Graph Neural Networks for Fast Node Ranking Approximation

SUNIL KUMAR MAURYA, Tokyo Institute of Technology
XIN LIU, AIRC, AIST
TSUYOSHI MURATA, Tokyo Institute of Technology

Graphs arise naturally in numerous situations, including social graphs, transportation graphs, web graphs, protein graphs, etc. One of the important problems in these settings is to identify which nodes are important in the graph and how they affect the graph structure as a whole. Betweenness centrality and closeness centrality are two commonly used node ranking measures to find out influential nodes in the graphs in terms of information spread and connectivity. Both of these are considered as shortest path based measures as the calculations require the assumption that the information flows between the nodes via the shortest paths. However, exact calculations of these centrality measures are computationally expensive and prohibitive, especially for large graphs. Although researchers have proposed approximation methods, they are either less efficient or suboptimal or both. We propose the first graph neural network (GNN) based model to approximate betweenness and closeness centrality. In GNN, each node aggregates features of the nodes in multihop neighborhood. We use this feature aggregation scheme to model paths and learn how many nodes are reachable to a specific node. We demonstrate that our approach significantly outperforms current techniques while taking less amount of time through extensive experiments on a series of synthetic and real-world datasets. A benefit of our approach is that the model is inductive, which means it can be trained on one set of graphs and evaluated on another set of graphs with varying structures. Thus, the model is useful for both static graphs and dynamic graphs.
Source code is available at https://github.com/sunilkmaurya/GNN_Ranking

CCS Concepts: • **Computing methodologies** → *Machine learning*; *Learning paradigms*; *Supervised learning*; *Learning to rank*; *Machine learning approaches*; *Neural networks*; • **Mathematics of computing** → *Discrete mathematics*; *Graph theory*;

Additional Key Words and Phrases: Betweenness centrality, closeness centrality, graph neural networks (GNNs), node ranking, dynamic graphs

ACM Reference format:
Sunil Kumar Maurya, Xin Liu, and Tsuyoshi Murata. 2021. Graph Neural Networks for Fast Node Ranking Approximation. *ACM Trans. Knowl. Discov. Data* 15, 5, Article 78 (May 2021), 32 pages.
https://doi.org/10.1145/3446217

Also with AIST-Tokyo Tech Real World Big-Data Computation Open Innovation Laboratory.
This work was supported by JST CREST (Grant Number JPMJCR1687), JSPS Grant-in-Aid for Scientific Research(B) (Grant Number 17H01785), JSPS Grant-in-Aid for Early-Career Scientists(Grant Number 19K20352), and the New Energy and Industrial Technology Development Organization (NEDO).
Authors' addresses: S. K. Maurya and T. Murata, Department of Computer Science, School of Computing, Tokyo Institute of Technology, W8-59 2-12-1 Ookayama, Meguro, Tokyo, Japan - 152-8552; emails: skmaurya@net.c.titech.ac.jp, murata@c.titech.ac.jp; X. Liu, Artificial Intelligence Research Center, AIST, AIST Waterfront ANNEX, 2-4-7 Aomi, Koto-ku, Tokyo, Japan - 135-0064; email: xin.liu@aist.go.jp.

1556-4681/2021/05-ART78
https://doi.org/10.1145/3446217

1 INTRODUCTION

Graphs provide a simple way to abstract the data into simple structures that are easy to understand and convenient to process. They preserve the flow of information and interaction patterns in real-world data. Thus, graphs can be used to uncover properties of the data and gain meaningful knowledge from it. One of the crucial tasks is to find the ranking of the nodes in graphs in terms of information spread and connectivity [50, 54]. Node ranking helps narrow down our focus from a huge population of data points to small and more relevant data points. Based on the situation, it can help us to take appropriate actions. For example, in the case of infectious diseases [13, 70], it can help to find susceptible people who are more prone to spread disease once infected. Targeted immunization can curb the disease spread in its infancy and before it becomes an epidemic while minimizing resource utilization. Similarly, in social networks, spread of misinformation can be prevented [61]. In road and power infrastructures, finding critical points can help to take preemptive actions to make systems more robust to failures. Thus, the study of node ranking is an essential part of graph analysis.

In general, researchers have proposed centrality measures for node ranking and these measures provide us a quantitative view of how nodes play a certain role in the graph and how their presence or absence affects the graph structure as a whole. Some of the proposed measures are betweenness centrality [10, 31, 57], closeness centrality [2, 8, 66], PageRank Centrality [12, 23, 29, 34], and so on. All of these measures rank nodes based on certain criteria. In this article, we will focus on two widely used shortest path based measures: betweenness centrality and closeness centrality.

Betweenness centrality ranks nodes based on how they facilitate the information flow in the graph. Removal of nodes with high betweenness can disrupt the flow of information in graph structure significantly [8]. Betweenness centrality has been used for various purposes such as analysis of knowledge networks [62], contingency analysis in power grid systems [41], the study of biological graphs [42], and traffic monitoring in transportation networks [63].

Closeness centrality is a measure of how close a given node is with respect to all other nodes in the graph. It determines the ease with which a node can propagate information across the graph. Closeness centrality has been used to identify influential nodes in social network community [55], and to determine the expected arrival or hitting time for information or disease spreading through the community [8, 9].

Both betweenness centrality and closeness centrality require the calculation of the shortest paths in the graph. As such, calculating the exact centrality value is prohibitive for large graphs in practice, many approximation algorithms based on sampling techniques have been proposed. Geisberger et al. [32] propose a method to approximate betweenness centrality by sampling k starting nodes (pivots) to find the shortest paths and estimate betweenness centrality for all nodes. Riondato et al. [65] take a different approach while providing a theoretical guarantee. r shortest paths are chosen between randomly sampled source–target node pairs, and betweenness is approximated based on these paths. Borassi's algorithm [7] provides an adaptive sampling technique for sampling shortest paths, which provides a faster computation of betweenness within a given absolute error. For approximations of closeness centrality, Eppstein et al. [26] and Okamoto et al. [58] provide algorithms based on single-source shortest path computations on uniformly sampled nodes for a given graph. Cohen et al. [19] propose an improved algorithm based on a hybrid approach.

However, more often, these random sampling-based methods lead to sub-optimal ranking accuracy. There is usually a tradeoff between time and accuracy with higher accuracy requiring more samples, thus higher execution time and vice versa. In this article,[1] we propose the use of **Graph**

[1]This article is an extension of our previous work [53] for approximating betweenness centrality using GNN, accepted at CIKM2019.

Neural Networks (GNNs) to solve the problem of approximation of shortest-path based centrality measures with higher ranking accuracy in a low amount of time. GNN is a type of neural network architecture that leverages graph structure and node/edge feature information to learn node or graph representations that can be used for many different downstream tasks. The general principle behind GNNs is the node feature aggregation scheme along the edges in the graph. In a multilayer GNN model, each node aggregates features of its neighbors, which lie along all the paths starting or ending at the given node. With repetitive aggregation, the resulting node representation captures the structural information of its neighborhood. One prominent example is the use of graph representations to classify input graphs based on their structure [27, 40, 51, 73]. Building on capability of GNN to learn the graph structure, we propose a novel selective feature aggregation scheme based on the shortest paths in the graph. The input graph is preprocessed, and the adjacency matrix is modified such that the node aggregate features over multiple hops along possible shortest paths in the graph. We then use a ranking based loss to learn a scoring function that maps the aggregated information of the node to a score correlated with the centrality measure scores.

We propose two variants of the model GNN-Bet and GNN-Close for approximating betweenness and closeness centrality, respectively. As per our knowledge, we are the first to propose robust GNN based algorithms for this task. One of the significant benefits of our model is that we can use the computational power of GPUs and once trained, the inference time of our model is very small, in order of milliseconds. Another benefit of our approach is that the model training is inductive, which means that it can be trained on one set of graphs and can be evaluated on another set of graphs with varying structures. The trained model can be used to perform ranking approximations for both static graphs as well as dynamic graphs (with multiple snapshot representation). We demonstrate that our approach dramatically outperforms current techniques (while taking less amount of time) through extensive experiments on a series of synthetic and real-world datasets. For betweenness approximation on static graphs, our model is up to 37 times faster than other methods while providing higher ranking performance. We also show that our proposed betweenness variant of the model is suitable for betweenness approximations for dynamic graphs and is up to 14 times faster in providing node ranking approximations. Closeness approximation variant of our model takes a similar time to the comparison method while providing comparable or better approximations. The rest of this article is organized as follows. Section 2 presents the definition of betweenness and closeness centrality and reviews GNN. In Section 3, we propose our approach. Section 4 reports the experiment results. Section 5 surveys related work. Finally, Section 6 gives our conclusion.

2 PRELIMINARIES

2.1 Betweenness Centrality

One of the ways a node can be ranked is based on its ability to control the spread of information between other nodes. Ranking of nodes based on this criterion is called betweenness centrality [10, 31, 57]. The main idea behind betweenness centrality is that a given node is central if many shortest paths pass through it. A high value of betweenness centrality for a node means this node lies on many shortest paths between other nodes and helps to pass information between other nodes. In other words, these nodes act as "bridges" in the graph. Removing these nodes from the graph will lead to longer paths for information to travel between other nodes. For a given graph $G = (V, E)$, betweenness centrality of a node v is defined as

$$\mathrm{BC}(v) = \sum_{u \neq v \neq w} \frac{\sigma_{uw}(v)}{\sigma_{uw}}, \tag{1}$$

where σ_{uw} denotes the total number of shortest paths from node u to w and $\sigma_{uw}(v)$ denotes the number of paths from u to w going through node v.

2.2 Closeness Centrality

Closeness centrality [3, 4, 30] gives a measure of how close a given node is to other nodes. Higher closeness centrality means that the node can spread information easier to other nodes. The closeness centrality of a node v can be defined as

$$\mathrm{CC}(v) = \frac{n-1}{\sum_{u \in V} d(v,u)}, \tag{2}$$

where $d(v,u)$ denotes shortest distance from node v to u.

In simple terms, closeness centrality is proportional to the average of shortest distances from node v to other nodes. Both betweenness and closeness centrality require calculation of the shortest paths in the graph. Calculating the exact centrality value requires time complexity of $\Theta(|V||E|)$, and thus it is prohibitive for large graphs. To overcome this issue, researchers have proposed approximations algorithms based on sampling techniques. However, these are either less efficient or suboptimal or both. For more details, please refer to Section 5.

2.3 Graph Neural Networks

In recent years, many neural network models [17, 24, 45, 68] have been proposed for graph-structured data and generally known as GNNs. In all of these models, the structure of the graph is used to aggregate the feature information of nodes and edges [37]. Based on the feature aggregation pattern, a neural network is trained to predict node labels, edge existence probabilities between two nodes and so on. All of these existing models can be generalized under a single common **Message Passing Neural Network (MPNN)** framework [33].

2.3.1 Framework Design. Let $G = (V, E)$ denote the graph with given node feature vectors as X_v for $v \in V$. A GNN is designed in such a way that it takes the graph G and feature information matrix of nodes/edges as input and is trained against some arbitrary loss function designated for a specific task. Similar to any other neural network architecture, GNN has a predetermined number of layers decided based on the multitude of factors like task in hand, characteristics of input graphs, the diameter of graph, etc.

At any given GNN layer, each node aggregates the features of its neighbors, which can also be considered as a message passing phase. Node feature vector is updated by combining its own feature vector with aggregated features from its neighbors. This aggregation operation is repeated for each layer in GNN. Mathematically, the aggregation operation can be formulated as follows.

$$a_v^{(k)} = \mathrm{AGGREGATE}^{(k)}(\{h_u^{(k-1)} : u \in N_v\}), \tag{3}$$

$$h_v^{(k)} = \mathrm{COMBINE}^{(k)}(h_v^{(k-1)}, a_v^{(k-1)}). \tag{4}$$

In Equation (3), the feature vectors h_u of neighboring nodes are aggregated for kth layer. The AGGREGATE operation can vary based on the model requirement and can represent summing, averaging, or max-pooling of feature vectors. In Equation (4), the aggregate feature vector is then added to the node feature vector. At the end of each layer, the node has cumulative feature information of the node's neighbors in the current layer and all previous layers. Then, the aggregated features are mapped to a learnable weight matrix and a nonlinear transformation like *ReLU* is used. The output of each layer is then forwarded as the input to the next layer. After K number of iterations, the node embedding vector at the final layer captures the structural as well as node feature information of all neighbors from 1-hop distance to kth hop distance.

Our proposed GNN framework is constructed following a similar design as mentioned above. In [71], the authors propose a simplified GNN model by removing weights and non-linearity from layers. The simplified model has similar performance to the standard GNN model in classification tasks. However, in our proposed framework, we keep the learnable weights and nonlinearity between neural network layers, as we found that the model performed better in this case.

2.3.2 Message Passing in GNN. As discussed earlier, each node aggregates feature information of its neighbors for multiple hop distances. This aggregation scheme resembles **breadth first search (BFS)** from a node and the feature information flow through all available paths in the graph. Both betweenness centrality and closeness centrality measures, in their calculation, aggregate node contributions by performing BFS (to find shortest paths) from each node. Similarly, in our proposed message passing scheme, we restrict the flow of feature information along certain paths, which are more likely to be the shortest paths in the graph.

