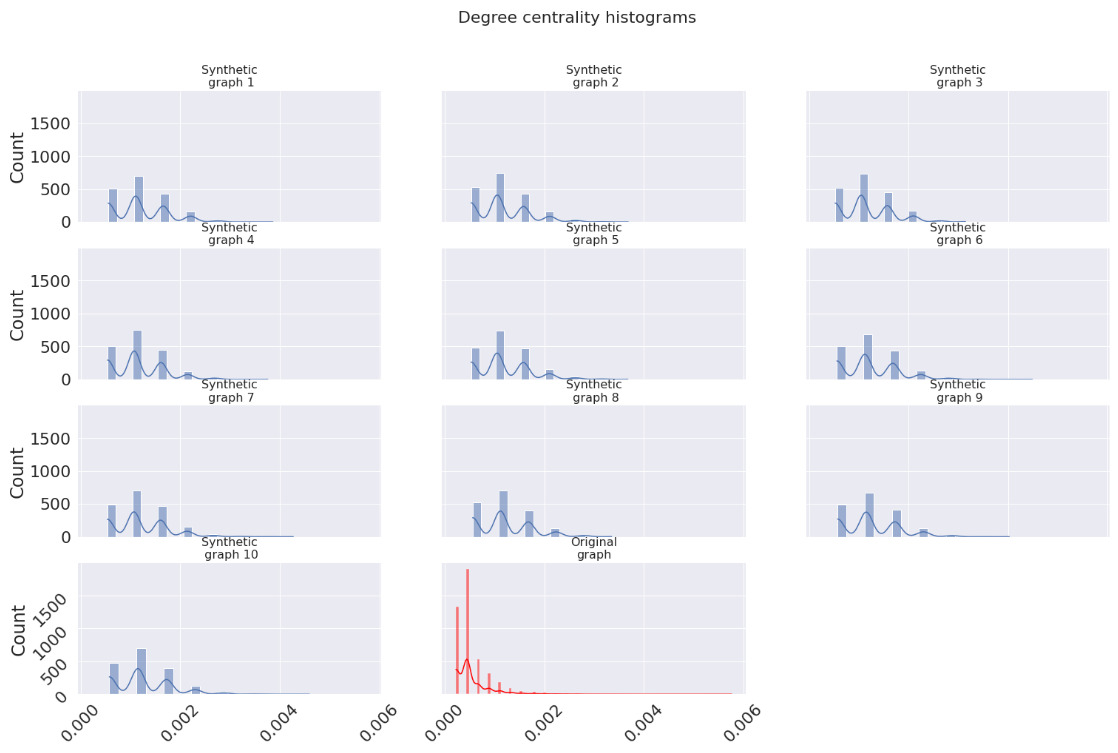


Figure A.6 low-trained graphgen citation dataset degree centrality histograms.

