Universitat Rovira i Virgili (URV) y Universitat Oberta de Catalunya (UOC)

Master in Computational and Mathematical Engineering

FINAL MASTER PROJECT

Area: Ad hoc project ideas proposals

Forecasting Betweenness Centrality with Graph Neural Networks

---

Autor: Aitor Sánchez Quiroga

Tutor: Sergio Gómez and Clara Granell

Barcelona, June 20, 2023

Dr./Dra. Sergio Gómez and Clara Granell certifies that the student Aitor Sánchez Quiroga has elaborated the work under his/her direction and he/she authorizes the presentation of this memory for its evaluation.

Director's signature:

Digitally signed by SERGIO GOMEZ JIMENEZ - DNI 46557028Z
Date: 2023.06.20 21:56:57 +02'00'

Clara Granell Martorell

Digitally signed by Clara Granell Martorell
Date: 2023.06.22 10:01:04 +02'00'

Credits/Copyright

FINAL PROJECT SHEET

Title:	Forecasting Betweenness Centrality with Graph Neural Networks
Autor:	Aitor Sánchez Quiroga
Tutor:	Sergio Gómez and Clara Granell
Date (mm/yyyy):	June 20, 2023
Program:	Master in Computational and Mathematical Engineering
Area:	Ad hoc project ideas proposals
Language:	English
Keywords	Complex Networks, Graph Neural Networks, Betweenness centrality

"All models are wrong, but some are useful."

George E.P. Box

Acknowledgments

I would like to express my gratitude and to dedicate this work to all the people that have made possible its conclusion, contributed to it in any way and supported me or helped me in the difficult moments.

On the one hand, I would like to express my most sincerely appreciation to Professor Sergi Gómez, who proposed this project to me and started working with me even before the beginning of the academic year. His knowledge and experience have been fundamental pieces in the development of this project and it's been a pleasure to work under his guidance. Additionally, I would like to extend my appreciation to Professor Clara Granell, who even joining the project later, has become an essential key to the project, bringing a new vision, proposing important improvements and helping with the closure of this project. Thanks a lot to both of them for their advice and for helping me to finish this master's thesis.

On the other hand, I would like to mention and to acknowledge the support of my family, my friends and specially to my girlfriend for the support given to me through this journey. I couldn't enumerate the times I explained to them the different problems I had when performing the different experiments and how could I solve them. Thanks a lot for your patience and your unconditional support.

Finally, I would like to appreciate the opportunity of performing this master's degree final thesis since it has let me to learn about a new field, to improve my programming skills, to apply the different knowledge acquired during the master's degree and, last but not least, to improve in my organization and working effectively skills.

Abstract

Betweenness Centrality (BC) is a fundamental measure in network analysis that quantifies, for each node, its importance in terms of their relative positions and ability to efficiently connect to other nodes and facilitate the flow of information of the network. Its analysis can lead to relevant applications on various domains, such as identifying influential individuals in social networks, critical nodes in transportation networks, and essential proteins in biological networks. However, its computation is really expensive when dealing with graphs with a large number of nodes and connections, as is often seen in real-world graphs. In view of this, we analyse the possibility of applying Graph Neural Networks for the computation of BC with the hope of reducing the computational effort needed.

In order to find a solution for our purpose, we conducted extensive research in which we found a Graph Neural Network model introduced by [1] for the prediction of the Betweenness Centrality. Therefore, we analysed this approach in depth in order to understand its main features and perform some experiments.

In this work we perform and reproduce a selection of the experiments shown by [1], extending their analysis in terms of parameter variability. To further assess the model's performance in realistic scenarios, we introduce novel experiments considering new metrics and the inclusion of graphs that show community structure for the analysis of the model's accuracy. Our results reveal that the accuracy of the model is heavily influenced by the specific graphs used for training and testing, and that the inclusion of trivial solutions can lead to misleadingly high accuracy. The insights gained from our research contribute to a better understanding of the application of GNNs for BC computation and provide meaningful conclusions for future investigations in this field.

Keywords: Complex Networks, Graph Neural Networks, Betweenness Centrality

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Project context and justification

Betweenness Centrality (BC) is one of the most important graph centrality measures when analysing the spread of information through communities. It measures the level of importance of some graph nodes in terms of their relative positions and its analysis can lead to relevant conclusions and real applications. However, the calculation of the BC can be very expensive in terms of computational effort when graphs with a high number of nodes and connections are considered. This is precisely the case with real-world networks, where the computation of BC holds immense utility.

On the other hand, Machine Learning algorithms have proven their ability on problem-solving tasks such as predicting numerical values, classifying components into different categories, or automatically grouping data into different clusters, among others.

This work combines the last two previous subjects with the aim of analysing the possibility of applying a Machine Learning algorithm to predict the Betweenness Centrality of graphs, and therefore, avoiding the computational effort needed for the exact BC calculation.

1.2 Project's social-ethical, sustainability and diversity-related impact

The global ethical commitment competency (GECC) is included in this section. Taking into account that the calculation of Betweenness Centrality can have substantial impact on society when determining the importance of some nodes on real networks we relate this work with the Sustainability core aspect of GECC.

Sustainability: We consider that this work is related with the SDG 11 - Sustainable cities and communities taking into account the possible social applications, for instance, the identification of optimal locations for placing recharge car stations on cities. Then, a good identification of nodes with a high betweenness centrality could have a positive impact on the SDG 11. However, the current is a theoretical work, so there is not an specific analysis for this specific application. Instead, we analyse a mathematical model for predicting Betweenness Centrality focusing on its performance in a general and broader sense.

Ethical behaviour and social responsibility / Diversity (including gender) and human rights: Regarding the last two core aspects of the GECC this work doesn't have neither a positive or negative impact on their related SDGs (United Nation's 2030 Sustainable Development Goals). The analysis performed is so technical that it does not have a relation with human ethical behaviour neither diversity or human rights.

1.3 Project planning and objectives

Considering the project context and justification given, the main objective of this project is to analyse the possibility of predicting Betweenness Centrality using Machine Learning and, more specifically, Graph Neural Networks models. Then, in order to do it the first step of the project was to conduct extensive research on Graph Neural Networks and their applications, with a focus on graph centrality measures to analyse the possibility of predicting the Betweenness Centrality of graphs with these algorithms. Subsequently, after identifying a relevant paper [1] that introduces a Graph Neural Network model for computing Betweenness Centrality, the subsequent sections and stages of the project planning were determined as outlined below:

1. Perform research on Graph Neural Networks and its applications, with specific focus on graph centrality measures.

2. Analysis of the paper by Maurya et al. (2021) [1] and its introduced Graph Neural Network model.

3. Analysis and understanding of the code provided by [1] at `https://github.com/sunilkmaurya/GNN_Ranking` considering the requirements for its execution and the creation of a similar Python environment

4. Perform initial experiments to gain familiarity with the code.

5. Adaptation of the code for reproducing selected experiments shown by [1].

6. Extend the analysis performed by [1] in certain the experiments.
7. Analyse the potential of increasing the accuracy of the model using graphs with community structure.
8. Consider new metrics for evaluating the model's accuracy and behaviour.
9. Analyze the obtained results and draw conclusions based on the findings.

The tasks were executed following the prescribed order, considering the results obtained at each stage. The adherence to a well-defined workflow facilitated the achievement of the primary objective and the production of the results presented in this project.

1.4 Brief summary of outcomes

In this work a set of experiments are performed for testing the accuracy of the model introduced by [1]. We replicate some of the experiments shown by the paper in which we extend the analysis performed considering more variation parameters. In addition, we consider completely new scenarios for analysing the performance of the model in which we take into account new accuracy metrics.

All the experiments explained and performed in this work have been run using the code available at https://github.com/asanchezqui/tfm-gnn. It is worth noticing that the main part of the code belongs to [1] and has been extracted from https://github.com/sunilkmaurya/GNN_Ranking. However, with the aim of performing the different experiments we have adapted some parts of the code including all the necessary development for the replicated experiments and the new scenarios considered. As it can be seen, there is a `Readme.md` in which the content of the repository is described. It consists on different Python notebooks that contain the code used for running the different experiments. Moreover, there is an `env.yml` file with the dependencies needed for generating a Python environment able to run the code.

1.5 Brief description of the other section of the report

The next part of this work is divided into two main chapters. The second chapter is focused on introducing the theoretical concepts that are needed to understand the different sections. Firstly, a brief introduction to graphs is given, as well as the definition of the Betweenness Centrality measure. Then, an introduction to some machine learning concepts is given with a focus on Deep Learning, and more specifically, to Graph Neural Networks (GNN). Then, we describe the algorithm used in the paper of reference of this work: [1].

The third chapter is made of three clearly different sections. The first section of this chapter introduces the GNN framework described at [1] showing its main features and assumptions. Then, the second section is based on the replication of some of the experiments shown by [1] in which we extend the analysis performed considering more parameter variations. The third and last section of this chapter contains some new experiments in which we analyse the performance of the model when considering completely new scenarios such as training with different networks or considering new accuracy metrics. Moreover, some interesting patterns related to the model's accuracy are found when testing over real networks and the results are analysed.

Finally, Chapter 4 is devoted to the conclusions based on the diverse set of results obtained in the course of this work.

Chapter 2

Definition of the main concepts

2.1 Graphs

2.1.1 Graph representation and definition

Graphs have been present in the literature in a wide range of fields. They constitute the central object analysed by graph theory and can have a large number of applications due to its definition and properties. The main data structure this work is focused on are graphs and therefore, it is relevant to start the project explaining what a graph is and giving some examples.

A graph is formed by two objects, the nodes and the edges. The nodes (or vertices) are the set of objects that form the graph and the edges correspond to the relations (or connections) between the nodes. Taking this into account, a graph can be represented by $G(V, E)$ where V corresponds to the set of vertices or nodes and E corresponds to the set of edges connecting the different nodes. An edge can be represented by a pair of nodes $e_{ij} = (v_i, v_j)$ meaning that the node v_i is connected to the node v_j by the edge e_{ij}. Moreover, there can be graphs in which the direction of the edges is relevant (directed graphs) and then it is understood that the edge e_{ij} goes from the node v_i to the node v_j. Conversely, undirected graphs do not consider the direction of edges and the important thing is that the two nodes are connected.

Regarding the representation of graphs, one can think about the efficient way of storing these objects when a high number of nodes and edges are considered. There exist different ways of storing the graph structure on a computer in order to recover it easily and efficiently, but one of the most commonly used representations (and the one used in this work) is the adjacency matrix. The adjacency matrix notation is based on representing a graph $G(V, E)$ using a matrix. If the graph $G(V, E)$ contains n nodes, we define the entries of the adjacency matrix as $A_{ij} = 1$ if there is an edge from node v_i to node v_j. Following this notation, the dimension of the adjacency matrix will be $n \times n$. In addition, it is interesting to notice that

not all edges must have the same value of 1 at the adjacency matrix. Some graphs consider edges with weights, and consequently the adjacency matrix is given by $A_{ij} = w_{ij}$, where w_{ij} corresponds to the weight of the corresponding edge e_{ij}. Besides, it is interesting to notice that the adjacency matrix of an undirected graph is symmetric since each edge e_{ij} can be considered as the two edges $\{e_{ij}, e_{ji}\}$ and then A_{ij} will be the same as A_{ji}. The following figure (Fig. 2.1) extracted from [2] shows an example of an undirected graph and its symmetric adjacency matrix:

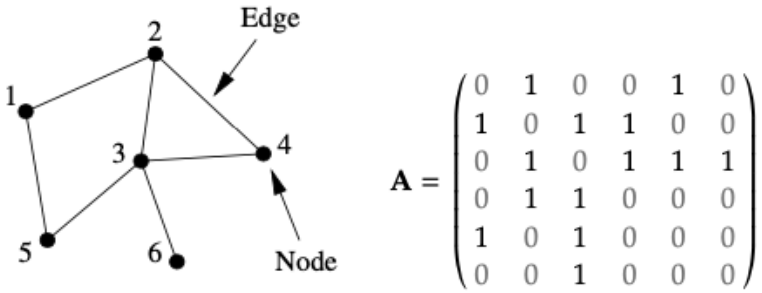

Figure 2.1: **Example of a graph and its adjacency matrix.**

2.1.2 Betweenness centrality and graph applications

Graphs have a large number of applications [2] and have been broadly used for representing real relational data. An example of a graph in the real world could be a social network in which the nodes are represented by people and the edges represent their relationships of friendship, debt or anything that relates two people. Another example of graph applications consists on representing the relations between scientific papers in which the nodes are the papers and the edges represent the references between them. Note that in this case the edges would be directed, since one paper referencing another does not imply a reference in the opposite direction. There are also applications related to biology and chemistry where graphs can be used to represent molecules and their interactions. Some interesting results arise from recent developments such as the contribution to the analysis of new drugs and the prediction of toxicity. As it can be seen, graphs can be applied to a wide range of real world fields.

Taking into account that graphs are constantly present in real life, it is interesting to consider some of the mathematical graph measures that might lead to relevant conclusions over real problems and therefore, its analysis could have a great impact on society. Betweenness centrality is a centrality measure that calculates the importance of nodes in a network based on their position as intermediaries in the flow of information or resources. It is important because nodes with high betweenness centrality can act as bridges, connecting otherwise disconnected

parts of the network and facilitating the flow of information or resources. Consequently, the analysis of betweenness centrality is important for different real situations. More specifically, betweenness centrality could be used for determining the priority of vaccine distribution to hospitals based on their potential to spread diseases, for example. Another useful application in the electric vehicle and transportation industry, would be to apply betweenness centrality to identify optimal locations for placing recharge stations.

Before defining betweenness centrality there is the need of introducing the shortest path concept. Given a graph $G(V, E)$, the shortest path between two nodes v_i and v_j is the set of edges that form the path between the two nodes minimizing the sum of the edge weights involved on the specific path.

The Betweenness Centrality (BC) measure is given by the following expression [1].

$$BC(v) = \sum_{u \neq v \neq w} \frac{\sigma_{uw}(v)}{\sigma_{uw}} \tag{2.1}$$

In the previous expression, $BC(v)$ corresponds to the Betweenness Centrality of node v, σ_{uw} corresponds to the total number of shortest paths going from node u to node w and $\sigma_{uw}(v)$ corresponds to the number of those shortest paths that go through v. For simplicity, if we consider every pair of nodes with at most one shortest path between them, the betweenness centrality of a node v corresponds to the fraction of all shortest paths (there will be one for each pair of nodes) between all the nodes except v that go through that node v.