Most straightforward formulation to aggregate feature information is multiplying the adjacency matrix to the feature matrix. As rows of adjacency matrix denote indices of node's neighbors, the process of matrix multiplication is similar to the embedding lookup of neighbors from the feature matrix and adding them together. To implement our message passing scheme, we modify the adjacency matrix and perform feature aggregation operation. In-depth details of the construction of our model with the proposed scheme are explored in Section 3.

3 PROPOSED FRAMEWORK

In Section 2, we observe the similarity in betweenness/closeness centrality formulation and aggregation scheme of GNNs. Taking advantage of this similarity, we propose two variants of GNN-based models that can be trained to approximate betweenness centrality (GNN-Bet) and closeness centrality (GNN-Close) in the graphs. In message passing scheme of GNNs, we observe that each node accumulates the feature vectors of its multi-hop neighbors with the increase in the number of layers or conversely, each node's feature is spread across the graph with each consecutive layer. We use this feature information flow to model paths in the graph. A node that is connected to many other nodes through paths of varying length is able to accumulate more features across the graph.

As both betweenness centrality and closeness centrality measures are based on paths in the graph, we can use GNNs to model the approximation of their calculation. Some other ranking methods like Eigenvector centrality, PageRank centrality are based on iterative methods on the graph. In this article, we have not explored the approximations of these types of centrality measures.

The general construction of our proposed framework is similar to other GNNs; however, there are two key differences in our framework, which aid to approximate betweenness and closeness centrality:

(i) First major difference is the use of constrained message passing scheme. In calculations of betweenness and closeness centrality, we do not consider all paths between nodes but shortest paths. In order to approximate these centrality measures in our model, we restrict the feature information flow through edges that lie along the shortest paths. Unlike other GNN frameworks, where regular adjacency matrix is used, we use a modified adjacency matrix such that the feature aggregation in each layer is confined along possible shortest paths. The modification procedure for betweenness variant and closeness variant differs slightly based on the feature gathering scheme.

(ii) The second main difference is that we do not add the node's own feature vectors with the current layer neighborhood feature aggregation vector. This avoids cumulative feature

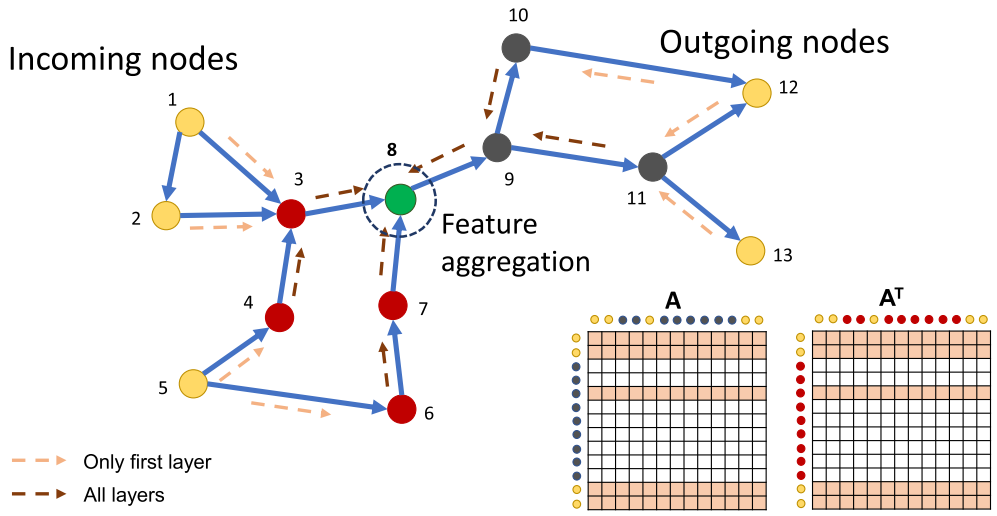

Fig. 1. Figure shows aggregation scheme for betweenness variant on a directed graph. Features of nodes on both incoming and outgoing paths are aggregated at green node. Dashed arrows show direction of feature flow along the edges. Nodes lying on incoming paths are marked red and outgoing paths marked as black. Yellow nodes do not have shortest paths going through and the corresponding rows are set as zero.

aggregation with subsequent layers and at each layer, the node has unique information about its neighborhood at k-th hop.

3.1 Zero Shortest Path Nodes

Both betweenness centrality and closeness centrality measures are based on the calculation of the shortest paths. In our proposed model, we identify beforehand the nodes which have no shortest paths going through them. There are two possible ways to identify such nodes:

(1) When a node has no path going through it. This condition arises when the node either does not have incoming neighbors or does not have outgoing neighbors. For example, in Figure 1, nodes 1 & 5 have no incoming neighbors and node 12 & 13 have no outgoing neighbors. Hence, these nodes have no paths going through them.
(2) When a node's neighbors form cliques such that there is no shortest path going through the node. To identify such nodes, we look at incoming neighbors and outgoing neighbors of a given node. Then, we check of there are edges from each incoming neighbor to all outgoing neighbors. For nodes, where all incoming neighbors are directly connected to outgoing neighbors, there will be no shortest paths going through them. For instance in Figure 1, node 2 has incoming neighbor node 1 and outgoing neighbor node 3. However, node 1 is directly connected to node 3, therefore, there is no shortest path going through node 2.

With following above-mentioned criteria, we can identify nodes with no shortest paths going through them. We refer to these nodes collectively as N_{zp}. In general, GNN aggregation scheme, a node aggregates the feature of neighboring node N_k at k-th hop via a path going through sequence of neighboring nodes lying on consecutively decreasing hop distance from source node i.e., $(N_k \rightarrow N_{k-1} \rightarrow N_{k-2} \rightarrow \cdots \rightarrow N_2 \rightarrow N_1)$. In order to restrict feature information flow along possible shortest paths, we can modify the aggregation scheme such that nodes N_{zp} do not act

as intermediary nodes and no feature information of can flow through the paths that are passing through the nodes N_{zp}.

In rest of the section, we provide details about the construction of both betweenness and closeness variant.

3.2 Betweenness Variant

In a given graph, a node with high betweenness centrality has a large number of shortest paths going through it, and these paths can be considered as a combination of a set of incoming paths and a set of outgoing paths to the node. In betweenness variant GNN-Bet, we use a selective feature aggregation scheme to estimate the number of shortest paths through the node. Incoming paths and outgoing paths are considered separately, and feature aggregation for both types of paths are performed in parallel. The adjacency matrix of the graph is used for outgoing paths and the transpose of adjacency matrix for incoming paths.

3.2.1 Adjacency Matrix. The first step is to identify the nodes N_{zp}, which have no shortest paths passing through them (as discussed previously in Section 3.1). During the feature aggregation step, feature information should not flow through these nodes. To achieve that, we set the rows corresponding to nodes N_{zp} in the adjacency matrix and its transpose to zero (refer Figure 1). With this modification, nodes N_{zp} have no entries for neighbors in their corresponding rows in the adjacency matrix. Hence during the aggregation step, these nodes do not receive any feature information from other nodes and have zero feature information at the output of the layer. This helps to avoid feature flow information through paths that are passing through such nodes as intermediary nodes. Please note in during model initialization all nodes are assigned unique embeddings. In the aggregation step of the first layer, N_{zp} nodes pass their own feature information to neighboring nodes (path length is one). However, in subsequent steps, N_{zp} nodes act as an intermediary (for path length 2 or more) and do not propagate any feature information.

We denote the new modified adjacency matrix as $A_{mod-row}$. The nodes N_{zp} have zero betweenness as per the definition, so with zero feature information, during training, the model learns to distinguish their ranking with respect to other nodes with aggregated feature information.

3.2.2 Model Layer. Initial features of nodes are embedding lookup from the weight matrix of the first layer. At the k-th layer, node aggregates features of its k-hop neighbors. In our model, we define aggregation as the sum of feature vectors. At each layer, when the feature matrix is multiplied with the adjacency matrix, each node sums up the features of all of its neighbors. Then, the aggregated node features from each layer are mapped to a **Multilayer Perceptron (MLP)** unit to output a vector whose values contain a single score corresponding to the node. This MLP learns to predict a single score based on the input features. Single MLP unit is used to output scores for all layers. The output score of all layers are summed separately for incoming and outgoing paths for a node. In our experiments, we use an MLP unit comprising three fully connected layers with ReLU as non-linearity.

The feature vectors of in-degree neighbors and out-degree neighbors are accumulated for multiple hops using two matrices. This aggregation approximates the information of incoming paths and outgoing paths to a given node. Hence using our proposed framework, we get two scores for all in-degree neighbors and out-degree neighbors. As the number of paths passing through a given node is the combinations of incoming paths and outgoing paths through the node, the in-degree and out-degree scores are multiplied in order to get the final score for each node. Figure 3 shows the schematic diagram of the betweenness variant of the model. Pseudo-code of the forward propagation of the model with feature aggregation scheme is presented in Algorithm 1. At line 1 and 2, adjacency matrix and its transpose is modified (row-wise) to get input matrices $\tilde{A}_{\text{out-degree}}$ and

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, $S_{(Bet)}$

1 $\tilde{A}_{\text{out-degree}} \leftarrow \text{ModifyAdjacencyRow}(A)$
2 $\tilde{A}_{\text{in-degree}} \leftarrow \text{ModifyAdjacencyRow}(A^T)$
3 **for** $k = 1...K$ **do**
4 $\quad H^{(k)}_{\text{out-degree}} \leftarrow \text{ReLU}\,(\tilde{A}_{\text{out-degree}} H^{(k-1)}_{\text{out-degree}} W^{(k)})$
5 $\quad H^{(k)}_{\text{in-degree}} \leftarrow \text{ReLU}\,(\tilde{A}_{\text{in-degree}} H^{(k-1)}_{\text{in-degree}} W^{(k)})$
6 $\quad S^{(k)}_{\text{out-degree}} \leftarrow \text{MLP}(H^{(k)}_{\text{out-degree}})$
7 $\quad S^{(k)}_{\text{in-degree}} \leftarrow \text{MLP}(H^{(k)}_{\text{in-degree}})$
8 **end**
9 $S_{\text{out-degree}} \leftarrow \sum_{k=1..K} |S^{(k)}_{\text{out-degree}}|$
10 $S_{\text{in-degree}} \leftarrow \sum_{k=1..K} |S^{(k)}_{\text{in-degree}}|$
11 $S_{(Bet)} \leftarrow S_{\text{out-degree}} \times S_{\text{in-degree}}$

$\tilde{A}_{\text{in-degree}}$ to GNN layer. For each k-th layer {k = 1...K}, $W^{(k)}$ represents the weight matrix and $H^{(k-1)}$ represents the output from the previous layer. MLP is used to map the output node embeddings from each layer to a score vector $S^{(k)}$. Scores are combined from each layer as represented in the line 9–11 to get final centrality scores for all the nodes.

3.3 Closeness Variant

Closeness centrality has two variations: incoming closeness centrality and outgoing closeness centrality, based on consideration of incoming paths or outgoing paths for the nodes. We focus on the outgoing closeness centrality and use a simple adjacency matrix as input in our model. In case incoming closeness centrality is to be approximated, transpose of adjacency matrix can be used in the model.

For closeness variant of model GNN-Close, we consider paths that originate from source node and end at other nodes. The paths in the graph, which go through N_{zp} are not considered, so we need to identify N_{zp} in this case as well. Unlike betweenness centrality, nodes N_{zp} can have non-zero closeness centrality values, so we use different adjacency matrix modification scheme.

3.3.1 Adjacency Matrix. Similar to the betweenness variant, we identify the nodes which have no shortest paths going through them. To ensure no features flow through edges of these nodes to other nodes, we modify the adjacency matrix by setting the columns corresponding to these nodes to zero (refer Figure 2). We refer to this matrix as $A_{mod-col}$. This operation removes the entries of N_{zp} nodes as neighbor of other nodes and avoids any feature aggregation through them when the path length is 2 or more. However, for the first layer, we still aggregate N_{zp} nodes own features. Therefore, we use the normal adjacency matrix for the first layer and modified adjacency matrix for subsequent GNN layers. To summarize, with this scheme, N_{zp} pass their own feature information to other neighbors in first iteration but do not act as intermediary for further feature aggregation in later iterations.

3.3.2 Model Layer. As we use an unchanged adjacency matrix in the first layer, nodes aggregate features of all its neighbors. For further layers, we use a modified adjacency matrix and restrict

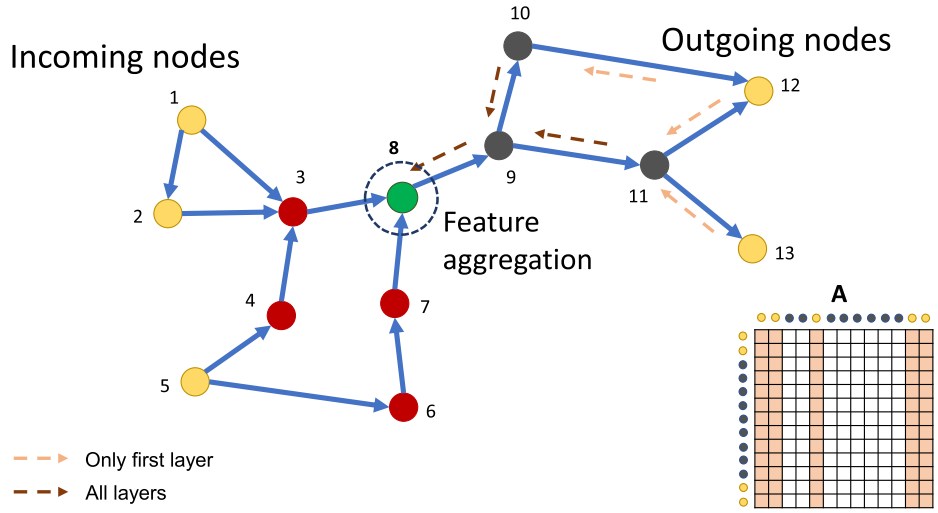

Fig. 2. Figure shows aggregation scheme of closeness variant along outgoing paths on a directed graph. Features are being aggregated at green node. Dashed arrows show direction of feature flow along the edges. Yellow node do not have shortest paths going through them and the corresponding columns are set as zero.