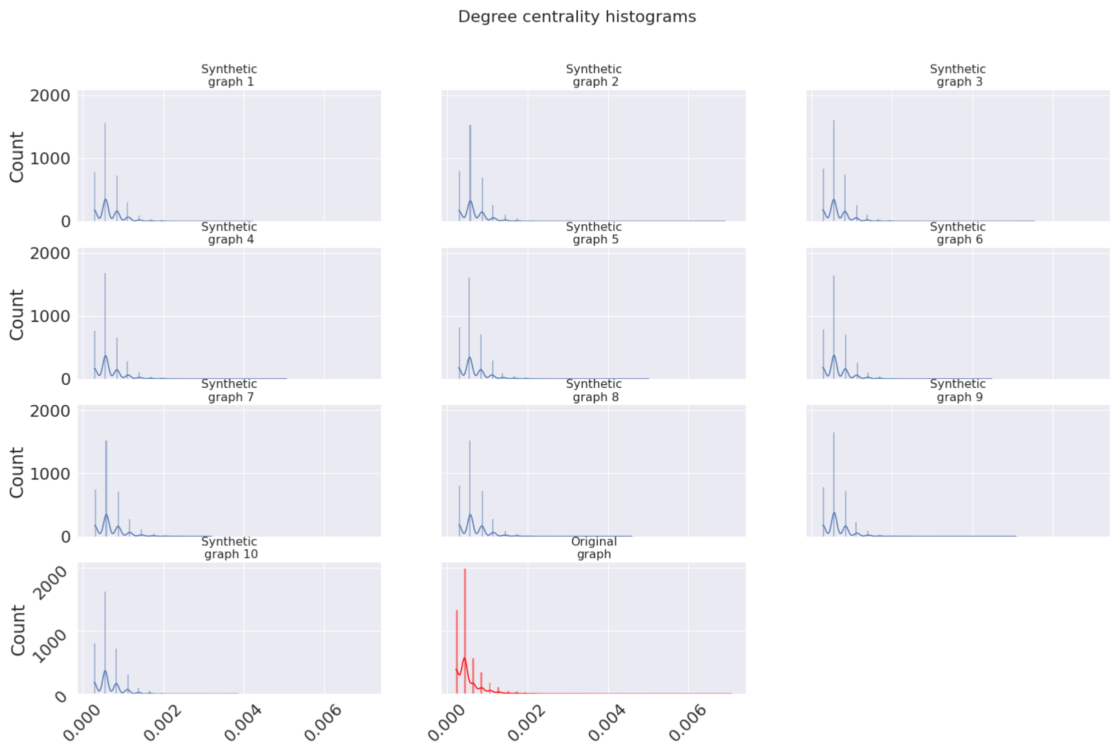


Figure A.7 high-trained graphgen citation dataset degree centrality histograms.

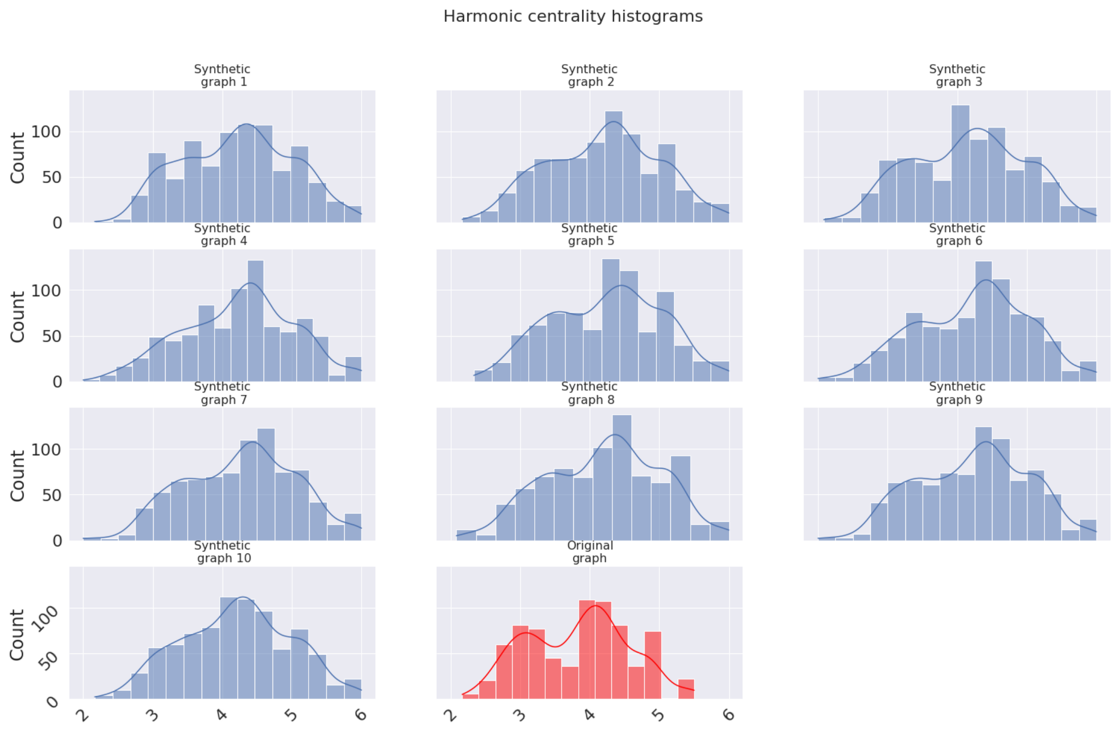


Figure A.8 GDSS harmonic centrality histograms.