Finally, it is worth to notice that the computational effort required for the calculation of the betweenness centrality once a graph is given grows rapidly with the size of the graph. Therefore, the time needed for computing the BC of a graph is directly related to the size of the graphs (number of nodes) since it requires analysing the shortest paths between all the pair combinations of nodes. Consequently, there is an interest on predicting the BC value of the nodes of a given graph using some algorithms that require a lower computational effort than the last expression. The solution this work is focused on is directly related to the prediction of BC using Graph Neural Networks.

2.2 Graph Neural Networks

The main purpose of the present work is to analyse the solution proposed at [1] based on Graph Neural Networks (GNN) for predicting the Betweenness Centrality of graphs. Therefore, it is worth introducing the concept of GNN but, in order to do it properly, the next sections briefly review the Machine Learning and Neural Networks (the model in which GNN are based on) concepts.

2.2.1 Introduction to Neural Networks

On the one hand, the concept of machine learning was introduced by Arthur Samuel [3] in 1959. It is a subfield of artificial intelligence and it is defined by IBM as a "branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy" [4]. In short, Machine learning algorithms use statistical and computational methods to identify patterns and relationships in data, and to make predictions or decisions based on that information. These algorithms typically involve training a model on a set of labeled data, adjusting its parameters to minimize the difference between its predictions and the true labels, and then applying it to new, unseen data. The output can be used to perform a variety of tasks, such as predicting numerical values or classifying input data into different categories.

On the other hand, Neural Networks are the main mathematical algorithms that Deep Learning (a subfield of Machine Learning) is focused on [5], and are the basis of Graph Neural Networks. Neural networks are a type of machine learning model that can make predictions for a variety of applications. These models are formed by a series of nodes (called neurons) and their weighted connections. At this point we can think of it as a kind of directed graph. The following figure (Fig. 2.2) extracted from [5] shows an example of a Neural Network:

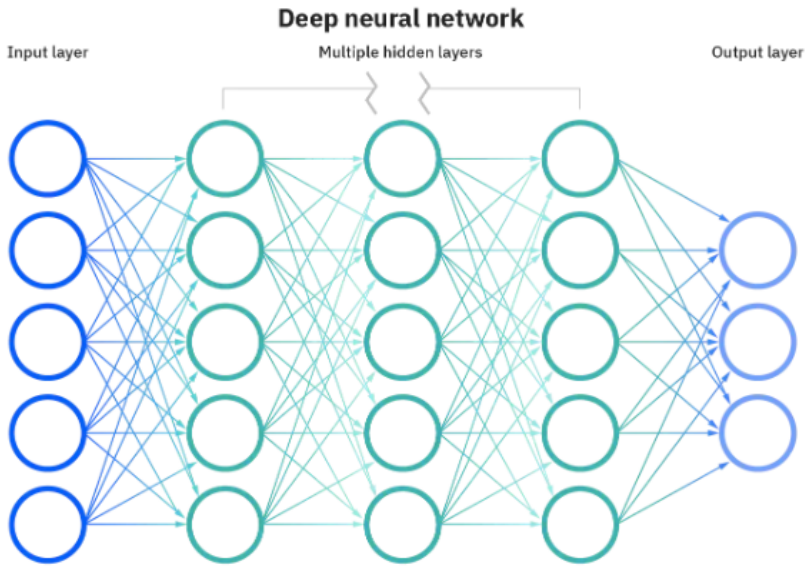

Figure 2.2: **Example of a Neural Network.**

As it can be seen in Figure 2.2, there is the input layer, the hidden layers and the output layer where each of the layers contain their own nodes. The input layer contains the input information with which the prediction (output layer) will be computed and the hidden layers contain a large number of parameters for adjusting the model with the objective of giving good

predictions and obtaining a good model performance. To understand how a Neural Network works, it is convenient to start by reviewing the concept of its fundamental unit, the Neuron. The purpose of a neuron is to perform a set of calculations on a set of input values, and then generate an output signal or value that is transmitted to other neurons in the network.

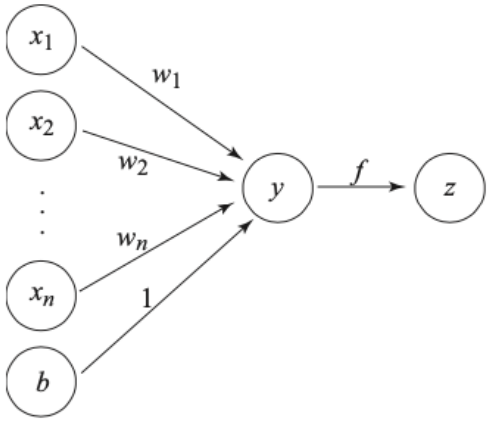

Figure 2.3: **Example of a neural network's node.**

In Figure 2.3, extracted from [6], we show the schema of what a neuron is. As we can see, the node y receives information from all the previous nodes considering different weights w_i and it outputs a value using the function f. Therefore, the output value from the node y will be $z = f(\sum_i w_i \cdot x_i + b)$. The function f is known as the activation function and it may vary depending on the Neural Network and the layer considered. Some of the typical activation functions are the sigmoid function and the Rectified Linear Unit function (ReLU).

Once we have reviewed the concept of a neuron, we can take a look at the previous figure of a Neural Network and better understand its parts:

- Input Layer: These nodes correspond to input values to the algorithm
- Weights: Each of the connections correspond to a weight parameter that will contribute to the output of each node.
- Hidden Layers: Hidden layers are sets of nodes that sit between the input layer and the output layer. Their outputs are used to calculate the output of the next layer or the final output of the model.
- Output Layer: These nodes correspond to the final output of the model

Finally, taking into account the different parts of the model, the output values are obtained from left to right of the Neural Network. The input values are passed to the first hidden layer considering all the weights for all the hidden nodes. Then, the process is repeated again for all the layers until the data arrives to the output layer where the result of the model is obtained.

2.2.2 Loss function and Back propagation

One of the most important parts of a machine learning model is the training process, where the model is adjusted in order to give good and reliable results. When a Neural Network is defined, the values of the weights are initialized randomly. After, they need to be properly adjusted, so the natural question is: How is a Neural Network trained? How does it learn from data? To answer these questions we need to introduce the concept of the *loss function*. The loss function, also known as the objective function or the cost function, quantifies the discrepancy between the predicted output of a neural network and the actual or desired output, with the goal of minimizing this discrepancy. It guides the adjustment of weights during the training process, aiming to achieve the lowest possible value for the loss function and thus improve the model's performance.

The backpropagation algorithm is the main algorithm for minimizing the loss function on Neural Networks so it is the key with which the Neural Networks models learn. The basic idea of this algorithm consists on using the derivatives of the cost function. If we have a training set available defined as some input data with its corresponding desired output, we can use the input data from the training set to obtain predictions with our Neural Network. Once we have the predictions made by the model, we can compare them with the expected or desired values. From this comparison we obtain the cost using the loss function and therefore, we have a measure of the error made by the model. The basic idea is to compute the derivative of the loss function and modify the weights with the aim of minimizing the cost. With the aim of knowing how much each weight should be modified, the partial derivatives of the loss function respect to the different weights are considered identifying the different weight contributions to the cost. As a consequence, the weights are adjusted taking into account their individual contribution to the model error.

For a deeper understanding of the algorithm and its mathematical derivation the following reference is a good resource [7].

2.2.3 The Graph Neural Network model

A Graph Neural Network is a sort of Neural Network totally focused on learning from graph data. As it is well known, graph data can be very complex because of its origin and representation. Graphs vary a lot in literature and they can show very different structures depending on the number of nodes and their connections. In order to introduce the concept of a GNN, it is common to start with the model proposed at [8].

The main objective of Graph Neural Networks is to define a Neural Network model which is able of representing graph data and learning from it. In graph theory, a graph is a mathematical

structure that consists of nodes (also known as vertices) and edges that connect them. Then, the the principal idea behind the model is based on the spreading of information through the graph nodes and edges. To obtain this spreading of information, an iterative operation is defined: in each iteration, each node aggregates the information of its neighbours. Therefore, as we perform more iterations each node will know, not only information about its neighbours, but also about the neighbours of its neighbours and so on.

More specifically, the model considers the nodes as objects or concepts and the edges represent their relationships. In addition, there are label vectors attached to each node and to each edge where the label vectors contain information representing some features of the corresponding object. The example given by [8] regarding the labels is based on considering an image. If we consider the representation of an image as a graph, each node of the image will have its label vector that can contain information related to the regions such as the area, perimeter and average color intensity. On the other hand, edge labels could contain information about the relative position regarding the regions. In addition, a state vector x_n (that will contain some aggregated feature information) is attached to each node in the graph and it will be used for obtaining an output o_n for each node. As it is already commented, the main idea is to define an aggregation operation with the aim of spreading the information through the graph.

The mentioned paper [8] introduces two important functions used by the GNN model, the local transition function f_w and the local output function g_w that are given by the following expressions:

$$x_n = f_w(l_n, l_{co[n]}, x_{ne[n]}, l_{ne[n]}) \tag{2.2}$$

$$o_n = g_w(x_n, l_n) \tag{2.3}$$

The last expression introduces some objects where l_n, $l_{co[n]}$ and $l_{ne[n]}$ correspond to the label of a node n, the labels of its edges and the labels of its neighbours respectively. As it is already commented, these labels contain feature information of the edges and the nodes. Finally, $x_{ne[n]}$ correspond to the vector states of the neighbours of the node n. It is interesting to show the next figure (Fig. 2.4) extracted from [8] in which the notation of the last expression is used:

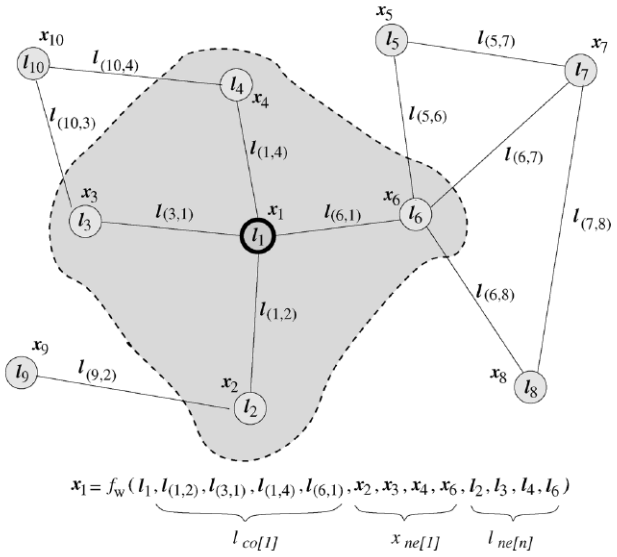

Figure 2.4: **Graph and the neighborhood of a node.** The state x_1 of the node 1 depends on the information contained in its neighborhood.

Following the last expression, it can be rewritten as:

$$x = F_w(x, l) \tag{2.4}$$

$$o = G_w(x, l_N) \tag{2.5}$$

where x, o, l and l_n correspond to the vectors containing all the states, outputs and labels. Then, F_w and G_w are known as the global transition and output functions respectively.

It is guaranteed by the Banach fixed-point theorem that the previous system of equations has an existing unique solution. The method for solving the last set of equations is given by the following expression:

$$x_n(t+1) = F_w(x(t), l) \tag{2.6}$$

We can iterate the previous expression in order to obtain the solution of the system, and it is assured that for any initial state the system will converge rapidly to the solution. Therefore, it can be seen that, using the previous expression, an operation for spreading the information through the network is obtained.

Considering the last expressions, in which the aggregation operation spreads information through the network, and the already mentioned method for training machine learning algorithms (using a loss function), we obtain a method for obtaining predictions and training the Graph Neural Network. As it will be seen, the GNN-based solution proposed by [1] analysed in the present work uses the multiplication of the adjacency matrix of a graph by its feature matrix to perform the aggregation operation and then spreading the information through the

graph.

2.2.4 Applications of GNN

It is well known that graphs have a large number of applications and, similarly, Graph Neural Networks can be applied to a wide range of fields [9]. Since there are so many applications, it is interesting to notice that the Graph Neural Networks tasks and applications can be at different levels such as Graph level, Node level or Edge level [10].

The main idea behind a graph-level graph neural network task is focused on the classification of an entire graph. A good example could be, in terms of drug discovery, a GNN model that is being used for predicting if a certain molecule is toxic or not.

On the other hand, node-level applications are those focused on nodes. Applications such as node classification and node regression belong to this group. The main objective of the present work is analysing the work performed in [1] for betweenness centrality prediction which is clearly a regression task focused at the node level.

Finally, in terms of edge-level tasks, the main purpose of graph neural networks is the edge prediction. A good example of this kind of problem could be the prediction of relationships between elements where the elements correspond to the nodes and the relations to the edges. For instance, in a social network, a graph neural network could be used for predicting some future node connections obtaining a social recommender.

For more information about the different Graph Neural Networks models and applications, a good resource could be [9].

Chapter 3

Graph Neural Networks for Fast Node Ranking Approximation

3.1 Article's proposed framework

With the aim of obtaining predictions of the betweenness centrality of graphs, a framework based on Graph Neural Networks is proposed by [1]. The proposed model is based on the aggregation operations of graph neural networks and, therefore, it uses the feature of message passing information through the layers of the model. Before introducing the model used by the related paper [1] it is worth showing two of the main model's features that are different from other GNN models:

- On the one hand, the model considers the typical GNN aggregation operation only through certain nodes. In order to do it, certain nodes (these nodes will correspond to N_z) with 0 betweenness centrality are identified beforehand. Then, these nodes are not considered for the graph neural network aggregation operation.

- On the other hand, the second main feature and difference with other models is that it does not include the node's own feature vector in the current layer. In other words, the node's own features are not directly used during the aggregation operation.