ALGORITHM 2: GNN-Close (Forward propagation)

Input: Directed Graph adjacency matrix A; depth K; weight matrices $W^{(k)}$
Output: Closeness Centrality value vector, $S_{(Close)}$

1 $A_{\text{mod-col}} \leftarrow \text{ModifyAdjacencyCol}(A)$
2 $H^{(1)} \leftarrow AW^{(1)}$
3 $S^{(1)} \leftarrow \text{MLP}(H^{(1)})$
4 **for** $k = 2...K$ **do**
5 $\quad H^{(k)} \leftarrow \text{ReLU}(A_{mod-col}H^{(k-1)}W^{(k)})$
6 $\quad S^{(k)} \leftarrow \text{MLP}(H^{(k)})$
7 **end**
8 $S_{(Close)} \leftarrow \sum_{k=1..K} |S^{(k)}|$

the feature aggregation. Aggregated features of each layer are mapped to an MLP unit to output scores for all layers for each node. Then for each node, output scores from each layer are summed together to get a final score. The output of the model is a score vector. The schematic diagram of GNN-Close is shown in Figure 4 and pseudo-code for the model is described in Algorithm 2. At line 1, input adjacency matrix is modified (column-wise) to output $A_{mod-col}$. Line 2 and 3 represent feature aggregation in the first layer with the normal adjacency matrix, the output of which is mapped to the MLP layer. For the rest of the layers, the modified adjacency matrix is used for feature aggregation. Layerwise score vectors are then combined to get final closeness centrality score vector $S_{(Close)}$ at line 8.

3.4 Loss Function

We use a ranking loss function to calculate the loss for scores predicted by model with respect to actual betweenness or closeness centrality scores. Ranking loss functions are commonly used in

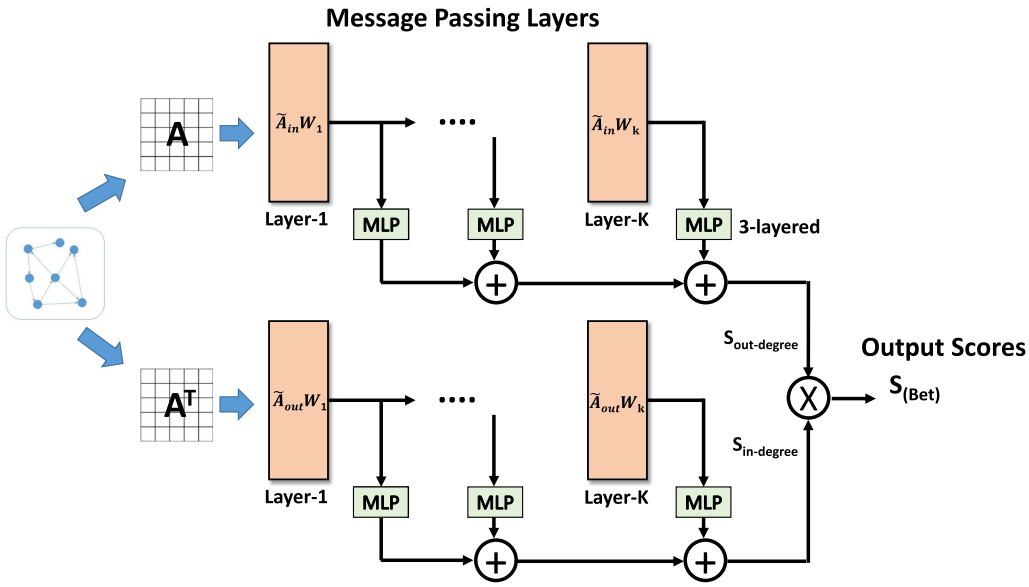

Fig. 3. Figure above shows the model of betweenness variant. Graph input is represented as an adjacency matrix. There are two parallel aggregation pipelines for incoming and outgoing paths for nodes using adjacency matrix and its transpose, respectively. At each layer, node features are aggregated and the output is mapped to a MLP layer. The output of MLP are added together and then subsequently multiplied to get final output scores of nodes.

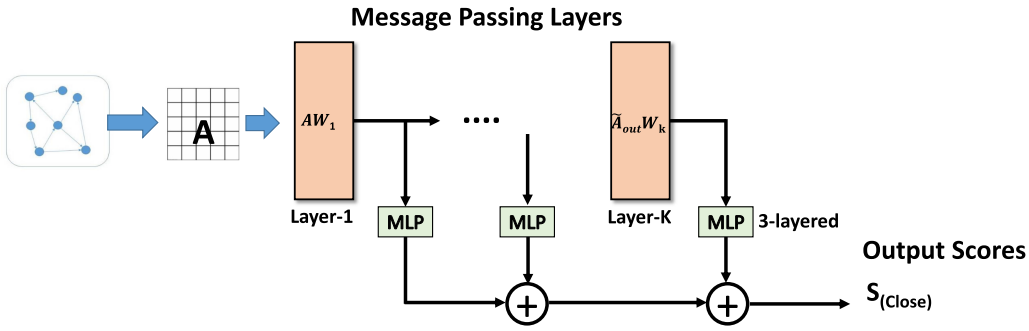

Fig. 4. Figure shows the model of closeness variant. Graph input is represented as an adjacency matrix. For first layer simple adjacency matrix is used and for further layers modified representation of adjacency matrix (for closeness variant) is used. Feature aggregation is performed at each layer and layer output is mapped to a MLP layer. Sum of output vectors from MLP give final node ranking vector.

recommender systems [14, 18]. Using this function helps the MLP to learn to predict the score based on the neighborhood of the node at each layer. For the model score, $S_i^{(m)}$ and actual betweenness score $S_i^{(b)}$, the loss function is defined as follows:

$$\text{Loss}\ (x, y) = \max\ (0, -y * (S_i^{(m)} - S_i^{(b)}) + \text{Margin}), \tag{5}$$

$$y = \begin{cases} 1 & \text{if } S_i^{(m)} \text{ should be ranked higher than } S_i^{(b)}. \\ -1 & \text{if } S_i^{(b)} \text{ should be ranked higher than } S_i^{(m)}. \end{cases}$$

4 EXPERIMENTS

In this section, we present the details of the training and testing of the model. We evaluate our model on synthetic graphs as well as real-world graphs and compare its performance with other betweenness and closeness approximation techniques.

4.1 Training Setup

4.1.1 Hardware and Software Setup. All experiments were conducted on Intel Core i7-8700K CPU @ 3.70GHz machine with 12-cores, 64 GB RAM and a single NVIDIA GeForce GTX 1080Ti GPU with 12 GB graphics memory. Software frameworks used are `PyTorch` [60] for implementing the proposed model, `NetworkX` [36] and `NetworKit` [67] for graph generation and betweenness calculations.

4.1.2 Hyper-parameters. The model size is determined by the largest training/test graph size. For smaller graph inputs, the adjacency matrix of the input graph is placed along diagonal and the rest is padded with zeros. We train the model with Adam as an optimizer with a learning rate parameter set to 0.005. The number of GNN layers in our model is set to 4. Calculation of loss value during training requires comparison of node pair rankings, but the total number of combinations of all node pairs can be really large ($\frac{n(n-1)}{2}$ with n nodes) and computationally prohibitive. In order to avoid this, we randomly sampled node pairs equal to 20 times the number of nodes.

For the betweenness variant, we have used dimension embedding of 12 (limited by GPU memory). For closeness centrality, we set dimension of embedding as 15.

4.1.3 Training. Given set of graphs are divided into training dataset and test dataset. For graphs in both datasets, exact betweenness centrality and closeness centrality values of nodes are calculated. These values are used in target vectors to train the model and to evaluate the performance of the model during inference time for the test dataset. Self-loops are removed from the input graphs while multi-edges are kept, if present in the graphs. We train the model for 5–10 epochs and training time is approximately 10–15 minutes. While training the model, it is possible to have different sizes of graphs for training and test datasets. To accommodate all the graphs, the model size is set to be equal to or greater than the number of nodes of the largest graph in training and test datasets combined. The adjacency matrices of smaller input graphs placed along the diagonals and rest dimensions are padded with zeros to make its size compatible to the input layer.

Note that the test graphs are totally unseen during the training stage (inductive settings). As the model is trained, it learns to map aggregated features for each node via MLP to ranking scores. Once trained, the model is able to take new graphs with different sizes or structures as input and provide node ranking approximations. Hence, the model is very robust and can be deployed in various scenarios.

4.1.4 Evaluation Measure. In order to evaluate the ranking quality of model output, we use Kendall's Tau rank correlation coefficient (refer as KT Score) [44] as a ranking measure. For any given pair (x_i, y_i) and (x_j, y_j), where $i < j$, are said to be concordant if the ranks of both elements in both pairs agree, i.e., both $x_i < x_j$ and $y_i < y_j$ or both $x_i > x_j$ and $y_i > y_j$. If the ranks don't agree, then they are said to be discordant. For two given lists with n items each, let us N_c be the total number of concordant pairs and N_d be total number of discordant pairs, then Kendall's ranking coefficient is calculated by:

$$\tau = \frac{N_c - N_d}{\frac{n(n-1)}{2}}, \tag{6}$$

Table 1. Summary of Real World Datasets Used

Dataset	#Nodes	#Edges
Wiki-Vote	7,115	103,689
P2p-Gnutella31	62,586	147,892
Soc-Epinions	75,879	508,837
Soc-Slashdot	77,350	516,575
Ego-Gplus	107,614	13,673,453
Email-EuAll	265,214	420,045
Web-Google	875,713	5,105,039
Wiki-Talk	1,140,149	7,833,140

τ denotes Kendall's Tau ranking coefficient and its value varies in the range $-1 \leq \tau \leq 1$. Value of 1 means all pairs are concordant (same order) and -1 means all pairs are discordant (opposite order). In our case, we compare the ranking of model output for nodes to the ranking of their real centrality values.

4.2 Dataset Preparation

To test the efficacy of our model on various graphs with unique structural properties and show that the model is able to generalize well, we use extensive set of synthetic as well as real-world graphs. We evaluate and compare our proposed model to other betweenness and closeness approximation methods on these graphs.

4.2.1 Synthetic Datasets. For synthetic datasets, we generate three classes of directed graphs namely, Erdos–Renyi random graphs, Scale-free graphs, and Gaussian random partition graphs. Sizes of graphs vary from 50,000 to 100,000 nodes. For each class, 15 graphs are generated with the number of nodes and generation parameters set randomly to incorporate structural variations of the dataset. More details of graph generation parameters and statistics of graphs are provided in Appendix A. For training of the model, five graphs are chosen to create 500 training samples by randomly permuting the node sequence in adjacency matrices and their corresponding target centrality vector. We train the model on the training set and evaluate the performance on 10 graphs in the test set.

4.2.2 Real-World Datasets. For model evaluation on real-world datasets, we have taken eight datasets from **Stanford Large Network Dataset (SNAP)** website.[2] Table 1 shows the details for these datasets. Real-world graphs have varying structural properties. Under the weak assumption that real-world graphs have some scale-free graph properties, the model is trained on synthetic scale-free graphs. To create training dataset, five synthetic directed scale-free graphs of 100,000 nodes each are generated using `NetworkX`. From these five graphs, 250 adjacency matrices are obtained by permuting the node sequences and corresponding betweenness or closeness centrality vectors. Two-hundred of these adjacency matrices are used for training and 50 for validation.

4.3 Static Betweenness Approximations

We compare the ranking performance and execution time comparison of our model GNN-Bet to the following betweenness approximation methods for static graphs. Parameters used in these methods are as follows:

[2]SNAP Dataset: https://snap.stanford.edu/data/index.html.

Table 2. KT Ranking Scores Comparison Static Betweenness Approximation on Real World and Synthetic Datasets

Dataset	KT Score				
	GS	RK	BN	GNN-Bet	Gain
Wiki-Vote	0.866	0.844	0.900	**0.967**	+0.067 (+7.4%)
P2p-Gnutella31	0.849	**0.963**	0.876	0.924	−0.039 (−4.0%)
Soc-Epinions	0.723	0.826	0.687	**0.891**	+0.065 (+7.8%)
Soc-Slashdot	0.766	0.793	0.639	**0.909**	+0.116 (+14.6%)
Ego-Gplus	0.817	0.673	0.515	**0.821**	+0.004 (+0.4%)
Email-EuAll	0.889	0.508	0.380	**0.990**	+0.101 (+11.3%)
Web-Google	0.755	0.366*	0.498	**0.779**	+0.024 (+3.1%)
Wiki-Talk	0.679	0.204*	0.271	**0.979**	+0.300 (+44.1%)
Synthetic-ER	0.845±0.05	0.829±0.07	0.629±0.14	**0.902±0.03**	+0.057 (+6.7%)
Synthetic-SF	0.464±0.02	0.574±0.03	0.411±0.03	**0.976±0.01**	+0.402 (+70.0%)
Synthetic-GRP	0.562±0.06	0.342±0.17	0.141±0.08	**0.899±0.04**	+0.337 (+59.9%)

Highest KT Score among comparison methods(GS, RK, and BN) is underlined and best ranking performance among all is highlighted with bold font. Performance gain for GNN-Bet is compared with respect to the best KT Score among comparison methods.