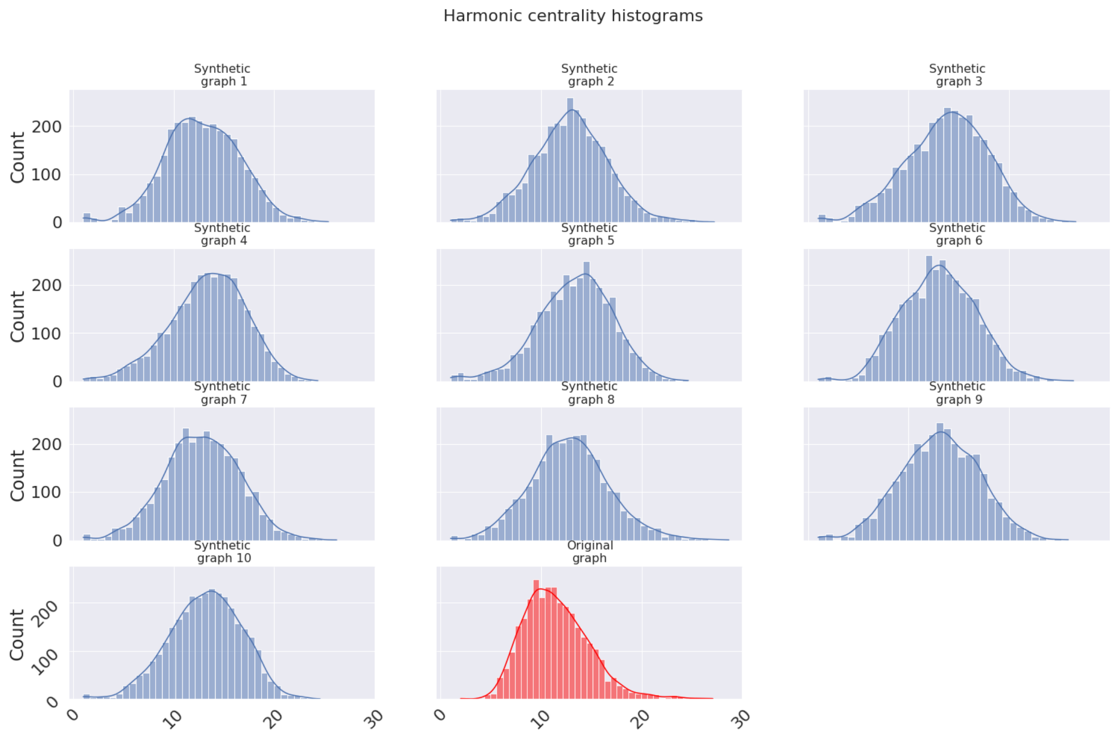


Figure A.9 Graphgen enzymes dataset harmonic centrality histograms.

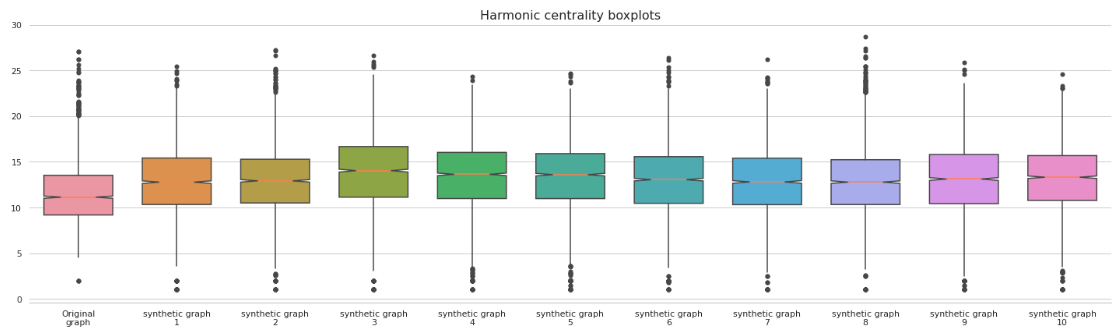


Figure A.10 Graphgen enzymes dataset harmonic centrality boxplots.