Taking into account the previously mentioned features, the following items delineate the primary assumptions and considerations of the proposed GNN model:

- The shortest paths going through a node are considered separately in incoming paths and outgoing paths. The computations regarding these two types of paths are performed in parallel and they do not interact until the end of the proposed model. This implies that the proposed GNN model considers directed graphs where the directionality of edges is taken into account.

- Identification of nodes with zero betweenness centrality:
  - Nodes that only have outgoing edges or incoming edges have not shortest paths going through them and therefore, resulting in a betweenness centrality of zero.
  - Nodes where all incoming neighbors are connected to all outgoing neighbors are also assigned a betweenness centrality of zero, since they do not act as intermediaries in the graph. Since all incoming neighbors are directly connected to all outgoing neighbors, there is no need for these nodes to mediate the flow of information or influence the shortest paths in the graph. For example, consider a node n_0 that has the connections e_{10}, e_{20} and e_{03}. If the connections e_{13} and e_{23} exist, node n_0 has zero betweenness centrality.
- With the aim of not considering the zero betweenness centrality nodes on the aggregation operation, the rows of the adjacency matrix regarding to these nodes are set to 0. Therefore, when the adjacency matrix is multiplied by the feature matrix, these nodes will not aggregate information of their neighbours.
- All nodes are assigned unique embeddings at the model initialization. Therefore, at the first layer, the information of the nodes N_z (the initialization embedding) is aggregated by their neighbours. However, it is worth mentioning again that although the initial embeddings of the N_z nodes are passed through their neighbours, these nodes N_z will not aggregate any information of other nodes since their row at the adjacency matrix is set to 0.
- The aggregation operation will correspond to the sum of the feature vectors. In order to perform the aggregation operation, the adjacency matrix is multiplied by the feature matrix with the aim of aggregating this information. Since each node will have a 1 at each position of the adjacency matrix where there is a neighbour, when multiplying by the feature matrix the output will be the sum of the neighbour's features for each node, except for the N_z nodes.
- A ReLu operation will be used as non-linearity in each layer
- The model will be formed by four layers where each layer will be formed by the aggregation operation, a ReLu operation as non-linearity and an MLP of this result. Where MLP corresponds to a Multilayer Perceptron, which is a totally connected Artificial Neural Network.

The following figure (Fig. 3.1) shows the pseudo-code of the algorithm described. As it can be seen, it uses the modified Adjacency matrix where the N_z nodes have their row set to zero.

In addition, ingoing paths and outgoing paths are considered separately: for ingoing paths, the algorithm employs the adjacency matrix, while for outgoing paths, it utilizes the transpose of the adjacency matrix. This first part of the process correspond to the preprocessing part of the model. Then, for each layer, the modified adjacency matrix is multiplied by the output of the previous layer $H^{(k-1)}_{out\_degree}$ and the weight matrix $W^{(k)}$. The output values are passed through a ReLu operation for breaking linearity obtaining $H^{(k)}_{out\_degree}$. Then, the obtained $H^{(k)}_{out\_degree}$ is passed through a MLP obtaining a vector $S^{(k)}_{out\_degree}$ that contains an output value for each node. Finally, the obtained values of each layer $S^{(k)}_{out\_degree}$ are summed to obtain the final output S_{out_degree}. The same process is performed for incoming paths obtaining S_{in_degree}. At the end of the process, S_{out_degree} is multiplied by S_{in_degree} obtaining the final values for each node that, once the model is trained, will be associated with the betweenness centrality of the nodes.

ALGORITHM 1: GNN-Bet (Forward propagation)

Input: Directed Graph adjacency matrix A; depth K; non-linearity ReLU; weight matrices $W^{(k)}$
Output: Betweenness Centrality value vector, $S_{(Bet)}$

$\tilde{A}_{\text{out-degree}} \leftarrow \text{ModifyAdjacencyRow}(A)$
$\tilde{A}_{\text{in-degree}} \leftarrow \text{ModifyAdjacencyRow}(A^T)$
for $k = 1...K$ **do**
  $H^{(k)}_{\text{out-degree}} \leftarrow \text{ReLU}\,(\tilde{A}_{\text{out-degree}} H^{(k-1)}_{\text{out-degree}} W^{(k)})$
  $H^{(k)}_{\text{in-degree}} \leftarrow \text{ReLU}\,(\tilde{A}_{\text{in-degree}} H^{(k-1)}_{\text{in-degree}} W^{(k)})$
  $S^{(k)}_{\text{out-degree}} \leftarrow \text{MLP}(H^{(k)}_{\text{out-degree}})$
  $S^{(k)}_{\text{in-degree}} \leftarrow \text{MLP}(H^{(k)}_{\text{in-degree}})$
end
$S_{\text{out-degree}} \leftarrow \sum_{k=1..K} |S^{(k)}_{\text{out-degree}}|$
$S_{\text{in-degree}} \leftarrow \sum_{k=1..K} |S^{(k)}_{\text{in-degree}}|$
$S_{(Bet)} \leftarrow S_{\text{out-degree}} \times S_{\text{in-degree}}$

Figure 3.1: **Pseudo-code of the model proposed by [1].**

In addition, the next figure (Fig. 3.2) shows the schema of the proposed model:

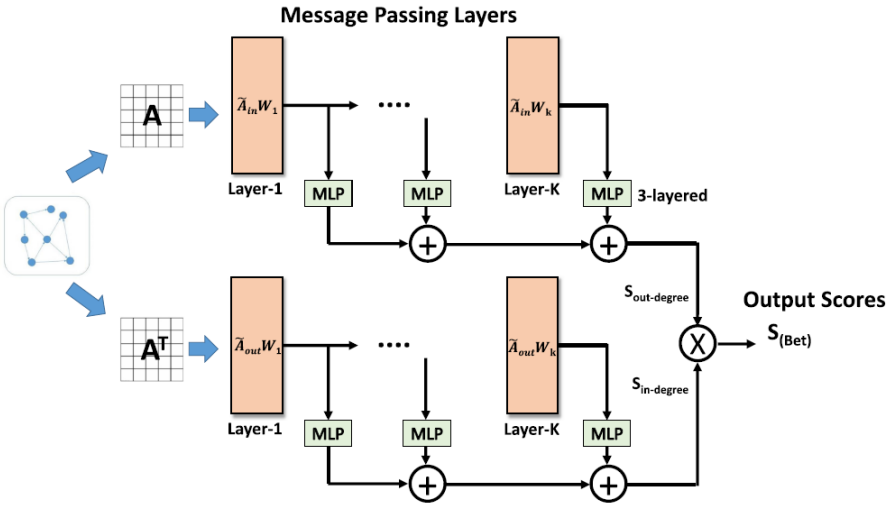

Figure 3.2: **Schema of the model proposed by [1].**

3.1.1 Loss function and Metrics

Considering the way in which machine learning models learn it is capital to define a good cost/loss function that informs the model when it is failing or succeeding.

The loss function proposed by the paper does not focus on making comparisons between the obtained values and the real ones of betweenness centrality. Instead, it places greater emphasis on the relative ranking of the results, specifically focusing on how nodes are ranked based on their centrality measure. Once the output vector with the predicted centrality values is generated by the model, the real betweenness centrality values are used for evaluating the wrongly ranked nodes over the generated results. Then, the `MARGINRANKINGLOSS PyTorch` function is used for evaluating the cost that is related to the following equation:

$$\text{loss}(x1, x2, y) = \frac{1}{l} \sum_i \max(0, -y_i * (x1_i - x2_i) + \text{Margin}) \tag{3.1}$$

where

$$y_i = \begin{cases} 1 & \text{if } x1_i \text{ should be ranked higher than } x2_i \\ -1 & \text{if } x2_i \text{ should be ranked higher than } x1_i \end{cases} \tag{3.2}$$

Here $x1$ and $x2$ correspond to two lists of a different selection of nodes containing the betweenness score obtained by the model. The size of the two lists is the same and correspond to l in the equation. In addition, the two lists $x1$ and $x2$ are related to different nodes in order to compare their betweenness score values having in mind their obtained ranking positions. Then, in order to know if a node from list $x1$ should have a higher or lower betweenness score than a node from list $x2$, the real list of betweenness centrality values is used for generating the y

vector of the equation. Finally, the "Margin" parameter of the equation is set to 1 by [1].

As it can be seen, when the predicted sequence of sorted nodes fits the real sequence perfectly, the cost will be 0 since the result is correct. However, when the model's obtained ranking is higher (or lower) than the actual ranking for a specific node, the loss function yields a positive value, indicating an increase in the loss.

3.1.2 Data preparation and performance

Once we have reviewed the proposed model and the loss function used for learning from data, we will focus on explaining how the data is prepared before using it as an input for the model, and how we will perform the accuracy assessment of the model.

Size of the model: First and foremost, it is needed to define an expected size of the model. This corresponds to the maximum size of the adjacency matrix that the model is able to accept as an input. Please note that the model can be trained using graphs of various sizes and can also be tested on graphs of different sizes. To be able to use adjacency matrices that are smaller than those used in the training phase, we can set the extra positions of the input adjacency matrices to 0. Please note that it is important to define a good model size before the training phase, since graphs with a higher size will not be accepted by the model once it is trained.

Splitting the data: The process of obtaining the training set and test set begins with a collection of graphs in which the betweenness centrality has been computed for all nodes. This initial information will allow us to train the model using the loss function and the real betweenness centrality of certain graphs. Furthermore, the model's performance is evaluated by using it to make predictions on additional graphs, and then comparing the predicted centrality values with the actual centrality values of those graphs. This comparison serves as a metric to assess how well the model performs in predicting the betweenness centrality of nodes in previously unseen graphs.

Once the initial data is ready, the amount of graphs that will be used for training or testing should be defined. In addition, a replication factor is defined. This replication factor is used to generate multiple training graphs from each original graph. By applying the replication factor, several versions of the original graph are created, each serving as a training example. Then, for each graph, a list of the nodes is created and randomly shuffled with the aim of creating an adjacency matrix using the permutation order obtained from the shuffling process. This process is performed n times (being n the replication factor) for each of the training graphs.

Modification of the adjacency matrix: Once the data splitting process is done and the test and train sets of graphs are ready to be used, the adjacency matrices are computed and modified as mentioned before. The process is the following:

1. The self-loop edges are removed.
2. The adjacency matrix of each graph is built using the sequence of nodes obtained in the shuffling process for each graph when splitting the data.
3. Each row of the resulting adjacency matrix is summed. Note that this quantity will correspond to the number of neighbours for each node. The same summation operation is performed for the transposed adjacency matrix. Then, the two previous values are multiplied for each node (i.e. element-wise). Nodes with a zero resulting value will not have shortest paths going through them and therefore will belong to the N_z set.
4. The nodes for which all the input neighbours are connected to all the output neighbours are identified. These nodes will also belong to N_z as mentioned before.
5. The rows of the N_z nodes are set to 0 for the adjacency matrix and its transpose.
6. Finally, the resulting adjacency matrix and its transpose are placed on the diagonal of a sparse matrix of model size dimension.

At this point the graphs are ready to be fed into the model to obtain predictions.

Performance Measure: As previously discussed, during the model training phase, we employ a loss function that assesses the progress by evaluating the resulting ranking of nodes based on their betweenness centrality values. Similarly, the evaluation phase will focus on the ranked list of nodes based on their betweenness centrality values.

The measure used for evaluating the model's performance is the *Kendall's Tau rank correlation coefficient*, a statistical measure used to assess the similarity or correlation between two ranked sets of data. The definition of this measure as given by the original paper [1] corresponds to the following equation:

$$\tau = \frac{N_c - N_d}{\frac{n(n-1)}{2}}. \tag{3.3}$$

Here N_c refers to the number of concordant pairs in the ranking, while N_d refers to the number of discordant pairs. Let us explain these two concepts. Consider two lists of equal length. We can select pairs of elements from these lists, denoted as (x_i, y_i) and (x_j, y_j) with $i < j$ where the x corresponds to values of the first list and y to the second. If $x_i < x_j$ and

$y_i < y_j$ or $x_i > x_j$ and $y_i > y_j$ (the pairs agree) they are said to be concordant, instead, they will be discordant. In the previous expression, n corresponds to the length of the lists that we are comparing. In this case, it corresponds to the quantity of nodes. As it can be seen, since the number of possible pairs is $n(n-1)/2$, the value of τ will fall in the interval $[-1, 1]$.

3.2 Replication of experiments

With the aim of analysing the commented solution for predicting the betweenness centrality of graphs we first replicate some of the experiments shown by the paper of reference. Furthermore, in certain cases, we conduct a more in-depth analysis of the obtained results by considering a wider range of variations in the parameters used during the experiments.

Since we are expecting a similar model behaviour and the idea is to perform some comparisons (where possible) between our results and the results shown by the paper, we need to take into account, if needed, the differences between the parameters and configurations used by the paper and the ones that we use for each of the experiments.

In order to perform the different experiments we use the code provided by [1] and available at https://github.com/sunilkmaurya/GNN_Ranking. We take their source code as a basis and we modify the code as necessary to perform all the experiments that are shown in the present work. Given that we utilized the code provided by the authors of the paper, available on GitHub, we conducted the experiments without altering the preexisting parameters in the code. This includes parameters related to graph generation as well as model parameters. However, it is important to note that we observed discrepancies between the parameters documented in the paper and those present in the source code. These variations on the parameters can lead to some differences on the results obtained. However, they do not significantly impact our conclusions and the overall findings. Furthermore, we obtained values that closely align with those presented in the paper, and when necessary, we discuss potential differences in experimental conditions. Here we detail the model parameters used in the current work and the differences with the ones used in the original paper:

- The number of layers of the model is set to 4 in both cases, the present work and the paper of reference.
- The model is trained using Adam as the optimizer with a learning rate of 0.0005 for the present work and a learning rate of 0.005 for the paper of reference.
- The calculation of the loss value is not performed using all the possible pairs of nodes. Instead (because of computational effort), the number of node pairs considered corresponds

to 20 times the number of nodes. The last configuration is considered for both the current work and also the paper of reference.

As it can be seen, the only difference between the model parameters utilized in the present work and the ones used by [1] corresponds to the value of the learning rate.