—Geisberger [32]: referred as **GS**. For **GS**, we need to provide the number of random nodes to be sampled. We set it to 8,192, which is the maximum number of samples used in the literature even for large graphs.
—Riondato et al. [65]: referred as **RK**. Two parameters are required, $\delta = 0.1$ and $\lambda = 0.005$. These values are used by the authors in their article. However, we found that **RK** was taking too much time for approximation calculations with $\lambda < 0.02$, sometimes approaching/exceeding the exact betweenness centrality calculation time. So far large graphs, we show results for $\lambda = 0.02$ marked with '$*$'.
—Borassi et al. [7]: referred as **BN**. Parameters are set as $\delta = 0.1$ and $\lambda = 0.005$ as used in this article.

In Table 2, we compare the ranking performance of our model with other betweenness approximation methods using KT Score. Similarly, Table 3 shows the comparison of execution time in seconds for betweenness approximations. We also show the time taken by exact betweenness centrality computation algorithm provided by Brandes [10] as a reference to compare all approximate algorithms. Execution time of our model is the sum of time taken to identify nodes with zero betweenness and model inference time. For other algorithms, the execution time is shown. For synthetic graphs, the average KT score (in Table 2) and the average time taken (in Table 3) for 10 test graphs are calculated and presented.

4.3.1 KT Score Comparison. Table 2 shows KT score for betweenness approximation values for our model and other comparison methods. For real-world graphs, all methods have good ranking performance but the KT Score drops for larger graphs. However, **GS** has consistently good performance compared to **RK** and **BN** for all graph sizes. For synthetic graphs, all methods show high ranking performance on ER graphs, while lower KT Scores for SF graphs and GRP graphs. Our model GNN-Bet, consistently performs well on real-world as well as synthetic datasets with up to 44% improvement for real-world graphs and up to 70% for synthetic graphs with respect to the best KT Score on comparison methods.

Table 3. Time Comparison for Static Betweenness Approximation Methods

Dataset	$\text{Time}_{\text{train}}$(s)	Time (s)					
	GNN-Bet	Exact	GS	RK	BN	GNN-Bet	Speedup
Wiki-Vote	633.5	0.8	7.1	1.6	0.2	**0.1**	≈2x
P2p-Gnutella31		57.2	18.06	163.4	5.4	**0.2**	≈27x
Soc-Epinions		222.1	86.5	567.9	6.8	**0.5**	≈14x
Soc-Slashdot		206.7	71.2	510.3	5.96	**0.9**	≈7x
Ego-Gplus		3845.7	979.4	7787.5	11.8	**7.2**	≈2x
Email-EuAll		1091.5	99.8	985.1	8.1	**1.1**	≈7x
Web-Google		52422	2264	8548*	314.4	**8.5**	≈37x
Wiki-Talk		32385	624.2	1115.4*	32.4	**3.6**	≈9x
Synthetic-ER	726.8	811.6±560.1	206.3±142.0	1354.2±765.7	20.2±3.3	**0.6±0.3**	≈33x
Synthetic-SF	317.0	33.81±4.4	5.2±1.7	114.0±30.7	1.8±0.3	**0.3±0.2**	≈6x
Synthetic-GRP	757.8	27.8±12.5	2.9±1.4	98.0±30.7	1.5±0.4	**0.6±0.2**	≈2x

Lowest execution time among all comparison methods is underlined and bold fonts denotes the fastest method. Speedup gains for GNN-Bet are calculated with respect to fastest among comparison methods.

4.3.2 Execution Time Comparison. Execution time comparison is presented in Table 3. We observe that **GS** provides good approximations with reasonable execution time, which is lower than exact betweenness values calculations. For **RK**, the execution frequently approaches/exceeds exact betweenness calculation time if we set parameters for better approximations. The method is therefore good if higher error tolerance is allowed. **BN** is the fastest among all the comparison methods, taking very little time to output approximation values. However, as evident from Table 2, the ranking accuracy drops at the expense of speed. Our proposed model is up to 37 times faster than **BN** for real-world datasets and up to 33 times faster for synthetic graph datasets.

4.4 Dynamic Betweenness Approximations

Our model can readily be applied to dynamic graphs, where nodes and edges changes with time. With these changes, we get multiple slices of graphs. Traditionally, we have to re-calculate betweenness centrality on each slice from scratch, which is computationally expensive. More recent methods overcome this limitation by first calculating betweenness for all nodes and then updating the values as the graph changes. However, in practice, this approach is still slower and may not provide good approximations with larger changes in graph. Since our model training is inductive, we can use the trained model for betweenness approximations to get approximations of all slices of dynamic graphs. In this way, our model can provide more accurate and faster approximations.

We compare our model with the algorithm proposed by [38] referred to as **HA**. In the experiments, for each graph, we initially removed 1000 edges. Then from those 1000 edges, we randomly take 200 edges and add them back to the graph structure while removing 200 different edges from the graph. We repeat this step five times. So at each step, there is a change of 400 edges. We then compare the resulting KT Score and time taken by our model and the comparison method.

Tables 4 and 5 show KT Score and time performance results. Results show significant gains in KT Score (up to 300% improvement), while still being faster (up to 14 times faster). This shows the versatility of the model to be used for both static and dynamic graphs. We note that if the number of edge changes is smaller (number of edge changes in tens or less), **HA** may have execution time comparable to our model. However, with smaller changes in graphs, changes in centrality values are not significant and may not be very useful in real-world applications.

Table 4. KT Ranking Scores Comparison for Dynamic Betweenness Approximation Method on Real World Dataset

Dataset		KT Score					
		Update-1	**Update-2**	**Update-3**	**Update-4**	**Update-5**	**Gain(avg)**
Wiki-Vote	HA	0.8909	0.8964	0.9013	0.9016	0.9031	+0.0647 (+7.2%)
	GNN-Bet	**0.9633**	**0.9636**	**0.9623**	**0.9622**	**0.9655**	
P2p-Gnutella31	HA	0.7233	0.7242	0.7282	0.7269	0.7256	+0.2023 (+27.8%)
	GNN-Bet	**0.927**	**0.928**	**0.928**	**0.928**	**0.929**	
Soc-Epinions	HA	0.5742	0.5719	0.5746	0.5698	0.5706	+0.3117 (+54.4%)
	GNN-Bet	**0.8832**	**0.8837**	**0.8847**	**0.8834**	**0.8846**	
Soc-Slashdot	HA	0.5358	0.5309	0.5368	0.5353	0.5360	+0.3705 (+69.2%)
	GNN-Bet	**0.9048**	**0.9053**	**0.9053**	**0.9057**	**0.9060**	
Ego-Gplus	HA	0.6073	0.6097	0.6126	0.6144	0.6123	+0.2001 (+32.7%)
	GNN-Bet	**0.8111**	**0.8119**	**0.8118**	**0.811**	**0.8111**	
Email-EuAll	HA	0.4312	0.4292	0.4263	0.4273	0.4316	+0.5581 (+130.0%)
	GNN-Bet	**0.9877**	**0.9808**	**0.9889**	**0.9891**	**0.9899**	
Web-Google	HA	0.3001	0.2997	0.2988	0.2998	0.3019	+0.4874 (+162.4%)
	GNN-Bet	**0.7664**	**0.7668**	**0.7673**	**0.7682**	**0.7684**	
Wiki-Talk	HA	0.2389	0.2366	0.2438	0.2402	0.2420	+0.7369 (+306.6%)
	GNN-Bet	**0.9770**	**0.9774**	**0.9773**	**0.9772**	**0.9772**	

Best KT Score are highlighted in bold.

Table 5. Time Comparison for Dynamic Betweenness Approximation Methods

Dataset		Time(s)					
		Update-1	**Update-2**	**Update-3**	**Update-4**	**Update-5**	**Speedup**
Wiki-Vote	HA	3.42	2.81	3.67	3.87	3.14	≈7x
	GNN-Bet	**0.53**	**0.52**	**0.51**	**0.54**	**0.44**	
P2p-Gnutella31	HA	2.07	2.17	2.10	2.07	2.04	≈13x
	GNN-Bet	**0.17**	**0.16**	**0.17**	**0.16**	**0.16**	
Soc-Epinions	HA	10.69	12.74	11.51	13.16	10.77	≈14x
	GNN-Bet	**0.97**	**0.80**	**0.83**	**0.80**	**0.80**	
Soc-Slashdot	HA	5.79	3.82	4.56	4.20	4.20	≈6x
	GNN-Bet	**0.77**	**0.77**	**0.76**	**0.76**	**0.77**	
Ego-Gplus	HA	99.06	74.25	77.25	72.34	77.47	≈12x
	GNN-Bet	**6.94**	**6.82**	**6.91**	**6.86**	**6.25**	
Email-EuAll	HA	7.21	8.25	8.22	3.21	7.59	≈7x
	GNN-Bet	**0.89**	**0.90**	**0.89**	**0.90**	**1.10**	
Web-Google	HA	**4.56**	**4.53**	**4.57**	**4.51**	**4.61**	≈0.8x
	GNN-Bet	5.45	5.48	5.52	5.52	5.56	
Wiki-Talk	HA	3.48	**3.48**	4.04	3.77	**3.67**	≈1x
	GNN-Bet	**3.45**	3.50	**3.61**	**3.66**	3.72	

Lowest execution values are highlighted in bold.

Table 6. KT Ranking Scores Comparison for Closeness Approximations on Real-world and Synthetic Datasets

Dataset	KT Score			
	OC	CD	GNN-Close	Gain
Wiki-Vote	0.916	0.567	**0.937**	+0.021 (+2.2%)
P2p-Gnutella31	0.930	0.926	**0.978**	+0.052 (+5.1%)
Soc-Epinions	**0.945**	0.810	0.917	−0.028 (−3.0%)
Soc-Slashdot	0.928	0.778	**0.955**	+0.027 (+2.9%)
Ego-Gplus	**0.964**	0.909	0.923	+0.041 (−4.4%)
Email-EuAll	0.847	0.494	**0.927**	+0.080 (+9.4%)
Web-Google	**0.920**	0.533	0.699	−0.221 (−31.6%)
Wiki-Talk	0.963	**0.979**	0.978	−0.001 (−0.1%)
Synthetic-ER	0.812 ± 0.04	0.651±0.26	**0.907±0.03**	+0.095 (+11.6%)
Synthetic-SF	0.861 ± 0.04	0.287±0.10	**0.903±0.01**	+0.042 (+4.8%)
Synthetic-GRP	0.527 ± 0.18	−0.403±0.11	**0.979±0.01**	+0.452 (+85.7%)

Higher KT Score between comparison methods(OC and CD) is underlined and best ranking performance among all is highlighted with bold font. Performance gain for GNN-Close is compared with respect to the best KT Score of comparison methods.

Table 7. Time Comparison for Closeness Approximation Methods

Dataset	$\text{Time}_{\text{train}}$(s)	Time(s)				
	GNN-Close	Exact	OC	CD	GNN-Close	Speedup
Wiki-Vote	349.0	0.6	0.03	**0.02**	0.1	≈0.2x
P2p-Gnutella31		42.9	1.3	**0.1**	0.2	≈0.5x
Soc-Epinions		160.7	2.1	**0.3**	0.4	≈0.7x
Soc-Slashdot		152.7	2.0	**0.3**	0.5	≈0.6x
Ego-Gplus		3458.1	4.5	**2.7**	6.2	≈0.4x
Email-EuAll		807.6	3.7	0.8	**0.5**	≈1.6x
Web-Google		38176.7	18.6	10.3	**5.3**	≈1.9x
Wiki-Talk		28792.0	9.7	**2.6**	3.1	≈0.8x
Synthetic-ER	371.4	484.1±452.0	3.0 ±1.1	**0.53±0.36**	0.8±0.6	≈0.6x
Synthetic-SF	235.2	26.4±11.4	0.9 ±0.2	**0.16±0.02**	0.3±0.4	≈0.5x
Synthetic-GRP	383.3	28.8±21.1	1.0 ±0.2	**0.24±0.1**	0.4±0.56	≈0.6x

Lowest execution time among all comparison methods is underlined and bold fonts denotes the fastest method. Speedup gains for GNN-Close are calculated with respect to fastest among comparison methods.

4.5 Static Closeness Approximations

In this section, we evaluate our closeness variant of our model with respect to other closeness centrality approximation methods. For this, we compare to the methods proposed by Okamoto et al. [58] and Cohen et al. [19] and refer to the methods as **OC** and **CD** respectively. KT-Score and execution time for algorithms are used as comparison metrics. For **CD**, the parameter for the number of nodes to be randomly sampled is chosen as 100, as used in this article [19] and we set parameter $\epsilon = 0.001$. The number of nodes to be sampled for **OC** is also set as 100.