Bibliography

1. DIESTEL, Reinhard. *Graduate texts in Mathematics: Graph Theory*. Third edition. Berlin: Springer, 2006. ISBN 3-540-26183-4.
2. BLOCH, Francis; JACKSON, Matthew O.; TEBALDI, Pietro. *Centrality Measures in Networks*. arXiv, 2016. Available from DOI: `10.48550/ARXIV.1608.05845`.
3. BAVELAS, Alex. Communication Patterns in Task-Oriented Groups. *The Journal of the Acoustical Society of America*. 1950, vol. 22, no. 6, pp. 725–730. Available from DOI: `10.1121/1.1906679`.
4. SABIDUSSI, Gert. The centrality index of a graph. *Psychometrika*. 1966, vol. 31, pp. 581–603.
5. ROCHAT, Yannick. Closeness Centrality Extended to Unconnected Graphs: the Harmonic Centrality Index. In: 2009.
6. GARG, Manuj. Axiomatic Foundations of Centrality in Networks. *ERN: Networks (Topic)*. 2009.
7. FREEMAN, Linton C. A Set of Measures of Centrality Based on Betweenness. *Sociometry* [online]. 1977, vol. 40, no. 1, pp. 35–41 [visited on 2022-12-11]. ISSN 00380431. Available from: `http://www.jstor.org/stable/3033543`.
8. JUNG, Alexander. *Machine Learning: The Basics*. arXiv, 2018. Available from DOI: `10.48550/ARXIV.1805.05052`.
9. MA, Xuewei; QIN, Geng; QIU, Zhiyang; ZHENG, Mingxin; WANG, Zhe. *RiWalk: Fast Structural Node Embedding via Role Identification*. arXiv, 2019. Available from DOI: `10.48550/ARXIV.1910.06541`.
10. MIKOLOV, Tomas; CHEN, Kai; CORRADO, Greg; DEAN, Jeffrey. *Efficient Estimation of Word Representations in Vector Space*. arXiv, 2013. Available from DOI: `10.48550/ARXIV.1301.3781`.
11. MIKOLOV, Tomas; SUTSKEVER, Ilya; CHEN, Kai; CORRADO, Greg; DEAN, Jeffrey. *Distributed Representations of Words and Phrases and their Compositionality*. arXiv, 2013. Available from DOI: `10.48550/ARXIV.1310.4546`.
12. MEHLIG, Bernhard. *Machine Learning with Neural Networks*. Cambridge University Press, 2021. Available from DOI: `10.1017/9781108860604`.
13. MCCULLOCH, Warren S.; PITTS, Walter. A Logical Calculus of the Ideas Immanent in Nervous Activity. *The Bulletin of Mathematical Biophysics*. 1943, vol. 5, no. 4, pp. 115–133. Available from DOI: `10.1007/bf02478259`.
14. ROSENBLATT, Frank. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*. 1958, vol. 65 6, pp. 386–408.

15. KINGMA, Diederik P; WELLING, Max. *Auto-Encoding Variational Bayes*. arXiv, 2013. Available from DOI: `10.48550/ARXIV.1312.6114`.

16. DOERSCH, Carl. *Tutorial on Variational Autoencoders*. arXiv, 2016. Available from DOI: `10.48550/ARXIV.1606.05908`.

17. GOODFELLOW, Ian J.; POUGET-ABADIE, Jean; MIRZA, Mehdi; XU, Bing; WARDE-FARLEY, David; OZAIR, Sherjil; COURVILLE, Aaron; BENGIO, Yoshua. *Generative Adversarial Networks*. arXiv, 2014. Available from DOI: `10.48550/ARXIV.1406.2661`.

18. SIMONOVSKY, Martin; KOMODAKIS, Nikos. *GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders*. arXiv, 2018. Available from DOI: `10.48550/ARXIV.1802.03480`.

19. SIMONOVSKY, Martin; KOMODAKIS, Nikos. *Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs*. arXiv, 2017. Available from DOI: `10.48550/ARXIV.1704.02901`.

20. DE CAO, Nicola; KIPF, Thomas. MolGAN: An implicit generative model for small molecular graphs. 2018. Available from DOI: `10.48550/ARXIV.1805.11973`.

21. WEININGER, David. SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*. 1988, vol. 28, no. 1, pp. 31–36. Available from DOI: `10.1021/ci00057a005`.

22. ARJOVSKY, Martin; CHINTALA, Soumith; BOTTOU, Léon. Wasserstein Generative Adversarial Networks. In: PRECUP, Doina; TEH, Yee Whye (eds.). *Proceedings of the 34th International Conference on Machine Learning*. PMLR, 2017, vol. 70, pp. 214–223. Proceedings of Machine Learning Research. Available also from: `https://proceedings.mlr.press/v70/arjovsky17a.html`.