Finally, it is important to note that all the results that we present are obtained using a Macbook Pro with 2,3 GHz Intel Core i5 (4 nodes) and a RAM memory of 16 GB 2133 MHz LPDDR3. Therefore, since we don't have the same computational power as the one used by the paper, certain experiments in terms of efficiency cannot be replicated, while others will be adapted using smaller graphs. However, we take these variations into consideration when obtaining conclusions and analysing the results obtained.

3.2.1 Accuracy of the model

In this section we aim to replicate some of the experiments shown in the original paper, involving the training and utilization of the model on various types of graphs. The graphs considered belong to three different types of synthetic graphs, while some are real graphs.

Synthetic graphs

The synthetic graphs considered in the experiment correspond to Erdős-Rényi (ER), Scale-free (SF), and Gaussian random partition (GRP) graphs. We follow the same process of graph generation used by [1] for this experiment. In order to generate the input data, training, and test datasets, a total of 15 graphs were generated for each type of synthetic graph. Out of these 15 graphs, 10 graphs from each group of synthetic graphs were reserved for testing, while 5 graphs (following the shuffling process explained earlier) were used to obtain a total of 500 training graphs. Moreover, following the method used by the paper of reference, with the aim of considering variations on the input graphs, the generation parameters are chosen randomly for each graph considering some boundaries. Next, we give details of the Python `Networkx` functions used to obtain each type of graph.

- **Erdős-Rényi**
  - These graphs are generated using the `Networkx` Python library and the `random_graphs.fast_gnp_random_graph()` function.
  - The function returns an Erdős-Rényi graph according to the n and p parameters, where n corresponds to the number of nodes considered and p corresponds to the probability of creating an edge between two nodes.
- **Scale-Free**

  - These graphs are generated using the `Networkx` Python library and the `scale_free_graph()` function. The function returns a scale-free directed graph with the following parameters:
    * n: Number of nodes in the graph.
    * α: Probability of adding a new node connected to an existing node, chosen randomly according to the in-degree distribution.
    * β: Probability of adding an edge between two existing nodes. One existing node is chosen randomly according to the in-degree distribution and the other is chosen randomly according to the out-degree distribution.
    * γ: Probability of adding a new node connected to an existing node chosen randomly according to the out-degree distribution.

- **Gaussian random partition**

  - These graphs are generated using the `Networkx` Python library and the `gaussian_random_partition_graph()` function. A Gaussian random partition graph is created by creating k partitions each with a size drawn from a normal distribution with mean s and variance s/v. Nodes are connected within clusters with probability p_{in} and between clusters with probability p_{out}. The parameters are:
    * n: Number of nodes in the graph.
    * s: Mean cluster size.
    * v: Shape parameter. It determines the variance of the cluster size distribution, where a higher value of v leads to a larger spread of cluster sizes. The variance of cluster size distribution is s/v.
    * p_{in}: Probability of intra-cluster connection. It determines the likelihood of creating edges between nodes within the same cluster.
    * p_{out}: Probability of inter-cluster connection. It determines the likelihood of creating edges between nodes belonging to different clusters.

For more information regarding these types of graphs please refer to the `Networkx` package documentation.

Regarding the parameters used for generating the different type of graphs, it is worth mentioning that we find some differences between the graph generation parameters that are present in the source code and the ones used by the paper. The parameters used for the current work are listed below:

- Erdős-Rényi : These graphs are generated setting the probability p between $\frac{2}{10000}$ and $\frac{25}{10000}$.

- Scale-free: The parameters are set to: α between $\frac{40}{100}$ and $\frac{60}{100}$, $\gamma = \frac{5}{100}$ and $\beta = 1 - \alpha - \gamma$.
- Gaussian random partition: s and v take values between 200 and 1000, and p_{in} and p_{out} are set between $\frac{2}{10000}$ and $\frac{25}{10000}$.

The parameters used in the original paper are the following ones:

- Erdős-Rényi: p between $\frac{1}{1000000}$ and $\frac{99}{1000000}$
- Scale free: α between $\frac{40}{100}$ and $\frac{60}{100}$, $\gamma = 1 - \alpha - \gamma$ and β of $\frac{5}{10}$.
- Gaussian random partition: s and v between 2000 and 10000, p_{in} and p_{out} between $\frac{2}{100000}$ and $\frac{25}{100000}$

In addition, the original paper considers experiments involving graphs ranging from 50,000 to 100,000 nodes. However, due to resource limitations such as the unavailability of a GPU and the use of a laptop for the simulations, we will generate graphs of 5,000 to 10,000 nodes, to accommodate to the lack of computational power. Lastly, the paper does not specify the exact number of epochs for which the model is trained but mentions a range of 5 to 10 epochs. Therefore, in our experiments, we also train the model for up to 10 epochs and evaluate the obtained values.

Once the graphs previously mentioned have been generated, the model is trained and tested independently for each group of synthetic graphs. The results of the accuracy of the model using the Kendall's Tau (KT) measure obtained for each experiment are shown below (Table. 3.1):

Dataset	Paper KT value	Obtained KT value
Synthetic-ER	0.902 ± 0.03	0.889 ± 0.019
Synthetic-SF	0.976 ± 0.01	0.974 ± 0.001
Synthetic-GRP	0.899 ± 0.04	0.879 ± 0.056

Table 3.1: **KT values for synthetic graphs.**

As it can be seen in Table 3.1, the results obtained are close to the results shown in the paper. However, there are some differences since we do not obtain the same level of accuracy. It is worth remembering that we have used synthetic graphs of 5,000 to 10,000 nodes for training and testing the model instead of 50,000 to 100,000 nodes and therefore, it is important to mention that we were able to achieve comparable results even when working with smaller graphs. Regarding our experiment, it is still consistent since we train and test the model over the same type (5,000 to 10,000 nodes) of graphs. However, we obtain slightly lower values of performance for the graphs used in the experiment.

It is interesting to note that certain configurations can alter the results of the experiment. For instance, as we have just seen, the differences on the parameters used for generating the graphs. In addition, the random seed used when replicating the training graphs also plays a role since different random seeds will lead to different input matrices for the training process. These parameters change the test and training sets of graphs that are used by the model, resulting in experiments conducted under varying conditions. In addition, even if we were to apply the same parameter boundaries for graph generation, selecting a different random seed during the generation process would still produce different graphs, making it challenging to precisely replicate the conditions outlined in the paper.

Finally, taking into account the possible variations on the train and test data due to random seeds and to certain parameters what we can conclude looking at the results is that, even though the experiments have been run with certain differences on the test and train graphs compared to the paper's experiment, the model shows a similar value of accuracy for the different types of graphs. Therefore, we still obtain a good performance of the model and it is not worth comparing the exact value obtained with the paper's value since we cannot reproduce the same experiment conditions exactly.

In addition to the previous experiment, we try to extend the analysis performed by [1] by analysing the performance obtained when modifying the replication parameter for the training graphs. The same experiment is repeated for a replication rate of 1, 2, 10, 20 and 40, so the training sets considered will contain a different number of training graphs for each experiment. The results obtained are shown below (Table. 3.2):

	Average KT values obtained using diferent training sets				
Dataset	5 graphs	10 graphs	50 graphs	100 graphs	200 graphs
Synthetic-ER	0.792 ± 0.043	0.807 ± 0.021	0.869 ± 0.025	0.880 ± 0.021	0.880 ± 0.022
Synthetic-SF	0.965 ± 0.002	0.970 ± 0.001	0.974 ± 0.001	0.974 ± 0.002	0.974 ± 0.001
Synthetic-GRP	0.783 ± 0.073	0.810 ± 0.075	0.857 ± 0.080	0.866 ± 0.071	0.871 ± 0.064

Table 3.2: **KT values for synthetic graphs when changing the replication factor of training graphs.**

The average KT results obtained in Table 3.2 show variations on the performance of the model when the replication factor is modified. We obtain that when the replication factor increases, and therefore the number of training graphs is increased, the performance obtained is higher, as expected. This result makes sense because when a higher replication factor is considered, a larger training set is obtained, providing the model with more data for learning. However, it is worth considering the standard deviation values of each result, as they indicate some variability when testing over different graphs. Finally, it is worth remembering that we are considering 10

epochs for training the models.

Because we want to explore the possible variations on the graphs considered due to the random selection of parameters for the generation of graphs and the replication process, we extend the analysis performed in the paper by considering variations on these parameters. For the next experiment we generate 10 SF, ER and GRP graphs respectively for testing the models. We then generate 5 SF, ER and GRP graphs for the training process, considering 5 different random seeds at the replication process (with a replication factor of 10). Finally, we train different models using the distinct training sets obtained and we test always with the same test set (for each type of graph) with the aim of testing over the same conditions across all cases. The next table (Table. 3.3) shows the results obtained when considering different random seeds at the replication process for the same train set of graphs.

ER KT values	SF KT values	GRP KT values
0.8589 ± 0.0186	0.9745 ± 0.0028	0.8988 ± 0.0078
0.8491 ± 0.0165	0.9744 ± 0.0027	0.8936 ± 0.0065
0.8537 ± 0.0164	0.9745 ± 0.0026	0.8962 ± 0.0086
0.8532 ± 0.0176	0.9744 ± 0.0026	0.8958 ± 0.0089
0.8496 ± 0.0175	0.9744 ± 0.0026	0.8966 ± 0.0075

Table 3.3: **Results obtained when considering different random seeds during the replication process.**

In the next experiment, we analyse the effect of generating again the training graphs. We generate again (using different random seeds) 5 SF, ER and GRP training graphs, and we test over the same graphs used in the last table. The generation process of graphs contains a random choice of parameters, so we expect to obtain different training sets for the same type of graphs. Then, the random seed used for the replication process of the training graphs is fixed for each of the train sets obtained, in order to specifically analyse the effect of changing the random seed in the graph generation process. The next table (Table. 3.4) shows the results obtained.

In the previous two experiments, we expected that the variations on the random seeds considered (at the replication and at the graph generation stages) would lead to variations on the accuracy results obtained, however, one of the changes is more relevant than the other.

As we can see in Table 3.3 when looking one column at a time, there are minimal variations on the results obtained when changing the random seed of the replication parameter. Conversely, when we consider different random seeds at the graph generation stage for the training graphs (see Table 3.4), we obtain higher variations on the accuracy results than when considering variations on the random seed at the replication stage. Besides, it is interesting to

ER KT values	SF KT values	GRP KT values
0.8589 ± 0.0186	0.9744 ± 0.0025	0.8981 ± 0.0087
0.8092 ± 0.0909	0.9745 ± 0.0027	0.8988 ± 0.0136
0.8143 ± 0.038	0.9746 ± 0.0027	0.8791 ± 0.0048
0.8396 ± 0.0638	0.9742 ± 0.0028	0.9032 ± 0.0131
0.858 ± 0.0228	0.9746 ± 0.0028	0.8972 ± 0.0149

Table 3.4: **Results obtained when considering different random seeds at the graph generation process.**

notice that this behaviour is consistent across the different synthetic graphs considered, except for SF graphs.

Considering the results obtained and also shown in the paper of reference, it is interesting to notice that, for all experiments, the model performance is higher for SF graphs than for GRP and ER graphs. The difference of performance for different types of graphs can be ascribed to the different nature and features of each type of graph. Therefore, it could be interesting to consider more types of synthetic graphs for a deeper analysis of the model performance.

Finally, it is worth remembering that the loss function also contains a certain level of randomness since the number of pairs considered when computing the loss is 20 times the number of nodes. As a consequence, we expect also some variations on the results when performing the same experiment again with a different random seed. However, despite this randomness, we obtain results that are consistent not only with the paper's results but also with repeated experiments using different random seeds while maintaining the same train and test sets. This demonstrates that the observed outcomes are not merely the result of chance, affirming the effectiveness of the training process.

Real graphs

The Graph Neural Network model introduced in the original paper is also tested over different real networks. With the goal of testing the model over real graphs, a synthetic set of SF graphs is used during the training process. The choice of using Scale-free graphs to train the model follows the assumption that real graphs have Scale-free properties and therefore their structure is similar. To perform the experiment, the paper considers 5 SF graphs of 100,000 nodes. Then, a replication factor is used for obtaining 200 training graphs. However, similarly to the last experiment, we will perform the same experiment considering smaller graphs but, at the same time, we perform a deeper analysis of the performance of the model when varying the parameters. The real graphs considered are obtained from the SNAP dataset and can be found at: https://snap.stanford.edu/data/.

We choose to work with four of the real graphs considered in the paper, considering their diverse structure and configurations. The results shown by the paper are presented below (see Table 3.5).

Graph	Nodes	Edges	Paper KT values
wiki vote	7115	103689	0.937
soc Epinions	75879	508837	0.917
email EuAll	265214	420045	0.927
web Google	875713	5105039	0.699

Table 3.5: **Real graphs considered and its features: Number of nodes, edges, and KT values obtained in the original paper.**

As previously mentioned, we conduct an extensive analysis of the model's performance by exploring a wide range of parameter values. We perform the experiment considering variations on the number of nodes considered when generating the SF training graphs, and we also modify the replication parameter. The number of nodes considered are 10, 100, 1,000, and 10,000 nodes, while the replication factors considered are 1, 10, 20 and 40, yielding training sets of 5, 50, 100 and 200 graphs, respectively. The Kendall's Tau values obtained are shown in Tables 3.6, 3.7, 3.8 and 3.9 below.

Copies	10 nodes	100 nodes	1000 nodes	10000 nodes
1	0.9262	0.9243	0.9229	0.924
10	0.8845	0.9187	0.9261	0.9253
20	0.9091	0.8986	0.9241	0.922
40	0.8804	0.8964	0.9284	0.9215

Table 3.6: **Wiki-Vote network.** KT values obtained testing over the wiki-Vote network. A model size of 10,000 was used for this experiment.

Copies	10 nodes	100 nodes	1000 nodes	10000 nodes
1	0.9032	0.9036	0.9036	0.9036
10	0.9075	0.9024	0.896	0.8895
20	0.9022	0.8994	0.8947	0.8915
40	0.9026	0.8877	0.8949	0.8943

Table 3.7: **soc-Epinions network**. KT values obtained testing over the soc-Epinions network. A model size of 100,000 was used for this experiment.