For closeness centrality approximations, Table 7 shows the results for real-world and synthetic datasets. Our model outperforms **OC** and **CD** for synthetic datasets with performance improvements up to 85% and 214%, respectively. For the real-world datasets, approximation performance is higher in five out of eight datasets when compared with **OC** and seven out of eight datasets when

compared with **CD**. However, our model's execution time is slightly higher than **CD**, though the absolute differences between values are not significant. Compared to **OC**, our model has significantly faster execution time in most of the comparison datasets. In our model implementation, most of the time is taken by the function to find nodes with no shortest path passing through them. As the code is implemented in `Python`, it is relatively slower. The time taken during GNN model inference is only a few milliseconds. Hence, there is a scope to reduce the approximation time even further. Interestingly, we found that the output of **CD** had a negative KT score for synthetic Gaussian Random Partition graphs.

4.6 Scalability Tests

In previous sections, we have demonstrated the ranking performance of GNN-Bet & GNN-Close and their robustness with different types of graphs. In this section, we study the effect on time to find zero shortest path nodes and the execution time of the model in forward propagation with respect to the increase in the size of the input graph.

As discussed in Section 3.1, there are two possible ways to identify N_{zp} nodes. First, if either in-degree or out-degree of the node is zero, there is no path passing through the node. Time complexity of finding such nodes is $O(V)$. We identify such nodes in first step. Second, if the node's neighbors form cliques such that there is no shortest path going through the node. We identify such nodes by iterating through the remaining nodes $v \in V_{remain}$ from the first step and check if any of the in-degree neighbors N_{in_v} of the node v is not connected directly to all out-degree neighbors N_{out_v}. If such a node in N_{in_v} is found, it means that there is a path going through node v, and the node is not considered to be in N_{zp}. Then, iteration over other in-degree nodes is skipped, and the process is repeated for the next node in V_{remain}. Considering the average number of in-degree or out-degree as $N_{in\_avg} = N_{out\_avg} = N_{avg}$, the time complexity of the operation is $O(VN_{avg}{}^2)$. In case of low density graphs, $N_{avg} << V$ and we do not have to iterate over all N_{in_v}, hence the complexity is almost linear (in number of nodes), approaching $O(VN_{avg})$. However, if the graph is dense and approaching a fully connected graph, then $N_{avg} \simeq V$ and the time complexity of the operation becomes $O(V^3)$. To experimentally verify the time scalability of the method to find zero shortest paths nodes, we generate three sets of Erdos–Renyi random graphs with number of edges to number of nodes ratio set to 2, 4, and 6, respectively. This provides a variation in density of the graphs. We chose ER graphs for this experiment for the convenience of setting the desired number of nodes and edges in the generated graphs easily. The number of nodes are varies from a hundred thousand nodes to a million nodes for each set. Figure 5 shows the plot of execution time to find zero shortest path nodes with respect to change in the graph size. We observe that execution time grows proportional to the number of nodes, and time increments are rather small.

Now, we study the scalability of the inference time of model with an increase in size of the input graphs. Here, we use the betweenness centrality variant of the model, as calculation of betweenness centrality takes a longer time compared to that of closeness centrality. For static betweenness centrality approximation, **BN** is the fastest betweenness approximation method. In order to compare the speed of approximation of our model (GNN-Bet) with **BN**, we test them on 10 synthetic scale-free graphs with the number of nodes ranging from 100,000 to 1,000,000. We set the size of the model to accommodate for all input graphs (i.e., maximum 1 million nodes). Input adjacency matrices of smaller graphs are padded with zeros to match with model size. Hence, the density of all input matrices is proportional to the number of edges in each graph. Therefore, the time taken is approximately proportional to the number of edges and time complexity is $O(|E|)$ (similar to **Graph Convolutional Network (GCN)** [72]). Figure 6 shows the execution time taken

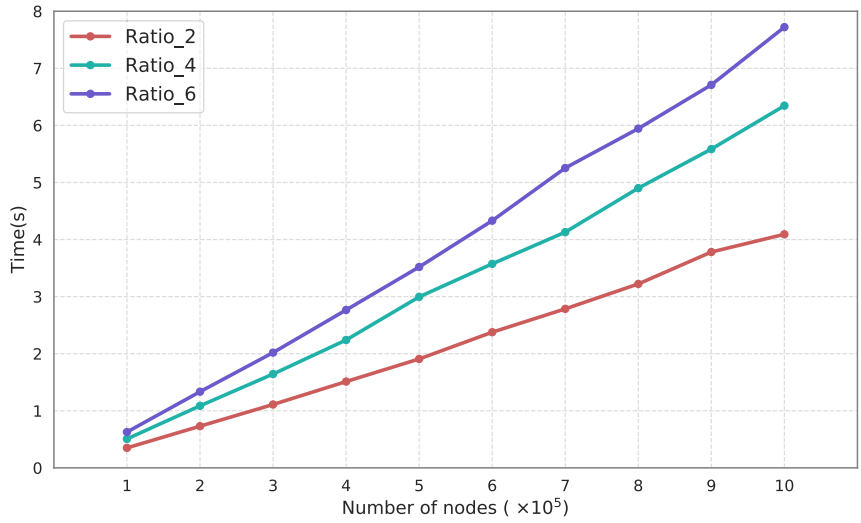

Fig. 5. Depicts the execution time with respect to increase in number of nodes in the input graph. The ratio of number of nodes to number of edges is fixed as 2, 4, and 6, respectively.

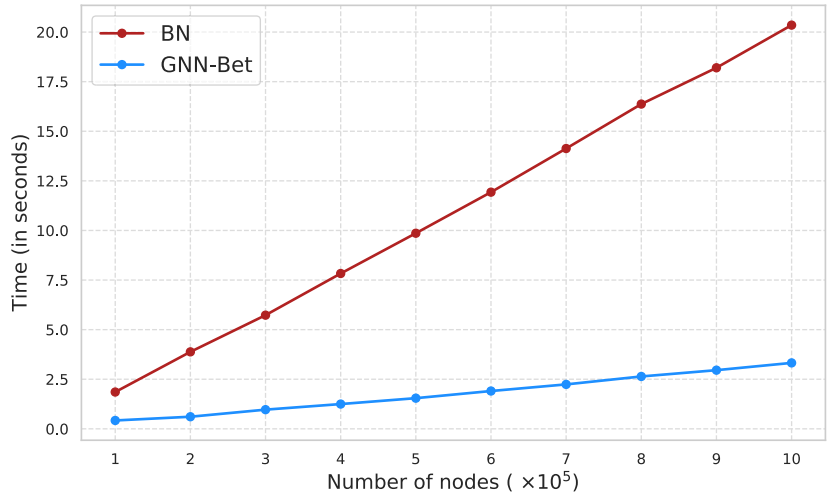

Fig. 6. Time taken by model as number of nodes increases for synthetic scale-free graphs.

by **BN** and betweenness variant of model for all graphs. We observe that in both cases, execution time grows proportional to the size of the graph. Nevertheless, our proposed model GNN-Bet has a lower rate of increment, thus scaling well with the size of graphs and providing faster approximations.

4.7 Ablation Tests

In this section, we conduct a series of experiments to study the effect of hyper-parameters on the model's performance for both variants of our proposed model. As the main parameters of the model are the number of layers and the number of dimensions of the embeddings, we vary both of them and evaluate the performance of the model. We choose to use synthetic datasets for these

experiments, as it provides a better understanding of the model's performance on different types of graph structures. The experiments follow the same methodology as discussed earlier.

— *Varying Number of Layers*: The number of layers in the model affects how much information any given node can accumulate about its neighborhood. With the increase in the number of layers, nodes have access to features of their neighbors located at multi-hop distances away. In this experiment, we vary the number of layers from 1 to 7 for both betweenness and closeness centrality variants. Figures 7 and 9 show Kendall's Tau score for betweenness and closeness variant, respectively for all three types of synthetic graphs. We observe that for a lower number of layers model performs poorly, which is within expectation as the nodes aggregation reach is limited. For both variants GNN-Bet and GNN-Close, we observe that an increase in the number of layers leads to better approximations accuracy.
— *Varying Embedding Dimensions*: While training any neural network, the embedding dimension determines the number of learnable model parameters. An under-parameterized neural network model may not learn very well while the over-parameterized model can overfit the data and have poor generalization ability. In this experiment, we look into the effect of changing embedding dimensions on the performance of our proposed model. We vary the model's embedding dimensions as [2, 4, 8, 12, 16, 20] and evaluate the ranking performance on synthetic graph datasets. Results are plotted in Figures 8 and 10 for betweenness and closeness variant, respectively. With lower dimensions of embedding (2 & 4), we observe lower performance, which is natural due to the limited number of parameters for optimal learning.

4.8 Discussion

The experimental results in Sections 4.3, 4.4, and 4.5 show our GNN-based approach to node ranking in graphs is highly performant and robust. However, like all other neural network based methods, they do require some hyper-parameters that need to be set to get the best out of the model. With ablation experiments, we see that the model's performance is relatively stable with various combinations of parameters. Thus, it does not require extensive hyper-parameter tuning. In Appendix B, we briefly discuss about oversmoothing in GNNs and our observations with respect to the proposed model. For betweenness and closeness approximations, we have set the number of layers in our model to four and seven, respectively for all graph datasets to compare results on a single model. Nevertheless, the number of layers can be chosen based on the structure and diameter of the target graph.

Even though the models are trained on synthetic scale-free graphs, tests on real-world graph datasets show good approximation accuracy. This implies that the proposed model is inductive in nature and has the ability to learn on one set of graphs and predict node ranking scores on others. Furthermore, the model is invariant to the node sequence in the adjacency matrix, providing the additional benefit of having no constraint of ordering the nodes to any specific sequence in graphs.

With almost all deep learning models, even though training time may vary, usually their inference time is really small. Another advantage of neural network models is to take advantage of high-performance GPUs for faster calculations. Our proposed model has the advantage of small training time as well as inference time.

As shown in results, our proposed frameworks are able to perform really well on different types of graphs. However, since the model training is based on ranking loss function where gradients rely on differences in score values. Therefore, it is possible that if the given graph has lots of nodes with similar (or same) centrality values, the model may not be able to distinguish the ranking of nodes and may not perform very well.

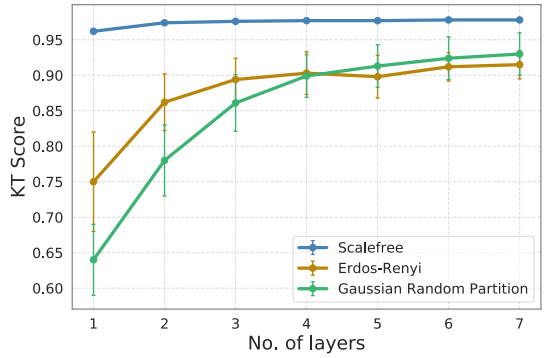

Fig. 7. Change in KT score with respect to number of layers for betweenness variant.

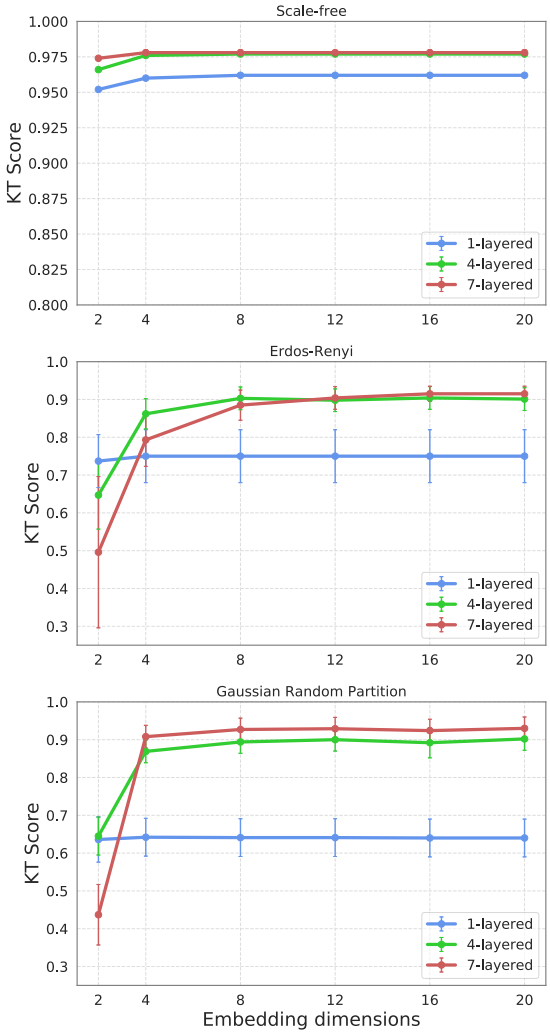

Fig. 8. Change in KT Score with respect to embedding dimensions for betweenness variant.

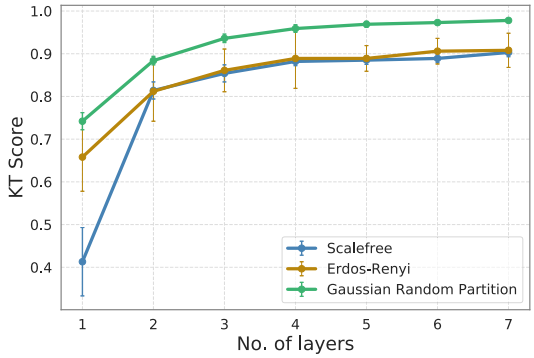

Fig. 9. Change in KT score with respect to number of layers for closeness variant.