23. SCHLICHTKRULL, Michael; KIPF, Thomas N.; BLOEM, Peter; BERG, Rianne van den; TITOV, Ivan; WELLING, Max. *Modeling Relational Data with Graph Convolutional Networks*. arXiv, 2017. Available from DOI: `10.48550/ARXIV.1703.06103`.

24. YOU, Jiaxuan; YING, Rex; REN, Xiang; HAMILTON, William L.; LESKOVEC, Jure. *GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models*. arXiv, 2018. Available from DOI: `10.48550/ARXIV.1802.08773`.

25. CHUNG, Junyoung; GULCEHRE, Caglar; CHO, KyungHyun; BENGIO, Yoshua. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. arXiv, 2014. Available from DOI: `10.48550/ARXIV.1412.3555`.

26. POPOVA, Mariya; SHVETS, Mykhailo; OLIVA, Junier; ISAYEV, Olexandr. *MolecularRNN: Generating realistic molecular graphs with optimized properties*. arXiv, 2019. Available from DOI: `10.48550/ARXIV.1905.13372`.

27. GOYAL, Nikhil; JAIN, Harsh Vardhan; RANU, Sayan. GraphGen: A Scalable Approach to Domain-agnostic Labeled Graph Generation. In: *Proceedings of The Web Conference 2020*. ACM, 2020. Available from DOI: `10.1145/3366423.3380201`.

28. LEVY, Omer; LEE, Kenton; FITZGERALD, Nicholas; ZETTLEMOYER, Luke. *Long Short-Term Memory as a Dynamically Computed Element-wise Weighted Sum*. arXiv, 2018. Available from DOI: `10.48550/ARXIV.1805.03716`.

29. JO, Jaehyeong; LEE, Seul; HWANG, Sung Ju. *Score-based Generative Modeling of Graphs via the System of Stochastic Differential Equations*. arXiv, 2022. Available from DOI: `10.48550/ARXIV.2202.02514`.

30. NIU, Chenhao; SONG, Yang; SONG, Jiaming; ZHAO, Shengjia; GROVER, Aditya; ERMON, Stefano. *Permutation Invariant Graph Generation via Score-Based Generative Modeling*. arXiv, 2020. Available from DOI: `10.48550/ARXIV.2003.00638`.

31. SÄRKKÄ, Simo; SOLIN, Arno. *Applied Stochastic Differential Equations*. Cambridge University Press, 2019. Institute of Mathematical Statistics Textbooks. Available from DOI: `10.1017/9781108186735`.

32. SONG, Yang; SOHL-DICKSTEIN, Jascha; KINGMA, Diederik P.; KUMAR, Abhishek; ERMON, Stefano; POOLE, Ben. *Score-Based Generative Modeling through Stochastic Differential Equations*. arXiv, 2020. Available from DOI: `10.48550/ARXIV.2011.13456`.

33. DOCKHORN, Tim; VAHDAT, Arash; KREIS, Karsten. *Score-Based Generative Modeling with Critically-Damped Langevin Diffusion*. arXiv, 2021. Available from DOI: `10.48550/ARXIV.2112.07068`.

34. PARISI, G. Correlation Functions and Computer Simulations. *Nucl. Phys. B*. 1981, vol. 180, p. 378. Available from DOI: `10.1016/0550-3213(81)90056-0`.

35. LIPTON, Zachary Chase; ELKAN, Charles; NARAYANASWAMY, Balakrishnan. *Thresholding Classifiers to Maximize F1 Score*. arXiv, 2014. Available from DOI: `10.48550/ARXIV.1402.1892`.

36. LOUPPE, Gilles. *Understanding Random Forests: From Theory to Practice*. arXiv, 2014. Available from DOI: `10.48550/ARXIV.1407.7502`.

37. CHAWLA, N. V.; BOWYER, K. W.; HALL, L. O.; KEGELMEYER, W. P. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*. 2002, vol. 16, pp. 321–357. Available from DOI: `10.1613/jair.953`.

Obsah přiloženého média

```
│ readme.txt ........................ stručný popis obsahu média
├─ src
│  ├─ impl .......................... zdrojové kódy implementace
│  └─ thesis ........................ zdrojová forma práce ve formátu LaTeX
└─ text ............................. text práce
   └─ thesis.pdf .................... text práce ve formátu PDF
```