Copies	10 nodes	100 nodes	1000 nodes	10000 nodes
1	0.9949	0.9949	0.995	0.9949
10	0.9945	0.9957	0.996	0.9961
20	0.9922	0.996	0.9961	0.9309
40	0.9939	0.9951	0.9957	0.9664

Table 3.8: **email-EuAll network.** KT values obtained testing over the email-EuAll network. A model size of 300,000 was used for this experiment.

Copies	10 nodes	100 nodes	1000 nodes	10000 nodes
1	0.8089	0.8093	0.8089	0.8094
10	0.7921	0.8119	0.8121	0.813
20	0.7921	0.8109	0.8153	0.8169
40	0.7993	0.7981	0.8116	0.8189

Table 3.9: **web-Google network.** KT values obtained testing over the web-Google network. A model size of 900,000 was used for this experiment.

The results obtained reveal minimal variations on the results when the parameters are modified. One would expect that considering more nodes for the training graphs would lead to higher values of the Kendall's Tau coefficient, since considering more nodes gives more information to the model to learn. Similarly, we would expect an increase of the performance of the model when increasing the replication factor. Indeed, we have already obtained, on previous experiments and considering other type of graphs, an increase of the model's performance when a higher replication factor is considered but the same pattern is not obtained on the last experiments over real graphs. However, we have shown in previous experiments that changing the parameters when considering scale-free graphs led to small variations on the results. Therefore, the results that we obtain from the current experiments where we train with SF and test over real networks are still consistent if we consider the assumption of similarity between real graphs and SF graphs. We therefore conclude that for this experiment we do not obtain high and consistent variations when the number of nodes considered for the training graphs or the replication factor is changed.

On the other hand, we obtain a good performance of the model. We obtain similar results to the ones shown in the original paper for the *Wiki-Vote* and *soc-Epinions* graphs. In addition, we have obtained a higher performance for the bigger graphs, which are *email-EuAll* and *web-Google*. As it can be seen we obtain a value around 0.99 for *email-EuAll* while the paper shows a KT (Kendall's Tau) value of 0.92. However, it is interesting that for a particular configuration (20 copies and 10,000 nodes) we obtain a KT value of 0.93 that is a closer value to the one shown by the paper of reference. Regarding the *web-Google* graph we obtain values up to 0.81

while the paper's value is of 0.69. Thus, we achieve notably higher KT values for the latter two graphs. The reason for this might be related to the networks used for the training process since, as it has been seen, the parameters used for graph generation are are not identical and also exhibit some degree of randomness.
Finally, it is worth mentioning that the model has been able to perform predictions over real networks obtaining a good performance when it is trained using SF graphs. In addition, it is interesting to notice that we still obtain high KT values when we consider small training graphs (10 to 100 nodes) and a low number of replications.

3.2.2 Scalability

This section focuses on the analysis of the computational time required to identify the zero shortest path nodes (N_z). As mentioned previously, the process of identifying these nodes takes place on the preprocessing part of the data.

To perform this experiment, synthetic Erdős-Rényi graphs are selected in the original paper, due to the ease of modifying the parameters generation of these graphs. Then, an ER graph is created for each combination of parameters. The experiment considers graphs with a number of nodes of (1 to 10) $\times 10^5$. The ratio between the number of nodes and the number of edges in these graphs is set to 2, 4, and 6. The following figure (Fig. 3.3) show our results (at the left) and the results obtained in the paper (at the right).

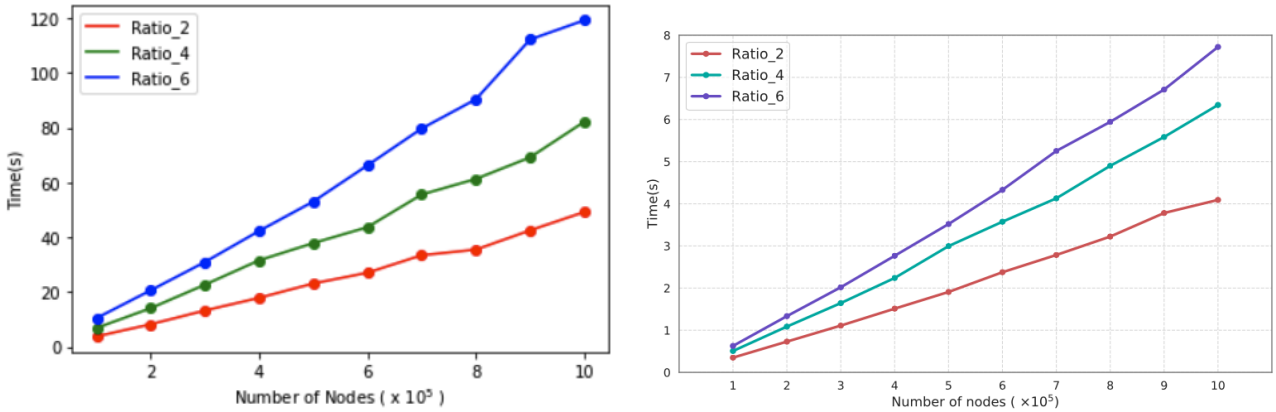

Figure 3.3: **Scalability results.** Time needed to process ER graphs as a function of the size of the graph, for three different ratios of the number of nodes and number of edges. The left plot is our result, while the right one is the result of the original paper. We can observe a good qualitative agreement between the two plots, with discrepancies in the time scale attributed to differences in computational power between the original paper and our work.

As it can be seen in Figure 3.3, the results obtained are in agreement with the results that are shown in the paper. On the one hand, the time needed for processing ER graphs increases with the number of nodes, as expected. Moreover, we obtain the same linear behaviour when increasing the number of nodes considered. On the other hand, the time needed when considering a higher ratio of number of nodes and edges is not only higher for higher ratios but also the slope of the linear behaviour is increased. Finally, it is worth noticing that the absolute time values obtained are far from the values shown by the paper since we do not have the same computational power for performing the experiments.

3.2.3 Ablation tests

Varying the number of layers

This experiment aims to analyse the accuracy of the model when the number of layers is changed. We train and test the model for each type of synthetic graphs considered and the number of layers is changed from 1 to 7. The synthetic graphs used here are the same train and test sets used when analysing the performance on synthetic graphs in Section 3.2.1. The results obtained, as well as the results provided by the paper, are shown in Fig. 3.4.

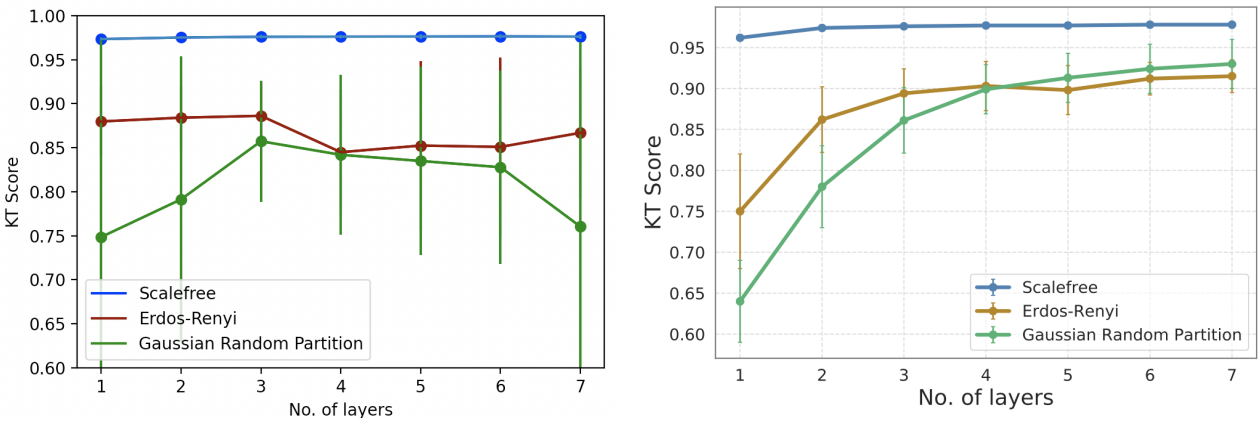

Figure 3.4: **Kendall's Tau coefficient obtained when training and testing over ER, SF and GRP synthetic graphs.** Variations on the number of model layers are considered. The left plot is our result, while the right one is the result of the original paper [1]. We can observe some discrepancies between the two plots where we obtain more variability on the results.

Looking at the results provided in the paper (see Fig. 3.4 (right)), the accuracy of the model is clearly increased in the three types of networks when the number of layers is increased from 1 to 4. When the number of layers considered is increased from 4 to 7, the performance of the model still increases slightly, but the lines obtained are close to flat lines. Therefore, one of the conclusions of this experiment is that the model is robust against changes when we consider a number of layers >= 4. In our case (see Fig. 3.4 (left)), it can be seen that we do not obtain

exactly the same increase in accuracy and behaviour when the number of layers is changed. Instead, our average KT values remain relatively constant with some variations depending on the graphs under consideration. Regarding scale-free graphs, our experiments show a very good agreement with the results of the original paper. In this type of networks, there is no variation on the results when the number of layers is changed, and we obtain similar KT score values. However, this is not the case in the specific case of GRP networks, which show high variations not only on the average KT score obtained but also on the standard deviation of the results. Let us remind the reader that the error bars are obtained from testing over 10 different graphs for each experiment. Our results show an increase on the KT score for GRP graphs when the number of layers is increased from 1 to 4. After that, when we increase the layers from 4 to 7, the KT score seems to decrease. However, taking into account the standard deviation (error bars) of our results, it is difficult to obtain a clear conclusion about the behaviour of the model's performance when the number of model layers is changed for GRP networks. Finally, regarding the Erdős-Rényi graphs, we do not obtain the clear increase on KT that the paper shows when increasing the number of layers from 1 to 4. Instead, we obtain more stable results on KT score as the number of layers increases.

In line with the previous analyses, it is worth noticing and remembering that our results have not been obtained with the same graphs used by the paper since we have considered different boundaries for the graph generation parameters. This can lead to differences on the KT results and standard deviation values. Even if we had considered the same graph generation parameters, we probably would have obtained different graphs due to the randomness of the process of generating the graphs and the test and train sets. In addition, different boundary values can lead to a wider range of graph structures and consequently a higher standard deviation values when testing the model over different graphs. Therefore, one of our main conclusions is that the model's performance can vary for different graphs, and the effects of parameter changes may be different depending on the specific test and train graphs considered.

Finally, it is important to note that the paper does not provide an explanation of how the error bars are obtained. We use standard deviation values to plot the error bars but this metric could differ from the one used by the paper. Additionally, if we were to calculate the SEM (Standard Error of the Mean), we would need to divide our standard deviation values by the square root of the sample size. Since we test over 10 different graphs for each experiment, we should divide the current error values by $\sqrt{10}$, which is nearly 3. By comparing the error bars in our results with those reported in the paper, we find that the paper's error bars are approximately one-third the length of ours, suggesting a possible match between our results and the paper's under the assumption that they would be using the SEM measure. However,

even if we obtained similar error values, the different trends commented previously are still present.

Varying the embedding dimensions

In addition to the last experiment, this experiment analyses the accuracy of the model when changing the embedding dimension used for the three types of synthetic graphs considering models of 1, 4 and 7 layers. Regarding the train and test graphs, the graphs used are the same that have been used by the last experiment. The results obtained are shown in Figure 3.5.

Before analyzing the results obtained, it is worth mentioning that we considered embedding dimensions from 4 to 20 instead of 2 to 20 since the code provided by the paper thrown "NaN" values for the specific configuration of 2 embedding dimensions.

As we can see in Figure 3.5, the results obtained for the Scale-free networks are similar to the ones given by the paper. The KT score values are lower for the 1-layered model than the values obtained when considering more layers. However, we obtain smaller variations on the results when considering different layers than the ones given by the paper. Regarding the Erdős-Rényi networks, we can see that the results obtained in the paper (right plot) for a 1-layered embedding yield lower KT scores than embeddings that consider more layers. However, we do not observe this behaviour in our results (left plot). Instead, we obtain very similar values for all the configurations. Similarly, regarding the Gaussian Random Partition networks we do not obtain the same variations when changing the number of layers. Taking into account the results obtained and the standard deviation values it is difficult to conclude a clear trend.

Finally, it is worth noticing that the main conclusion obtained from this experiment and also the one given in the original paper is that the model is robust against changes on the embedding dimensions. As we have seen, we also obtain the same general conclusion from our experiments.