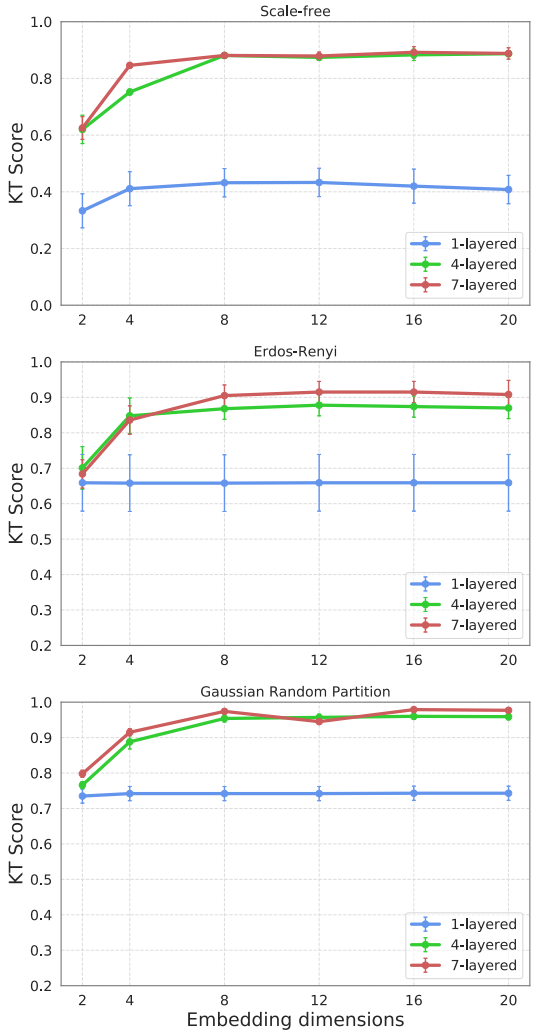

Fig. 10. Change in KT score with respect to embedding dimensions for closeness variant.

5 RELATED WORK

In previous sections, we have shown the viability of neural network based approach for approximating betweenness and closeness centrality of nodes in the graphs. Our proposed model provides a new way to approach the problem and adds to the tool-set in addition to algorithmic-based approaches. In this section, we provide a brief summary of current related works on betweenness and closeness centrality approximation methods and various applications of neural networks on graph-structured data.

5.1 Betweenness Centrality

Intuitively, we have to find all the shortest paths between all pairs of nodes to calculate betweenness. Brandes [10] improves upon the naive algorithm by proposing to generate **directed acyclic graph (DAG)** starting from each node $v \in V$ using BFS and aggregating the contributions of nodes on paths in DAG for all nodes. Calculating exact betweenness centrality value for nodes is computationally expensive and has a time complexity of $\Theta(|V||E|)$. Hence, it is prohibitive for large graphs. Two different approaches have been taken to tackle the problem of high computation time. The first approach is to adapt the betweenness centrality algorithm for distributed computation. Recently, many algorithms [21, 25, 39] have been proposed, which extend the computation from a single machine to a cluster of high performance machines. This approach reduces the computation time significantly for large graphs. However, access to such computation resources is rather limited. The second approach is to rely on approximation methods.

In most of the cases, we do not require an exact calculation of rankings of nodes, so an approximation is sufficient for tasks in hand. Many algorithms have been proposed. Brandes and Pich [11] proposed an algorithm by taking a subset of k starting nodes (pivots) rather than all the nodes. Selecting these pivots uniformly at random and with $k << N$, where N is the number of nodes and betweenness centrality values are extrapolated in the graph. Unfortunately, this method produces large overestimates for unimportant nodes that happen to be near the pivot. Geisberger et al. [32] solve the problem of overestimation by changing the scheme of aggregation of betweenness contributions so that the nodes near pivot do not get extra contributions. To limit the bias, they introduce a scaling function, which gives less importance to contributions from the pivots that are close to the node. Since this algorithm works with random sampling of nodes (pivots), as the number of samples increases, approximation accuracy also increases. Although a higher number of samples also cause higher computation overhead of calculating more shortest paths in the graph.

Riondato et al. [65] take a different approach while providing a theoretical guarantee. The idea is to sample a set of r shortest paths between randomly sampled source–target pairs (s,t). Then, the algorithm computes the approximated betweenness centrality of a given node v as the fraction of sampled paths that v is internal to, by adding $\frac{1}{r}$ to the node's score for each of these paths. Borassi et al. [7] provide a new adaptive sampling technique for sampling shortest paths, which provides a faster computation of betweenness within a given absolute error.

While finalizing this manuscript, we became aware that Fan et al. [28] have also concurrently proposed a GNN-based method for approximation of betweenness centrality. There are several significant differences from our work. First, their method is limited to undirected graphs, while ours is proposed for the more general case of directed graphs. Second, their method is limited to betweenness centrality in static graphs while our model is more flexible and is able to handle both betweenness and closeness centrality in both static and dynamic graphs. Third, their method is proposed for identifying top-N high betweenness nodes while our model is able to make a complete ranking of all the nodes in a graph. Due to these constraints, we do not compare to this work [28] in this article.

In addition to the regular graphs, the concept of betweenness centrality has been applied to directed hypergraphs [1], where relations (also called hyper-edges) are defined from a set of source nodes to a single target node. [47] proposes a new measure called **Participation-based Betweenness Centrality (PBC)** for a special case of directed hypergraphs called B-hypergraphs and provides an exact and approximate method for calculation of PBC.

5.2 Closeness Centrality

To approximate the closeness centrality Eppstein et al. [26] and Okamoto et al. [58] proposed algorithm based on sampling k nodes from the graph. However, these algorithms are not optimal when distance distribution of sampled nodes with other nodes is heavy-tailed. Cohen et al. [19] provided an improved algorithm with a hybrid approach of sampling and pivoting. For directed graphs, there are two variations of closeness centrality based on the direction of paths. One is incoming closeness centrality for paths incoming to the node and outgoing closeness centrality for paths going away from the given node to other nodes. In this article, we have focused on outgoing closeness centrality as it is more meaningful in terms of node importance to spread information.

5.3 Centrality for Dynamic Graphs

In the real world, node interactions and the level of activity of nodes change all the time. Some nodes and edges can disappear and new nodes and edges can appear with time. There is some research on updating betweenness centrality of nodes in dynamic graphs. A naive way is to recompute from scratch, which is however computationally expensive. Many dynamic algorithms [35, 43, 46, 48] have been proposed, which avoid recomputation by maintaining data structure based on original graph structure and update it as the structure changes. These algorithms have faster updates compared to Brandes's exact algorithm. However, they are not scalable as they require a huge amount of memory $\sigma(n^2)$ to store the data structures.

Bergamini et al. [6] proposed a method to approximate the dynamic centrality of nodes with the insertion of edges. This method is based on Riondato et al. [65], although the proposed method does not support the removal of edges. Hayashi et al. [38] propose a new method based on the data structure named *hypergraph sketch*, which is used to represent the shortest paths between nodes sampled randomly. Their proposed method supports both nodes and edges addition/removal operations.

5.4 Graph Neural Networks

Due to an abundance of graph-structured data, many graph-centric neural network models have been proposed. These GNNs have been designed to solve multitude of problems like node classification [45, 68, 69], graph classification [75, 76], link prediction [5, 74], prediction of molecular properties [33], functional analysis of brain [59, 64], synthesize chemical compounds [15, 49], and so on.

All of these GNN frameworks utilize node or edge feature aggregation scheme guided by the structure of the graph [72, 77]. In addition to the simple summation for node features as described in this article, Hamilton et al. [37] have proposed aggregation based on LSTM and max-pooling operation of randomly sampled neighbors. Earlier proposed GNN models were transductive in nature, in other words the models can only be trained and tested on the same graph [22, 22, 45]. However, there have been many emerging works, which are inductive by construction [17, 37, 69]. The models can be trained on one set of graphs and tested on another set of graphs. Recent works are now delving deeper into the working of GNNs from a theoretical perspective, exploring their learning capacity, [20] and expressiveness with respect to graph isomorphism heuristic algorithms [52, 56, 73].

6 CONCLUSION

In this article, we propose a novel approach to solve the betweenness and closeness approximation problem in graphs using GNN. Using the constrained message passing scheme, we train GNN model to output ranking scores for nodes in correlation with betweenness and closeness centrality values. We compare the performance of the model with other state of the art approximation techniques and the experimental results show better performance of our model while taking significantly less time.

For future works, possibilities can be explored to extend the neural network framework to other centrality measures. Graphs based on real-world data often accompany explicit node/edge feature information. Current graph centrality measures lack in utilizing this additional information in ranking nodes as they are solely based on the structure of the graph. However, GNN-based models can easily incorporate any additional data for learning. There is a potential to find new ways to rank nodes based on the graph structure of data and node/edge external features, which explains real-world interactions more consistently.

APPENDICES

A DETAILS OF SYNTHETIC GRAPHS

In this section, we provide generation parameters and other relevant statistics of synthetic graphs used for evaluation of proposed models.

A.1 Synthetic Graph Generation Parameters

We have generated all graphs using a Python based graph library `NetworkX`.[3] Three different graph types: Erdos–Renyi (ER), Scale-Free (SF) and Gaussian Random Partition (GRP) are generated using graph generators provided in `NetworkX`. Table A1 provides the summary of `NetworkX` functions and generation parameters used.

Table A1. `NetworkX` Functions and Parameters used to Generate Synthetic ER, SF, and GRP Graphs

Graph type	NetworkX function	Generation parameters	
ER	fast_gnp_random_graph	Number of nodes	sample(50000,100000)
		p	sample(1,99) $\times 10^{-6}$
SF	scale_free_graph	Number of nodes	sample(50000,100000)
		alpha	sample(40,60) $\times 10^{-2}$
		beta	0.5
		gamma	1 - alpha - gamma
GRP	gaussian_random_partition_graph	Number of nodes	sample(50000,100000)
		s	sample(2000,10000)
		v	sample(2000,10000)
		p_in	sample(2,25) $\times 10^{-5}$
		p_out	sample(2,25) $\times 10^{-5}$

The notation `sample(start,end)` represents uniform sampling function, which randomly samples an integer between `start` and `end`. Parameter names used are the same as provided in `NetworkX` documentation. In the case of ER graphs, when the probability value "p" is low, it is possible for the generated graph to have isolated nodes. We filter out these nodes in the pre-processing step.

[3] https://networkx.github.io/.

Table A2. Statistics of three Types Synthetic Graphs ER, SF and GRP used for Test Datasets is Provided

Graph Type	#Nodes	#Edges	Shortest Paths	
			Avg	Max
ER	88203	126162	19.8± 15.5	91
	69357	175354	11.5±2.6	30
	52369	71773	19.6±17.2	96
	91279	491778	6.9±0.9	14
	73981	173110	12.5±3.1	35
	96390	649893	6.2±0.8	11
	73311	268817	8.6±1.3	19
	92495	652114	6.0±0.7	11
	92937	612060	6.2±0.8	12
	37789	28605	1.3±0.7	11
SF	85683	144937	4.6±2.5	17
	81626	150397	4.4±2.3	17
	99689	207035	3.5±2.1	18
	54547	88096	4.8±2.6	20
	70828	154573	4.0±1.8	13
	67673	137958	3.7±1.9	17
	60137	98727	4.3±2.8	17
	64155	126305	3.8±2.2	16
	83106	136311	4.5±2.7	18
	63653	111007	4.4±2.3	18
GRP	95447	571347	4.3± 2.7	24
	64922	391928	4.4±2.7	23
	50096	188980	2.9±1.8	17
	71002	295411	3.4±2.2	20
	78081	235093	2.5±1.6	16
	55048	294552	3.9±2.5	20
	86739	371866	3.7±2.5	21
	89998	400220	5.5±3.6	34
	68483	365743	4.2±2.7	26
	71422	218240	2.5±1.6	16

A.2 Statistics of Synthetic Graphs

For evaluation of the model, 10 synthetic graphs were generated for each graph type ER, SF, and GRP. Table A2 provides the information of synthetic datasets used as test datasets for the evaluation of our proposed model. All shortest path lengths were measured using Dijkstra's algorithm and for each graph, average and maximum of all shortest path lengths are shown in the table.

A.3 Nodes with Zero Centrality Values

In our proposed model, one of the important steps is to identify the nodes in the graph with no shortest paths going through them and use this information to modify the aggregation scheme in GNN-Bet and GNN-Close. Please note that such nodes in the graph will have zero betweenness

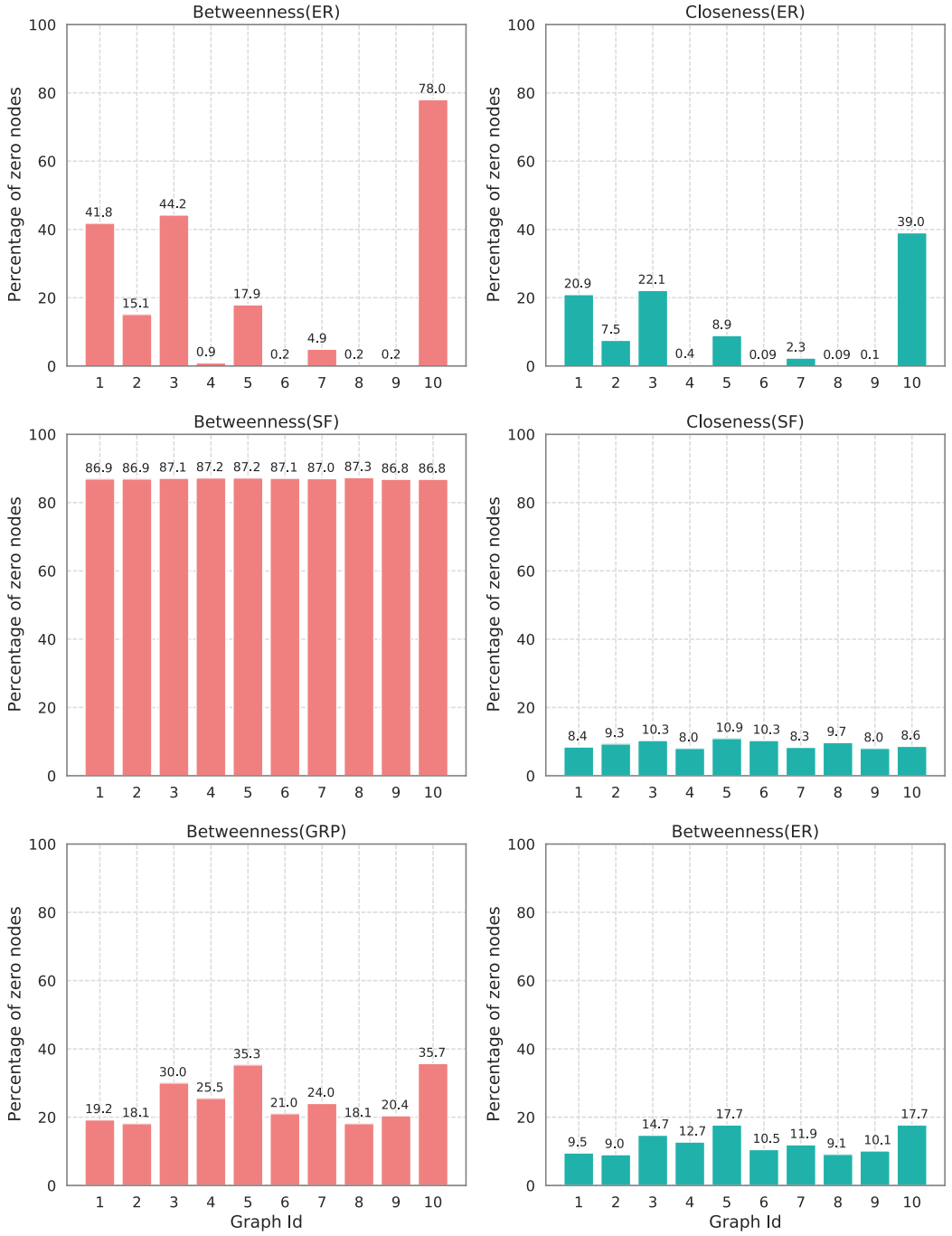

Fig. A1. Figure shows the number of nodes with zero betweenness and closeness centrality for different types of graphs in synthetic evaluation datasets.

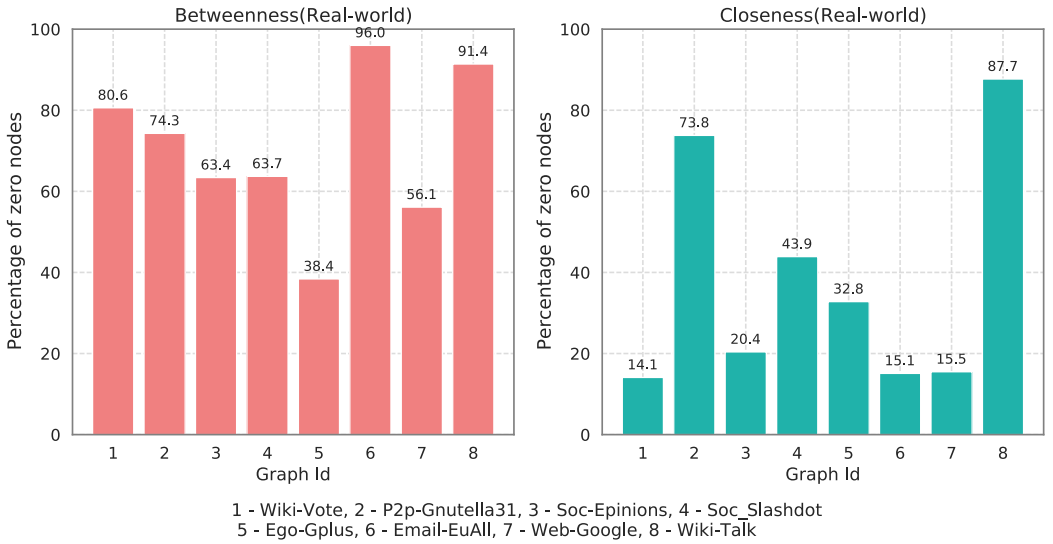

Fig. A2. Figure shows the number of nodes with zero betweenness and closeness centrality for different types of graphs in real-world datasets.

centrality but not necessarily zero closeness centrality. Here, we show the percentage of the nodes with both zero betweenness and closeness to provide the overview of statistics of the test graphs.

Figures A1 and A2 depict the percentage of nodes that have zero betweenness and closeness centrality for synthetic and real-world evaluation datasets. It is interesting to find that the percentage of the nodes for synthetic scale-free graphs is very similar(86%–88%). To investigate further, we generated more synthetic graphs with a wider range of generation parameters than presented in this article. We find that for betweenness centrality, the percentage of such nodes always varied between 86%–89%. With our experimental observations, we conclude that this is an inherent property of graphs generated with `NetworkX` scale-free graph generation function.

B OVERSMOOTHING MEASUREMENT IN GNN-BET & GNN-CLOSE

In the current literature of GNNs, most common tasks are classification or link-prediction-based tasks. In general, the performance of these types of tasks is dependent on the node having similar features to its neighbors. In conventional GNN models, a cumulative aggregation scheme is used by adding self-loops to gain and retain features of the node's neighbors at multiple hops. However, with a higher number of layers in GNN, due to this scheme, nodes tend to have similar features at the final layer, and hence it becomes difficult to learn to classify them with respect to the labels. This problem in general is known as over-smoothing in GNNs.

In our proposed scheme, we aggregate the node features at each layer separately (no self-loops) and map them to an MLP to get layer-specific scores of nodes. The scores are dependent on each node's unique neighborhood at multiple hops. Ranking loss is then calculated based on the combination of the output of the multiple layers. The shortest path lengths can go higher than seven and thus adding more information with a higher number of layers. Nevertheless, the effect diminishes due to fewer number of longer paths.

In order to empirically study the over-smoothing effect in our model, we use a metric similar to one proposed by Chen et al. [16]. In [16], the authors calculate pairwise cosine distance between node embeddings after each layer. They observe that after each successive layer, the average cosine distance reduces thus implying presence of over-smoothing effect. To measure the effect of

Table A3. Table Shows the Average of Pairwise Score Difference of Nodes for Each Layer in Seven-layered GNN-Bet Model

GNN Layers		Average Score Difference (GNN-Bet)		
		ER	SF	GRP
Incoming	Layer-1	0.0025 ±0.003	0.0274 ±0.028	0.0042 ±0.005
	Layer-2	0.0959 ±0.083	0.0520 ±0.043	0.0717 ±0.052
	Layer-3	0.0694 ±0.091	0.0771 ±0.060	0.0456 ±0.046
	Layer-4	0.1124 ±0.108	0.0916 ±0.072	0.1605 ±0.119
	Layer-5	0.0771 ±0.062	0.0312 ±0.028	0.0931 ±0.090
	Layer-6	0.1223 ±0.120	0.0282 ±0.025	0.1355 ±0.101
	Layer-7	0.0959 ±0.101	0.0993 ±0.109	0.0605 ±0.050
Outgoing	Layer-1	0.0025 ±0.004	0.0265 ±0.026	0.0040 ±0.004
	Layer-2	0.0956 ±0.083	0.0647 ±0.050	0.0719 ±0.052
	Layer-3	0.0688 ±0.090	0.0557 ±0.056	0.0451 ±0.046
	Layer-4	0.1120 ±0.108	0.0532 ±0.059	0.1609 ±0.119
	Layer-5	0.0768 ±0.063	0.0297 ±0.035	0.0927 ±0.089
	Layer-6	0.1232 ±0.122	0.02584 ±0.022	0.1344 ±0.100
	Layer-7	0.0970 ±0.103	0.1491 ±0.192	0.0599 ±0.049

Table A4. Table Shows the Average of Pairwise Score Difference of Nodes for Each Layer in Seven-layered GNN-Close Model

#Layer	Average Score Difference (GNN-Close)		
	ER	SF	GRP
Layer-1	0.5202 ±0.508	1.0771 ±0.892	1.9355 ±1.574
Layer-2	1.2834 ±1.052	1.5232 ±1.611	2.5324 ±2.130
Layer-3	3.1149 ±2.643	1.3551 ±1.458	3.9440 ±3.024
Layer-4	2.7950 ±2.216	1.5866 ±1.387	4.5423 ±3.724
Layer-5	0.7851 ±0.651	1.1937 ±1.133	0.9860 ±0.839
Layer-6	0.7561 ±0.886	1.2197 ±1.192	0.8559 ±0.681
Layer-7	0.8555 ±0.716	0.8492 ±0.921	1.4291 ±1.481

over-smoothing on our model's performance, we take similar approach and perform experiments on a seven-layered model. In our proposed model, the output from each layer is a score vector corresponding to the nodes in the graph. Therefore, we replace cosine distance measure with the difference of two score values. As the total number of pairs of nodes can become very large ($\approx N^2$ pairs), we randomly sample 1000 non-zero centrality nodes and calculate pairwise score difference (5×10^5 node pairs) for each layer. In Tables A3 and A4, we show the average score difference values of each layer for betweenness variant and closeness variant, respectively. We observe that there is no obvious consecutive decrease in difference values with respect to the layers. Thus, the model is still able to provide stable performance with seven-layers.

REFERENCES

[1] Giorgio Ausiello and Luigi Laura. 2016. Directed hypergraphs: Introduction and fundamental algorithm: A survey. *Theoretical Computer Science* 658, Part B (2016), 293–306. DOI: https://doi.org/10.1016/j.tcs.2016.03.016

[2] Alex Bavelas. 1950. Communication patterns in task-oriented groups. *Journal of the Acoustical Society of America* 22, 6 (1950), 725–730. DOI: https://doi.org/10.1121/1.1906679

[3] Alex Bavelas. 1948. A mathematical model for group structures. *Applied Anthropology* 7, 3 (1948), 16–30. DOI: https://doi.org/10.17730/humo.7.3.f4033344851gl053

[4] Murray A. Beauchamp. 1965. An improved index of centrality. *Behavioral Science* 10, 2 (1965), 161–163. DOI: https://doi.org/10.1002/bs.3830100205

[5] Rianne van den Berg, Thomas Kipf, and Max Welling. 2017. Graph convolutional matrix completion. arXiv:1706.02263. Retrieved from https://arxiv.org/abs/1706.02263

[6] E. Bergamini, H. Meyerhenke, and C. Staudt. 2015. Approximating betweenness centrality in large evolving networks. In *Proceedings of the 17th Workshop on Algorithm Engineering and Experiments*. Society for Industrial and Applied Mathematics, 133–146. DOI: https://doi.org/10.1137/1.9781611973754.12

[7] Michele Borassi and Emanuele Natale. 2019. KADABRA is an Adaptive Algorithm for Betweenness via Random Approximation. *ACM Journal of Experimental Algorithmics* 24, 1 (2019), 1.2:1–1.2:35. DOI: https://doi.org/10.1145/3284359

[8] Stephen P. Borgatti. 1995. Centrality and AIDS. *Connections* 18, 1 (1995), 112–115.

[9] Stephen P. Borgatti. 2005. Centrality and network flow. *Social Network* 27, 1 (2005), 55–71. DOI: https://doi.org/10.1016/j.socnet.2004.11.008

[10] Ulrik Brandes. 2001. A faster algorithm for betweenness centrality. *The Journal of Mathematical Sociology* 25, 2 (2001), 163–177. DOI: https://doi.org/10.1080/0022250x.2001.9990249

[11] Ulrik Brandes and Christian Pich. 2007. Centrality estimation in large networks. *International Journal of Bifurcation and Chaos in Applied Sciences and Engineering* 17, 7 (2007), 2303–2318. DOI: https://doi.org/10.1142/S0218127407018403

[12] S. Brin and L. Page. 1998. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems* 30, 1–7 (1998), 107–117. Retrieved from http://ilpubs.stanford.edu:8090/361/

[13] Dirk Brockmann and Dirk Helbing. 2013. The hidden geometry of complex, network-driven contagion phenomena. *Science* 342, 6164 (2013), 1337–1342. DOI: https://doi.org/10.1126/science.1245200

[14] Christopher J. C. Burges. 2010. *From RankNet to LambdaRank to LambdaMART: An Overview*. Microsoft Research Technical Report MSR-TR-2010-82. Microsoft, Albuquerque, NM.

[15] Nicola De Cao and Thomas Kipf. 2018. MolGAN: An implicit generative model for small molecular graphs. In *ICML Workshop, Theoretical Foundations and Applications of Deep Generative Models*. Retrieved from https://arxiv.org/abs/1805.11973

[16] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. 2019. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. *Proceedings of AAAI conference on Artificial Intelligence* 34 (2019), 3438–3445. Retrieved from http://arxiv.org/abs/1909.03211

[17] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *Proceedings of the International Conference on Learning Representations*. Retrieved from https://openreview.net/forum?id=rytstxWAW&noteId=ByU9EpGSf.