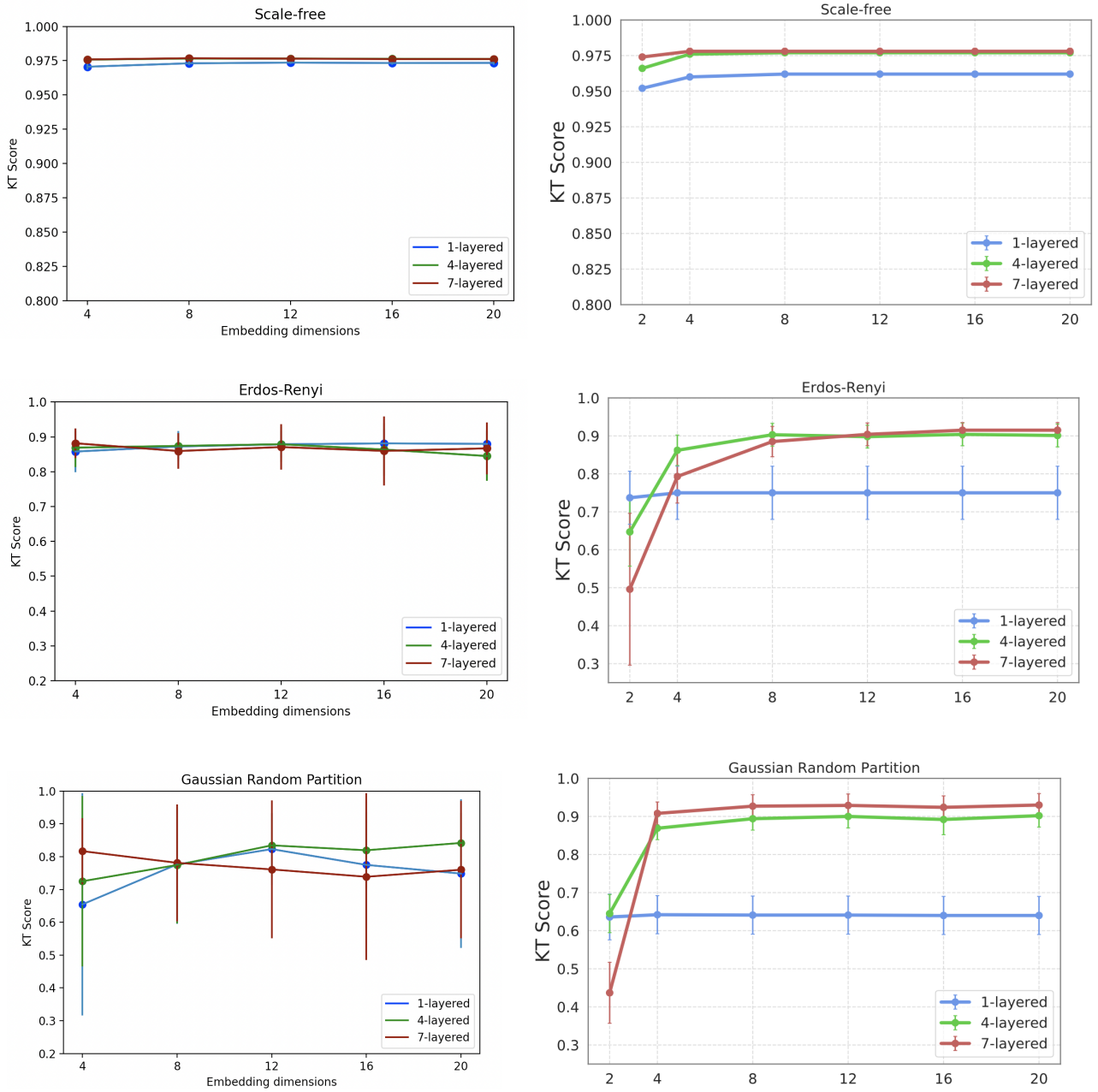

Figure 3.5: **Kendall's Tau coefficient obtained when training and testing over SF, ER and GRP synthetic graphs.** Variations on the number of model layers and embedding dimensions are considered. The left plot is our result, while the right one is the result of the original paper [1]. We can observe similarities between the KT values obtained and the variations when changing the embedding dimensions. Some differences are obtained when changing the number model layers.

3.3 Considering new scenarios and experiments

In this section we perform more experiments considering additional scenarios different from the ones explored in the original paper of reference. First, we show some interesting results considering trained and untrained models. Then, we evaluate the model's performance on real networks by training it using synthetic networks that exhibit community structure. Finally, we perform a deeper analysis on the results considering more accuracy metrics.

3.3.1 Prediction with untrained models

It is interesting to show the results of untrained models since in some cases we have obtained a surprisingly high performance on accuracy when no training has been performed. In order to understand this behaviour, we contacted with the authors of [1], obtaining that this is an already known behaviour since it is possible to have some graph information encoded in the output due to the nature of the aggregation operation used by the Graph Neural Network. Similar observations can be found in [11] and [12].

Without prior knowledge of this behavior, one would expect untrained models to yield a KT score close to 0, since the expected KT score between two random sorted lists is close to 0. In order to test the performance of the model when no training is performed we generate 50 randomly initialized models (using different random seeds) for each of the real networks considered. Then we perform a prediction using each of the generated models. After obtaining the results from the untrained models, we train all the models for one epoch using the same set of Scale-free networks that were used to evaluate the model's performance on real networks. At this point we are able to compare the differences in the KT scores obtained with the untrained and trained models. The result of this comparison is show in Figure 3.6.

As it can be seen in Figure 3.6, when the models are not trained (left plot) we obtain an uniform range of results covering all possible KT score results. The KT scores range uniformly from -1 to 1, and therefore in some cases we obtain a good KT score without training. However, it is interesting to notice that the average KT score of all the results when using untrained models is 0, which is the expected value. In addition, an important observation is that when the models are trained we do not obtain such variations on the KT scores. As it can be seen in the right plot, in all the cases the KT scores are close to 1 and have a small standard deviation. Therefore, we can conclude that the model is really learning and the results obtained when training the model are not merely random.

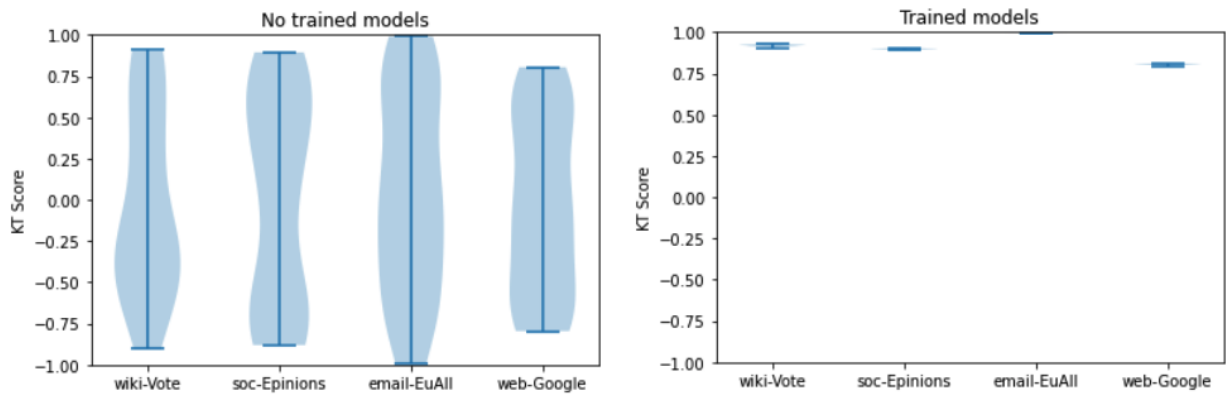

Figure 3.6: **Kendall's Tau coefficient obtained when testing with trained and untrained models for the different real graphs considered.** We can observe a wide range of KT scores for untrained models. Conversely, small variations on KT score are obtained for trained models.

3.3.2 Training with LFR Networks

Real networks are commonly formed by a characteristic structure based on communities. In the paper on which our work is based, the model that predicts over real networks is trained using scale-free networks based on the assumption that scale-free networks have similar properties with real networks. However, in our analysis, we extend beyond this assumption by examining the model's performance when trained using networks that display a community structure, particularly LFR networks. LFR (Lancichinetti-Fortunato-Radicchi) networks are a type of synthetic network model specifically designed to mimic the community structure observed in real-world networks. They are generated based on a generative algorithm that allows for the control of community sizes, degree distributions, and mixing patterns. Then, the main idea of this experiment is to analyse the performance of the model when testing over real networks and training with LFR networks instead of scale-free networks with the hope of improving the resulting KT scores.

First of all, before making predictions over real networks, we perform a baseline experiment and we analyse the performance of the model when training with LFR networks and predicting also over LFR networks. The experiment performed is the same as we did when analysing the performance on synthetic graphs. Here, 15 LFR networks have been obtained where 5 graphs will be used for training and 10 for testing. Similarly to previous experiments, we test the performance when varying the replication factor for the training graphs. The parameters used for the LFR graph generation are shown below:

- Graph generation: The function used for the graph generation corresponds to the `Networkx` Python library and it is `LFR_benchmark_graph()`. For a detailed explanation about the

generation of the graphs please check the `Networkx` official documentation.

- $N = 10,000$. This is the number of nodes in the created graph.
- $\tau_1 = 3$. Power law exponent for the degree distribution of the created graph. This value must be strictly greater than one.
- $\tau_2 = 1.5$. Power law exponent for the community size distribution in the created graph. This value must be strictly greater than one.
- $\mu = 0.05$. Fraction of inter-community edges incident to each node. This value must be in the interval [0, 1]. Please note that we choose a small value in order to have bridges between communities.
- average_degree = 6. Desired average degree of nodes in the created graph. This value must be in the interval [0, n].
- min_community = 20. Minimum size of communities in the graph.

The results obtained using a replication factor of 1, 10, 20 and 40, yielding 5, 50, 100 and 200 training graphs are shown below in Table 3.10.

5 training graphs	50 training graphs	100 training graphs	200 training graphs
0.7125 ± 0.0082	0.7529 ± 0.0057	0.7576 ± 0.0057	0.7639 ± 0.0063

Table 3.10: **KT values for LFR synthetic graphs considering variations on replication factor.**

As it can be seen in Table 3.10 there is a clear improvement in the KT scores when increasing the number of training graphs from 5 to 50. In fact, we obtained this same result when performing the same experiment for ER and GRP graphs in Section 3.2.1. However, there are not relevant KT score variations when we keep increasing the replication factor beyond 10, since we obtain similar values for the experiments that consider 50, 100, and 200 training graphs. In addition, it is interesting to notice that the KT values obtained in this experiment where we train and test on LFR graphs are lower compared to the results obtained with the other types of synthetic graphs considered previously in this work. As a result, the model faces greater challenges when learning from LFR graphs. This finding is especially significant because it emphasizes the variations in model performance when dealing with different types of graphs.

With the aim of comparing the performance over real networks when training with LFR or SF networks we perform the following experiment. We consider 5 SF and 5 LFR graphs of 10,000 nodes each to train each model independently. We then consider a replication factor of

1, 10, 20 and 40 yielding training sets of 5, 50, 100 and 200 graphs respectively. Additionally, in order to obtain standard deviation values we consider 15 different random seeds to initialize the models during training. Finally, we train a model for each of the combinations of the different parameters involved, effectively generating 480 different models, which will be used to obtain the results. Considering this vast amount of models and our limited computational power, we train only for one epoch, since we have seen (and this will be further validated in the upcoming experiments) minor variations when increasing the number of epochs if we consider enough training graphs. As will be observed, the standard deviation values diminish as the number of training graphs is increased. However, it is important to analyze the following results having in mind that they are based on a single epoch only. The results obtained are shown below, in Table 3.11.

		Training graphs			
Graph	Train set	5	50	100	200
wiki-Vote	SF	0.7688 ± 0.3429	0.9199 ± 0.0051	0.9199 ± 0.0046	0.92 ± 0.0042
wiki-Vote	LFR	0.8373 ± 0.152	0.9202 ± 0.0046	0.917 ± 0.003	0.9157 ± 0.0018
soc-Epinions	SF	0.5198 ± 0.6069	0.8995 ± 0.0019	0.8985 ± 0.0018	0.8948 ± 0.0023
soc-Epinions	LFR	0.5679 ± 0.5921	0.8987 ± 0.0019	0.8958 ± 0.0012	0.8947 ± 0.0009
email-EuAll	SF	0.7004 ± 0.4462	0.9945 ± 0.0004	0.995 ± 0.0003	0.9959 ± 0.0001
email-EuAll	LFR	0.7038 ± 0.4062	0.9954 ± 0.0002	0.9961 ± 0.0	0.9962 ± 0.0
web-Google	SF	0.715 ± 0.1079	0.8069 ± 0.0023	0.8074 ± 0.0021	0.8079 ± 0.0035
web-Google	LFR	0.7432 ± 0.0795	0.8094 ± 0.0011	0.8142 ± 0.0011	0.8172 ± 0.0006

Table 3.11: **LFR vs SF training. Testing over real graphs for one epoch.**

As we can see in Table 3.11, there is an increase of the KT scores when the number of replications is increased from 1 to 5 (5 to 50 graphs) in all cases. Then, similarly to the last experiment where we kept increasing the replication factor, we obtain minor variations on the accuracy and we do not see a clear pattern. Regarding the differences on the KT score when comparing the LFR and SF trained models we do not obtain a clear difference when considering a number of training graphs greater than or equal to 50. Once again, we observe small variations without a clear pattern.

It is interesting to notice that if we focus on the experiment that uses 5 training graphs with no replication factor, we obtain better average KT results for all the real networks when we train the network using LFR networks instead of scale-free networks. However, as it can be seen, the standard deviation values for this particular experiment are high. This finding is interesting since it means that when the model has less data for training and only one epoch is considered, it performs better on real graphs using LFR networks for training than SF. Finally, it is worth remembering that only one training epoch has been considered for the results obtained.

However, as it will be seen on the following experiments there are no significant variations on the results when we consider more epochs given a training set sufficiently large.

With the aim of obtaining more insights and to compare the performance of LFR and SF training models, we perform the following experiments. Taking into account the differences obtained when considering a different number of training graphs we will perform the same experiment considering only 5 training graphs and considering 50 generated (without replication) LFR and SF graphs. Once we obtain the training graphs we train the model for 10 epochs and consider 15 different random seeds to initialize the models. Therefore we generate a total of 150 models for each training set and real graph, since we set a different model size for each real graph as shown before. Taking into account the computational effort required, we only consider the 3 first real networks (we exclude the biggest one) and then a total of 1800 models is generated. The results obtained are shown in Figure 3.7.

Please note that in Figure 3.7 the range of values of the vertical axis of the graphs is different for the experiments using 5 training graphs and the experiments using 50 training graphs. This difference is intentional to effectively visualize the variations in the results.

After analysing the results (see Fig. 3.7), we observe a clear difference when training using 5 graphs compared to 50 graphs. On the one hand, when we consider a larger training set we obtain significantly more stable results and also lower standard deviation values. Then, as commented before, when we use a large enough training set, training for more epochs does not result in relevant differences. On the other hand, it can be seen that when the models are trained using only 5 graphs, the KT values are clearly increased when the epochs are increased in all the cases for the first epochs. In addition, the standard deviation values are also reduced as the number of epochs increases. Besides, it is interesting to mention that there is a point from which the KT values remain stable even if we keep increasing the training epochs for the experiments using 5 training graphs.

Regarding the differences between the performance obtained when training with LFR or SF networks, we do not obtain a clear conclusion. There is not a clear improvement using either of these two types of synthetic graphs for training since we do not always (it depends on the tested graph) obtain a better result for one type of graphs. However, we observe that the results are consistent with the last experiment in which we trained using only one epoch. If we analyse the first point (one epoch) of each of the plots we obtain similar trends to the ones given in the last table. Additionally, it is interesting to note that similar results (for one epoch) are obtained when considering 50 generated graphs and when considering 50 graphs replicated from 5 initial graphs. Similar patterns with unclear trends were also observed when evaluating the model's performance on real graphs in Section 3.2.1.