[18] Wei Chen, Tie-yan Liu, Yanyan Lan, Zhi-ming Ma, and Hang Li. 2009. Ranking measures and loss functions in learning to rank. In *Proceedings of theAdvances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta (Eds.). Curran Associates, Inc., 315–323. Retrieved from http://papers.nips.cc/paper/3708-ranking-measures-and-loss-functions-in-learning-to-rank.pdf.

[19] Edith Cohen, Daniel Delling, Thomas Pajor, and Renato F. Werneck. 2014. Computing classic closeness centrality, at scale. In *Proceedings of the 2nd ACM Conference on Online Social Networks* . ACM, 37–50. DOI: https://doi.org/10.1145/2660460.2660465

[20] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. 2020. Principal neighbourhood aggregation for graph nets. In *Advances in Neural Processing Systems 33 (NeurIPS'20)*.

[21] Pierluigi Crescenzi, Pierre Fraigniaud, and Ami Paz. 2020. Simple and fast distributed computation of betweenness centrality. In *IEEE Conference on Computer Communications*. DOI: https://doi.org/10.1109/INFOCOM41043.2020.9155354

[22] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. arxiv:1606.09375 Retrieved from http://arxiv.org/abs/1606.09375

[23] Ying Ding, Erjia Yan, Arthur Frazho, and James Caverlee. 2010. PageRank for ranking authors in co-citation networks. *Journal of the American Society for Information Science and Technology* 60, 11 (2010), 2229–2243. DOI: https://doi.org/10.1002/asi.v60:11

[24] David K. Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P. Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*. 2224–2232. Reterieved from http://papers.nips.cc/paper/5954-convolutional-networks-on-graphs-for-learning-molecular-fingerprints.pdf.

[25] Nick Edmonds, Torsten Hoefler, and Andrew Lumsdaine. 2010. A space-efficient parallel algorithm for computing betweenness centrality in distributed memory. In *Proceedings of the 2010 International Conference on High Performance Computing*. 1–10. DOI: https://doi.org/10.1109/HIPC.2010.5713180.

[26] David Eppstein and Joseph Wang. 2001. Fast approximation of centrality. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 228–229. Retrieved from http://dl.acm.org/citation.cfm?id=365411.365449.

[27] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. 2019. A fair comparison of graph neural networks for graph classification. arXiv:1912.09893.

[28] Changjun Fan, Li Zeng, Yuhui Ding, Muhao Chen, Yizhou Sun, and Zhong Liu. 2019. Learning to identify high betweenness centrality nodes from scratch: A novel graph neural network approach. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. ACM, 559–568. DOI: https://doi.org/10.1145/3357384.3357979

[29] Massimo Franceschet. 2011. PageRank: Standing on the shoulders of giants. *Communications of the ACM* 54, 6 (2011), 92–101. DOI: https://doi.org/10.1145/1953122.1953146

[30] Linton C. Freeman. 1978. Centrality in social networks conceptual clarification. *Social Networks* 1, 3 (1978), 215–239. DOI: https://doi.org/10.1016/0378-8733(78)90021

[31] Linton C. Freeman. 1977. A set of measures of centrality based on betweenness. *Sociometry* 40, 1 (1977), 35–41. DOI: https://doi.org/10.2307/3033543

[32] Robert Geisberger, Peter Sanders, and Dominik Schultes. 2008. Better approximation of betweenness centrality. In *Proceedings of the Meeting on Algorithm Engineering & Expermiments*. 90–100. Retrieved from http://dl.acm.org/citation.cfm?id=2791204.2791213.

[33] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. 1263–1272. Retrieved from http://dl.acm.org/citation.cfm?id=3305381.3305512.

[34] David F. Gleich. 2015. PageRank beyond the web. *Society for Industrial and Applied Mathematics* 57, 3 (2015), 321–363. DOI: https://doi.org/10.1137/140976649

[35] O. Green, R. McColl, and D. A. Bader. 2013. A fast algorithm for streaming betweenness centrality. In *Proceedings of the 2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing*. 11–20. DOI: https://doi.org/10.1109/SocialCom-PASSAT.2012.37

[36] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference*. 11–15.

[37] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), ACM, 1024–1034. Retrieved from http://papers.nips.cc/paper/6703-inductive-representation-learning-on-large-graphs.pdf.

[38] Takanori Hayashi, Takuya Akiba, and Yuichi Yoshida. 2015. Fully dynamic betweenness centrality maintenance on massive networks. 9 (2015), 48–59. DOI: https://doi.org/10.14778/2850578.2850580

[39] Loc Hoang, Matteo Pontecorvi, Roshan Dathathri, Gurbinder Gill, Bozhi You, Keshav Pingali, and Vijaya Ramachandran. 2019. A round-efficient distributed betweenness centrality algorithm. In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*. Association for Computing Machinery, 272–286. DOI: https://doi.org/10.1145/3293883.3295729

[40] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. 2019. Pre-training graph neural networks. In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*. 272-286

[41] S. Jin, Z. Huang, Y. Chen, D. Chavarría-Miranda, J. Feo, and P. C. Wong. 2010. A novel application of parallel betweenness centrality to power grid contingency analysis. In *Proceedings of the 2010 IEEE International Symposium on Parallel Distributed Processing*. 1–7. DOI: https://doi.org/10.1109/IPDPS.2010.5470400

[42] Maliackal Poulo Joy, Amy Brock, Donald E. Ingber, and Sui Huang. 2005. High-betweenness proteins in the yeast protein interaction network. *Journal of Biomedicine and Biotechnology* 2005, 2 (2005), 96–103. DOI: https://doi.org/10.1155/JBB.2005.96

[43] M. Kas, M. Wachs, K. M. Carley, and L. R. Carley. 2013. Incremental algorithm for updating betweenness centrality in dynamically growing networks. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining* 33–40. DOI: https://doi.org/10.1109/ASONAM.2013.6785684

[44] M. G. Kendall. 1938. A new measure of rank correlation. *Biometrika* 30, 1 (1938), 81–93. DOI: https://doi.org/10.1093/biomet/30.1-2.81

[45] Thomas N. Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representation (ICLR'17)*.

[46] N. Kourtellis, G. De Francisci Morales, and F. Bonchi. 2016. Scalable online betweenness centrality in evolving graphs. In *Proceedings of the 2016 IEEE 32nd International Conference on Data Engineering*. 1580–1581. DOI: https://doi.org/10.1109/ICDE.2016.7498421

[47] Kwang Hee Lee and Myoung Ho Kim. 2020. Linearization of dependency and sampling for participation-based betweenness centrality in very large B-hypergraphs. *ACM Transactions on Knowledge Discovery from Data* 14, 3 (2020), 25:1–25:41. DOI: https://doi.org/10.1145/3375399

[48] Min-Joong Lee, Jungmin Lee, Jaimie Yejean Park, Ryan H. Choi, and Chin-Wan Chung. 2012. QUBE: A quick algorithm for updating betweenness centrality. In *Proceedings of the 21st international conference on World Wide Web*. 351–360. DOI: https://doi.org/10.1145/2187836.2187884

[49] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter W. Battaglia. 2018. Learning deep generative models of graphs. In *International Conference on Machine Learning*.

[50] Hao Liao, Manuel Sebastian Mariani, Matúš Medo, Yi-Cheng Zhang, and Ming-Yang Zhou. 2017. Ranking in evolving complex networks. *Physics Reports* 689 (2017), 1–54. DOI: https://doi.org/10.1016/j.physrep.2017.05.001

[51] Andreas Loukas. 2019. What graph neural networks cannot learn: Depth vs width. In *Proceedings of the International Conference on Learning Representations*.

[52] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. 2020. Provably powerful graph networks. In *Advances in Neural Processing Systems 32 (NeurIPS'19)*.

[53] Sunil K. Maurya, Xin Liu, and Tsuyoshi Murata. 2019. Fast approximations of betweenness centrality using graph neural networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. ACM, 2149–2152.

[54] Matus Medo. 2013. Network-based information filtering algorithms: Ranking and recommendation. In *Dynamics on and of Complex Networks, Volume 2: Applications to Time-Varying Dynamical Systems*, Animesh Mukherjee, Monojit Choudhury, Fernando Peruani, Niloy Ganguly, and Bivas Mitra (Eds.). Springer, New York, 315–334. DOI: https://doi.org/10.1007/978-1-4614-6729-8_16

[55] Peter Mika. 2004. Social networks and the semantic web. In *Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence*. IEEE Computer Society, 285–291. Retrieved from http://dl.acm.org/citation.cfm?id=1025132.1026332.

[56] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2018. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*. DOI: https://doi.org/10.1609/aaai.v33i01.33014602

[57] M. E. J. Newman. 2003. A measure of betweenness centrality based on random walks. *Social Networks* 27, 1 (2003), 39–54. DOI: https://doi.org/10.1016/j.socnet.2004.11.009

[58] Kazuya Okamoto, Wei Chen, and Xiang-Yang Li. 2008. Ranking of closeness centrality for large-scale social networks. In *Frontiers in Algorithmics*, Franco P. Preparata, Xiaodong Wu, and Jianping Yin (Eds.). Lecture Notes in Computer Science, Springer, Berlin, 186–195.

[59] Sarah Parisot, Sofia Ira Ktena, Enzo Ferrante, Matthew Lee, Ricardo Guerrero Moreno, Ben Glocker, and Daniel Rueckert. 2017. Spectral graph convolutions for population-based disease prediction. In *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention*, Maxime Descoteaux, Lena Maier-Hein, Alfred Franz, Pierre Jannin, D. Louis Collins, and Simon Duchesne (Eds.). Lecture Notes in Computer Science, Springer International Publishing, 177–185. DOI: https://doi.org/10.1007/978-3-319-66179-7_21

[60] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *Proceedings ofthe31st Conference on Neural Information Processing Systems*.

[61] Sen Pei, Lev Muchnik, José S. Andrade, Jr, Zhiming Zheng, and Hernán A. Makse. 2014. Searching for superspreaders of information in real-world social media. *Scientific Reports* 4, Article 5547 (2014). DOI: https://doi.org/10.1038/srep05547

[62] M. C. Pham and R. Klamma. 2010. The structure of the computer science knowledge network. In *Proceedings of the 2010 International Conference on Advances in Social Networks Analysis and Mining*. 17–24. DOI: https://doi.org/10.1109/ASONAM.2010.58

[63] Rami Puzis, Yaniv Altshuler, Yuval Elovici, Shlomo Bekhor, Yoram Shiftan, and Alex Pentland. 2013. Augmented betweenness centrality for environmentally aware traffic monitoring in transportation networks. *Journal of Intelligent Transportation Systems* 17, 1 (2013), 91–105. DOI: https://doi.org/10.1080/15472450.2012.716663

[64] Zarina Rakhimberdina and Tsuyoshi Murata. 2020. Linear graph convolutional model for diagnosing brain disorders. In *Complex Networks and Their Applications VIII*, Hocine Cherifi, Sabrina Gaito, José Fernendo Mendes, Esteban Moro, and Luis Mateus Rocha (Eds.). Studies in Computational Intelligence, Springer International Publishing, 815–826. DOI: https://doi.org/10.1007/978-3-030-36683-4_65

[65] Matteo Riondato and Evgenios M. Kornaropoulos. 2016. Fast approximation of betweenness centrality through sampling. *Data Mining and Knowledge Discovery* 30, 2 (2016), 438–475. DOI: https://doi.org/10.1007/s10618-015-0423-0

[66] Gert Sabidussi. 1966. The centrality index of a graph. *Psychometrika* 31, 4 (1966), 581–603. DOI: https://doi.org/10.1007/BF02289527

[67] Christian L. Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. 2016. NetworKit: A tool suite for large-scale complex network analysis. *Network Science* 4, 4 (2016), 508–530. DOI: https://doi.org/10.1017/nws.2016.20

[68] Petar Veliĉković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks. In *International Conference on Learning Representation (ICLR'18)*.

[69] Petar Veliĉković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. 2018. Deep graph infomax. In *International Conference on Learning Representation (ICLR'19)*.

[70] Zhen Wang, Chris T. Bauch, Samit Bhattacharyya, Alberto d'Onofrio, Piero Manfredi, Matjaz Perc, Nicola Perra, Marcel Salathé, and Dawei Zhao. 2016. Statistical physics of vaccination. Physics Reports 664 (2016), 1–113. DOI: https://doi.org/10.1016/j.physrep.2016.10.006

[71] Felix Wu, Amauri H. Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. 2019. Simplifying graph convolutional networks. In *Proceedings of the 36th International Conference on Machine Learning*. 6861–6871.

[72] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* 32, 1 (2021), 4–24.

[73] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? In *International Conference on Learning Representation (ICLR'19)*.

[74] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. DOI: https://doi.org/10.1145/3219819.3219890

[75] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. Curran Associates Inc. 4805–4815. Retrieved from http://dl.acm.org/citation.cfm?id=3327345.3327389.

[76] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*. 4434–4445.

[77] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2020. Graph neural networks: A review of methods and applications. *AI Open* 1 (2020), 57–81. https://www.sciencedirect.com/science/article/pii/S2666651021000012.

Received January 2020; revised October 2020; accepted December 2020