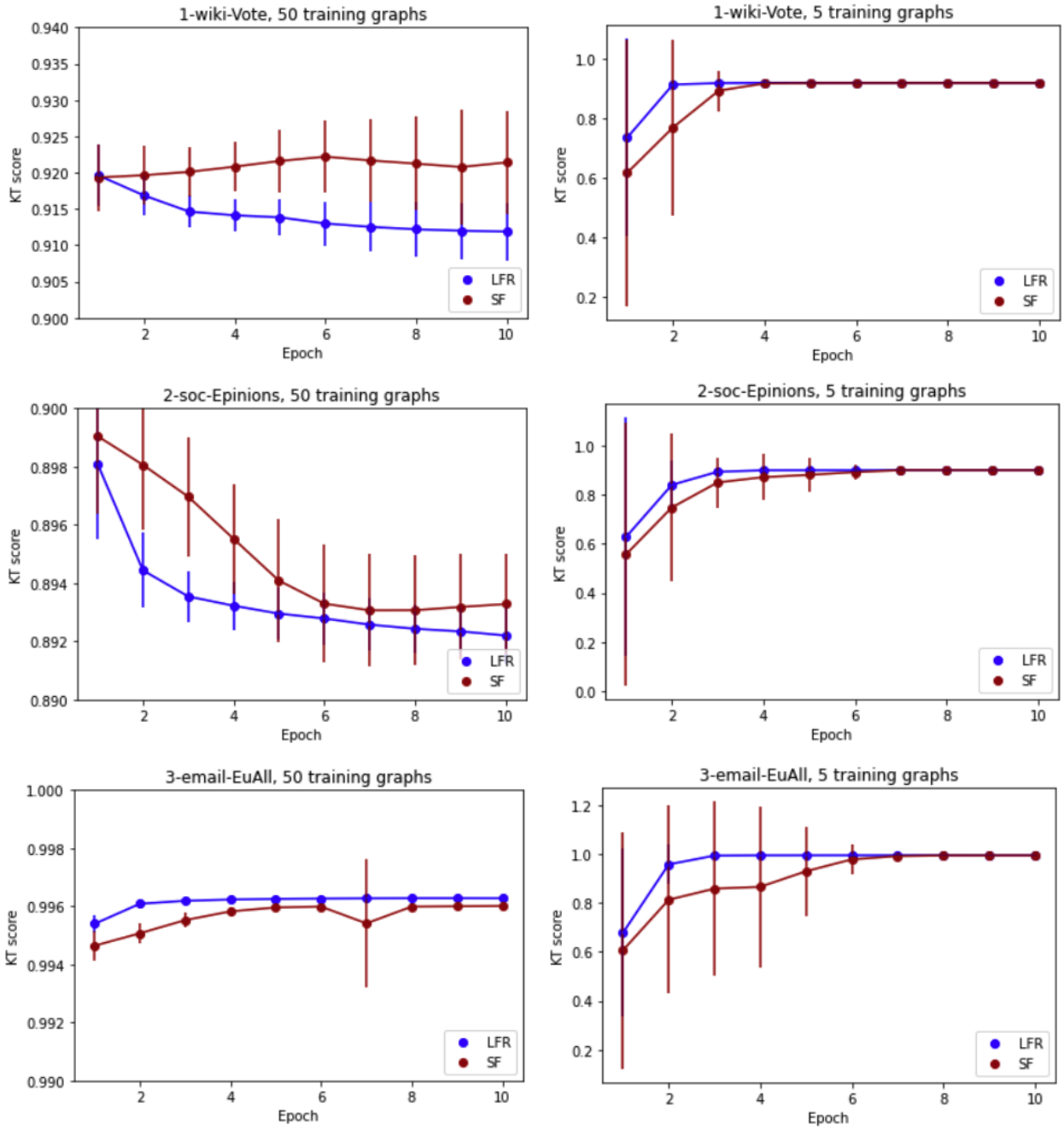

Figure 3.7: **Kendall's Tau coefficient obtained over real graphs when training with SF and LFR networks.** Variations on the number of training graphs, type of training graphs, training epochs considered and model's initialization random seed are considered. We can observe small variations on KT scores in all the cases when 50 training graphs are considered and the epochs are increased for each of the real graphs. Conversely, we obtain higher variations on the results (for each real graph, respectively) when 5 training graphs and less than 4 epochs for training are considered.

3.3.3 Considering other metrics

Taking into account the results obtained (mainly when testing over real graphs) by all the experiments performed, we try to represent the accuracy of the model considering different metrics.

Up to now, the accuracy of the model has been tested in all the performed experiments by computing the betweenness centrality between the output vector of node's betweenness values and the real one focusing on differences over the ranked values. Taking into account that the main purpose is to compare two ranked lists, we try to define some new metrics to analyse the accuracy.

When we perform a prediction with the model we obtain two lists that correspond to the prediction and the real betweenness values for all the nodes of the corresponding graph. Once we have the predicted and real lists of betweenness values we can sort the nodes of the two arrays by its betweenness centrality values in descending order. Consequently, we will obtain two new lists where the first position of each list will have the node's label (from predicted or real values) with the highest betweenness centrality value. Therefore we obtain the ranked node labels based on the betweenness centrality values and the two lists will contain exactly the same total number of labels. Considering the two ranked label lists obtained we can define the following two metrics:

- Coincident values: This metric calculates the percentage of labels that appear in the same position in both the predicted and real ranked label lists. For each sublist (1 to 100 % of the sorted label lists) we compute the percentage of nodes or labels that are predicted on the same position of the real ones. For example, consider that a the graph has 5 nodes and the predicted and real ranked label lists are $[3, 0, 1, 4, 2]$ and $[1, 0, 3, 2, 4]$. If we consider the first 80% of the nodes (that is, positions from 1 to 4), the score obtained is $\frac{1}{4} = 25\%$ because only the label '0' is classified in the same position in both lists. In this example, if we consider the complete lists we obtain that 20% of the nodes were predicted on the real ranked position.

- In-top values: This metric calculates the percentage of predicted nodes or labels that appear in the real percentage of top labels. For each sublist (1 to 100% of the sorted label lists) we compute the percentage of predicted nodes or labels that are present in the real percentage of top labels. For instance, using the last example we would obtain a value of $\frac{3}{4} = 75\%$ considering the first 80% of the nodes, because labels '3', '0', and '1' are present in both sublists. It is interesting to notice that this metric will tend to 100% as we consider more nodes, since the two sublists will have the same labels when all the nodes are included.

Taking into account the previous definitions, we compute the two described metrics for all the results obtained from the previous two experiments. In these experiments, we used 5 and 50 SF and LFR training graphs, considering the results obtained using 15 different random seeds to obtain the standard deviation. In addition, we consider the results obtained for the highest number of training epochs computed (10). Then, we compute the metrics for the three real graphs considered. The metrics were calculated for different subsets of the ranked node lists, ranging from the first 1% to the full 100%. The results obtained are shown in Figure 3.8.

The first conclusion we can arrive to (see Fig. 3.8) is that there are not clear differences on the metric values obtained between LFR trained models and SF trained models. In addition, and similarly to the last results obtained, we obtain significantly higher standard deviation values when we consider models trained only with 5 graphs if we compare with the results obtained when using 50 training graphs. Regarding the metric values obtained, it can be seen that the "equal" (i.e. "coincident values") scores are always below the "in_top" metric scores. This is not surprising since it is more difficult to predict the exact raking position of each node than predicting, for example, if a node is included in the 10% top values.

On the one hand, the "in_top" metric starts at a significantly higher point of the vertical axis than the "equal" metric. Moreover, it increases rapidly as the percentage of nodes considered increases up to 100%. Then, it remains at 100% until all the nodes are considered. This is an interesting result since it means that the differences between the predicted and sorted ranked nodes are at the beginning of the lists and once the metric reaches the 100% there are no differences between the lists. This behaviour will be understood when analysing the percentage of nodes with zero Betweenness Centrality values.

On the other hand, regarding the "equal" metric it is interesting to show that, when considering a small part of the nodes, the results are close to 0 and there is not a general good prediction on exact ranked positions. However, there is a point from which the metric starts increasing when considering more percentage of sorted nodes.

It is interesting to notice that the same patterns have been obtained for all the real networks under consideration. Indeed, all the results obtained show a clear phase transition for the two metrics when the percentage of sorted nodes considered is increased. In addition, the phase transition takes place, on each graph, at the same point for the two metrics considered. However, it is worth noticing that the observed phase transition does not appear always at the same value of the percentage of nodes for the three networks considered. Instead, it seems to take place at the point in which the "in_top" metric reaches the 100%, which corresponds to 33% of sorted nodes considered for the two first real graphs and the 2% for the *email-EuAll* graph.

To further investigate the cause of the observed phase transitions, we analyse the different label vectors obtained and their corresponding betweenness centrality values. We find that the

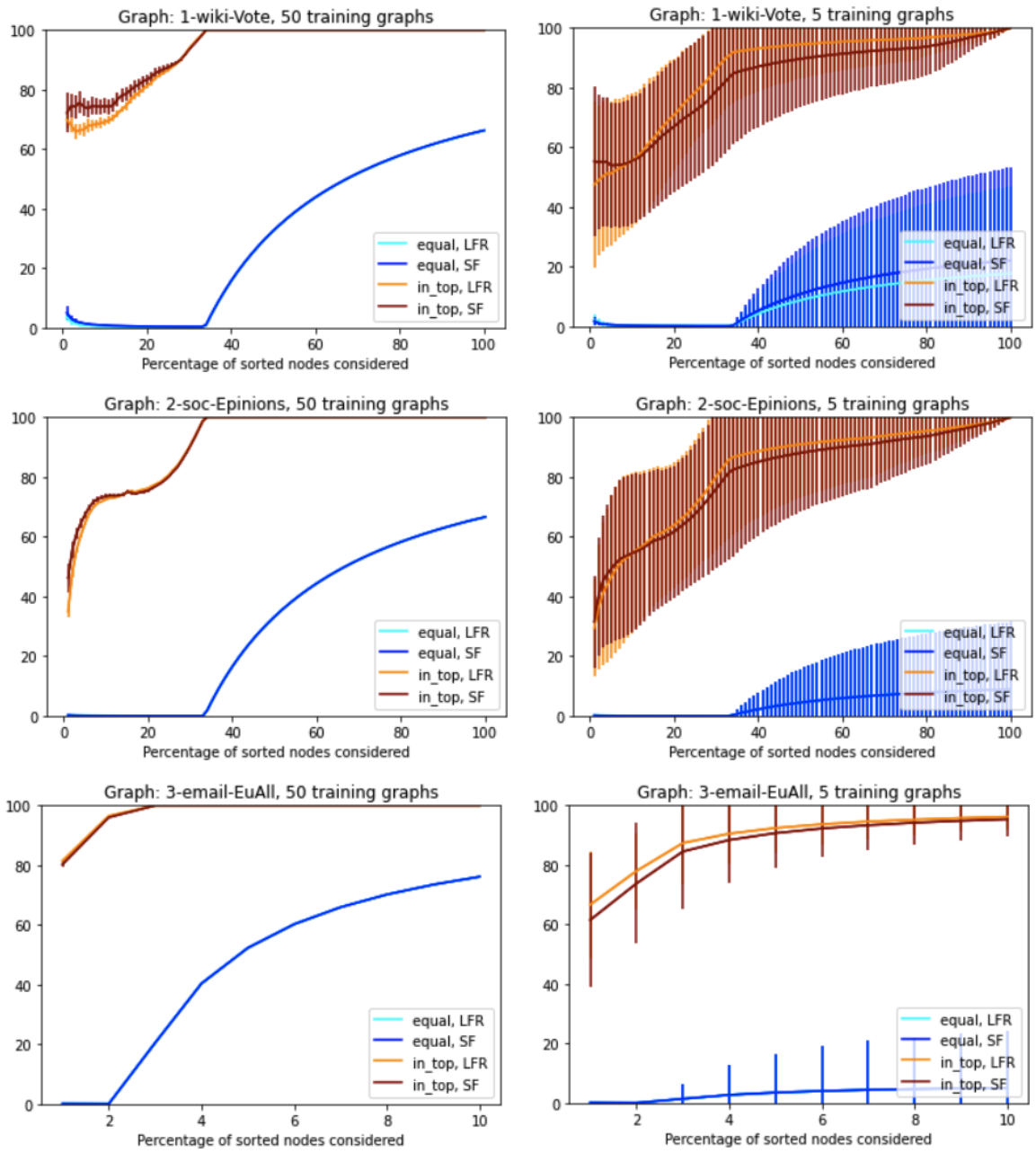

Figure 3.8: **Accuracy metrics over the different real graphs considered using the results obtained in Fig. 3.7**. The metrics were calculated for different subsets of the ranked node lists, ranging from the first 1% to the full 100%. No clear differences on metric's values are obtained between SF and LFR trained models. A clear phase transition is observed for each of the real graphs considered in all the results. It is worth to notice that the horizontal axis range from 1 to 10 in email-EuAll plots (due to its size) to effectively visualize the variations in the results.

phase transition point matches the point in which the nodes start having a zero betweenness centrality values. Table 3.12 shows, for each graph, the number of nodes having 0 betweenness

centrality.

Graph	Total Nodes	Nodes with zero BC	Percentage of nodes with zero BC
wiki-Vote	7115	4711	66.21%
soc-Epinions	75879	50540	66.61%
email-EuAll	265214	258884	97.61%

Table 3.12: **Nodes with a score of zero betweenness centrality.**

As it can be seen in table 3.12 , we find that in the different real graphs considered, an important percentage of nodes have 0 betweenness centrality. If we remember the data preparation process explained in Section 3.1.2, some of the nodes with 0 betweenness centrality (N_z) are identified beforehand, in order to avoid the feature aggregation process through these nodes. Taking into account that these zeros correspond to trivial zeros (since they are identified by certain logical rules) we analyse the proportion between N_z and all the zero BC nodes, finding that all the nodes with zero BC are identified beforehand. Since the accuracy of the model is computed (using the Kendall's Tau coefficient) considering all the nodes of the different graphs, we raise the concern that the results obtained are directly affected by this high percentage of trivial values. Consequently, we show the same KT results obtained for the last experiment but now without considering the zero BC nodes. The results obtained are shown in Figure. 3.9.

As it can be seen in Figure. 3.9, the KT values are significantly reduced, as expected, when considering only the non-zero BC nodes instead of considering all the nodes. Taking into account the results obtained we see that the model is not so good as expected when training with LFR or SF graphs and predicting over real graphs. This result is obtained because of the high percentage of trivial zeros that were considered when computing the KT. On the other hand, we continue to observe the same patterns as previously noted when comparing training with LFR and SF networks, regardless of whether we consider 5 or 50 training graphs.

Finally, it is worth mentioning that in the original paper [1] there is an analysis to find the proportion of nodes with zero Betweenness Centrality. The paper reveals a substantial proportion of zero BC nodes for both real and SF graphs. However, despite this finding, all the nodes are considered when computing the KT coefficient at the code provided in the paper.

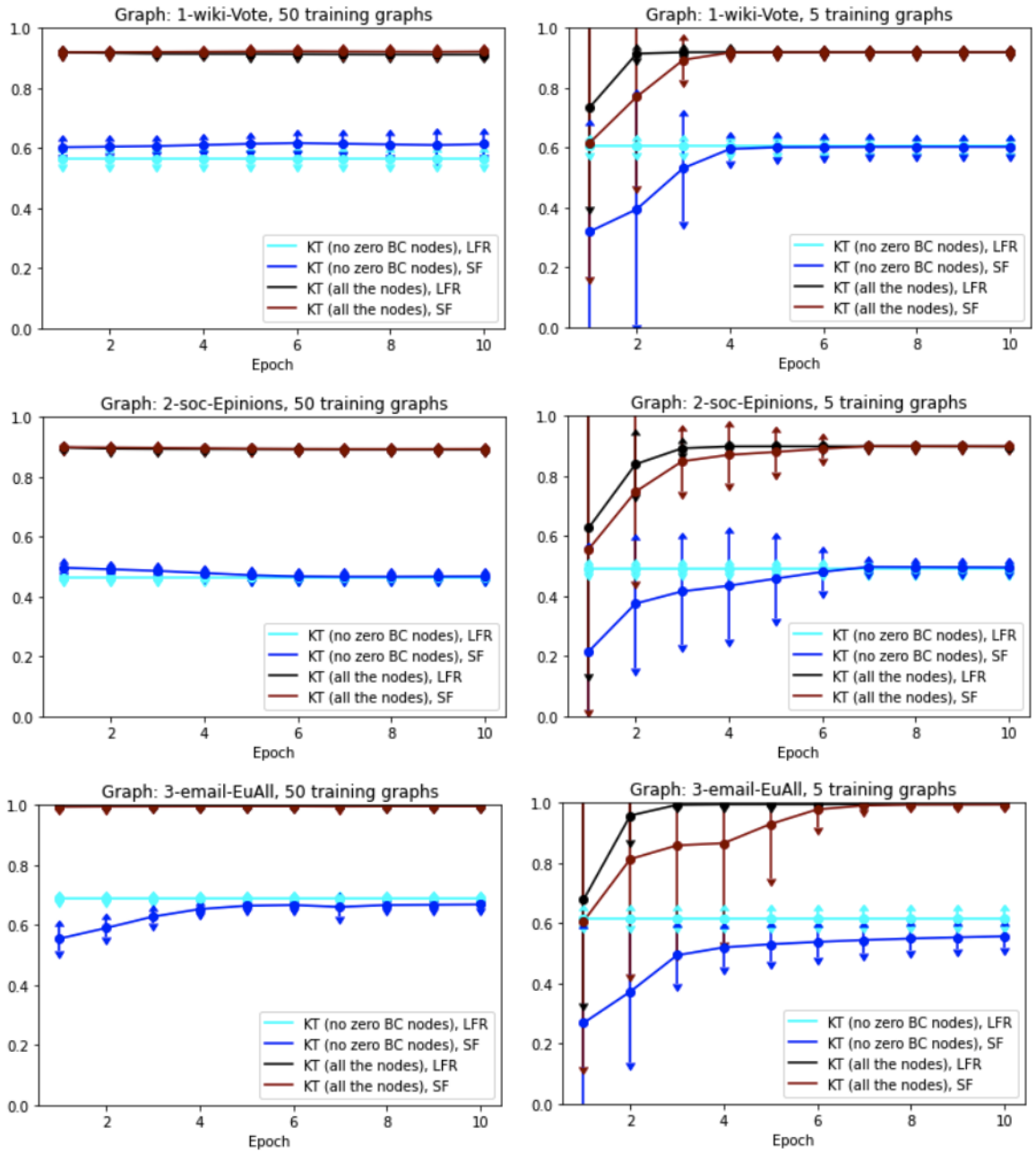

Figure 3.9: **Comparison between the KT scores obtained including or excluding the zero betweenness nodes.** We can observe that the KT score values are significantly reduced when the nodes with zero BC are not included on the accuracy calculation for the three real graphs considered.

3.3.4 Analysis of the different graphs used

Taking into account the previous finding that the majority of graphs had a non-negligible amount of nodes with zero BC value, we aim to further explore the features of the graphs used in the paper. Therefore, we generate 10 graphs for each of the synthetic type of graphs

considered, taking into account the boundary parameters already explained respectively for each type of graph. Then, we compare the results obtained with the plots shown by [1] regarding the different synthetic graphs used.

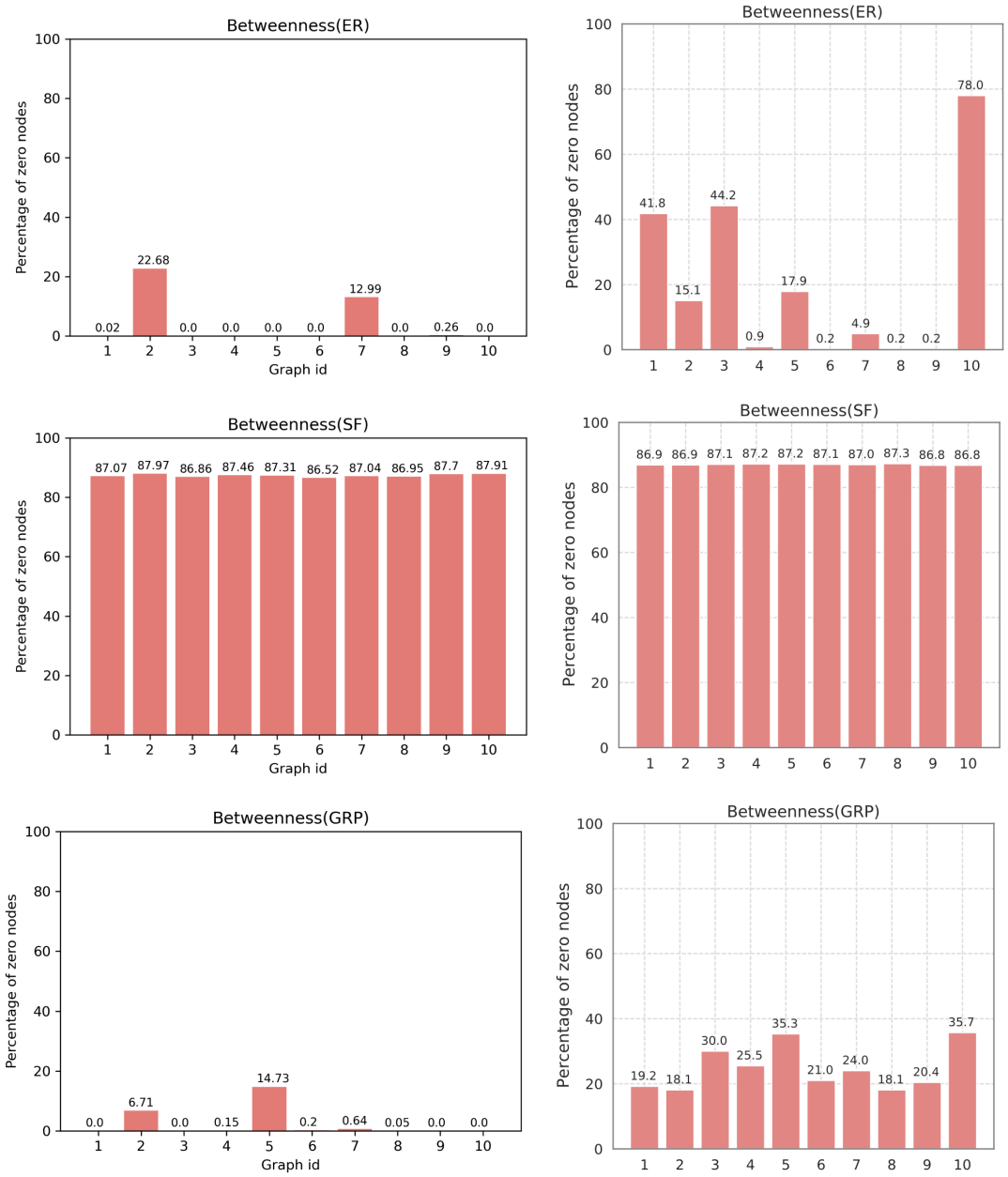

Figure 3.10: **Percentage of nodes with zero Betweenness Centrality for the different synthetic graphs considered.** At the right the graphs used by [1]. At the left the graphs generated in this work. Each pair of plots correspond to a type of synthetic graph (SF, ER or GRP) where each *Graph id* corresponds to a single generated graph. We observe differences between the graphs used by the paper and the ones generated in this work for ER and GRP graphs. Conversely, similar results are obtained when SF graphs are considered.

As it can be seen in Figure 3.10, there are clear differences regarding to the nodes with zero betweenness centrality. On the one hand, we obtain clearly a lower percentage of zero BC nodes when generating the ER and GRP synthetic graphs using the parameters considered in this work. On the other hand, we obtain a similar zero BC nodes percentage when considering SF graphs. Then, it is worth to notice that, taking into account the results obtained on the last experiments and the percentage of zero BC nodes obtained by [1], the zero BC nodes can increase the accuracy obtained by the model.

Chapter 4

Conclusions

To begin the conclusions, it is important to highlight that the main objective of this project has been successfully accomplished. We have been able to predict the Betweenness Centrality of graphs using Graph Neural Networks. To do so, extensive research was conducted, to thoroughly analyse the GNN model introduced by [1], leading to a deep understanding of its architecture and its behaviour. Then, we conducted a series of experiments to replicate the experiments and the analysis shown in [1]. Furthermore, we explored new scenarios by performing additional experiments, yielding noteworthy results and conclusions.

On the one hand, regarding the predictions performed over synthetic graphs, the following observations were made:

- Similar Kendall Tau values are obtained between our results and the ones shown by the paper of reference when sufficient training graphs are considered.
- Increasing the replication factor improves the accuracy of the trained models.
- Variations on the generation parameters lead to different training sets, which, in turn, can lead to differences on accuracy results.
- Regarding the ablation tests, there are variations between the obtained results and those presented in the paper.

Consequently, our main conclusion when considering synthetic graphs is that the accuracy of the model strongly depends on the graphs used for testing and training. There is no universal model accuracy that applies to all graphs. In addition, considering the differences observed in the ablation experiments, we can conclude that it is difficult to show a general model's behaviour since differences on the graphs used lead to differences on the results obtained.

On the other hand, regarding the predictions performed over real graphs we obtain the following:

- We obtain high Kendall's Tau values for most configurations tested, when we calculate the KT score using all nodes.
- We do not obtain clear patterns when varying the parameters at the different experiments performed regarding real graphs.
- The choice between training the model using LFR graphs or SF graphs does not appear to significantly impact the accuracy when testing on real networks. Our experiments revealed that the performance of the model was similar regardless of the type of synthetic graphs used for training.
- When introducing new metrics and analysing the model's performance, we observed an interesting phase transition that highlighted a significant percentage of trivial solutions being considered.
- When we consider only the non-trivial predictions, we observe a decrease in the model's performance.

Based on the last observations, our main conclusion is that, when we consider graphs with a high percentage of trivial BC zeros, the accuracy of the model is misleadingly increased due to the inclusion of trivial solutions. However, when we consider the non-trivial solutions only, the performance of the model over real graphs is substantially reduced. Let us note that in this analysis we did not observe clear differences in the accuracy obtained when training with LFR or SF graphs.

Finally, considering all the factors that can lead to changes on train and test graphs such as the random seeds used for graph generation, the boundaries of the parameters selected for each graph or differences on graph size and variability considered, we conclude that to obtain stronger and more general results about the performance of the model we should consider larger experiments with more variability. This can be achieved by running tasks in parallel and using computational resources with higher processing power. By considering a broader range of graph types and sizes, we can enhance the comprehensiveness and reliability of our findings. Nevertheless, the findings presented in this work provide a foundation for future research and optimization of graph neural networks for accurate predictions and analysis of network structures, and in particular, of its Betweenness Centrality values.

Bibliography

[1] Sunil Kumar Maurya, Xin Liu, and Tsuyoshi Murata. Graph neural networks for fast node ranking approximation. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(5):1–32, 2021.

[2] Mark E.J Newman. *Networks*. Oxford University Press, 2 edition, 2019.

[3] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[4] By: IBM Cloud Education. What is machine learning?

[5] By: IBM Cloud Education. What are neural networks?

[6] Zhiyuan Liu and Jie Zhou. *Introduction to graph neural networks*. Morgan amp; Claypool Publishers, 2020.

[7] Michael A. Nielsen. How the backpropagation algorithm works, Jan 1970.

[8] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[9] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

[10] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B. Wiltschko. A gentle introduction to graph neural networks, Sep 2021.

[11] Thomas Kipf. How powerful are graph convolutional networks?, Sep 2016.

[12] Haochen Chen, Syed Fahad Sultan, Yingtao Tian, Muhao Chen, and Steven Skiena. Fast and accurate network embeddings via very sparse random projection. In *Proceedings of*

the 28th ACM international conference on information and knowledge management, pages 399–408, 2019.