Accelerating Large Network Data Analytics Using Graph Neural Networks (GNNs)

Nidhin Harilal
Unversity of Colorado, Boulder
Boulder, CO, USA
nidhin.harilal@colorado.edu

Abstract

Analysis of large network data analytics has as high theoretical and practical significance. Centrality measures are one of the most common schemes that is employed currently for large-scale network analysis. However, existing deterministic algorithms for computing this for individual node takes a lot of time and are computationally super-expensive. This problem elevates when the analysis is done for very large graphs. Getting an approximate estimates of these centrality measures with much lower time would be of great significance in network analytics. In this project, I aim to do the same using Graph Neural Networks (GNNs). In this project, I aim to do the same using Graph Neural Networks (GNNs). I train a GNN based model on a synthetically generated dataset consisting of variety of networks to approximate these centrality values. The tests on complex real networks datasets shows that GNN gives promising results in predicting these values along with a huge speed-up of nearly 80-times when compared to the currently used deterministic algorithms when computed over a CPU. Furthermore, the gap in the speedup of GNN-based approach over conventional methods seems to be more massive when a GPU is harnessed.

Keywords: Network Analytics, Large-scale networks, Graph Neural Networks, Network centrality

1 Introduction

From determining friend suggestions in social media platforms to predicting possible congestions in communication/ internet protocols, Large network data analytics have wide range of applications in our day-to-day lives.

The problem of identifying the important nodes and ranking it in a graph is very significant in the field of network analysis. One of the key challenges in large-network data analytics is finding influential nodes. Here, influential nodes refers to certain nodes that can affect network structure and function to a greater extent than other nodes in the network. One of the ways in which influential nodes can be identified is ranking is based on its ability to control the spread of information in the graph [6]. Other measures include the degree of connectivity of nodes [8] and the closeness of the nodes in a network [8] etc.

For the scope of this project I will be focusing on Betweenness, Closeness and Degree centrality.

- **Degree centrality:** It refers to the number of connections that a node v has with the other nodes.

$$C_D(V) = deg(v)$$

- **Closeness centrality:** It refers to the inverse of the average length of shortest path between the node v & all others.

$$C(x) = \frac{N}{\Sigma_y} d(y, x)$$

- **Betweenness Centrality:** It refers to the number of times the node v acts as abridge in the shortest path between any of the two nodes.

$$C_D(x) = \Sigma_{s \neq v \neq t \neq \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

The traditional method of computing these centrality values takes a lot of time and are computationally expensive. This situation worsens when the network size increases. With this project, I aim to tackle this issue and develop super-efficient method for approximate inference on large network data using Graph Neural Networks (GNNs) [5].

2 Background

2.1 Large Network Data analytics

2.2 Graph Neural Networks (GNNs)

GNN is a type of Neural Network which operates directly on graphs. Originally, Graph Neural Networks were introduced back in 2005 but it regained attention in the past few years. Studies [3, 10] have shown that Graph Neural networks have a great representation power and have been successful in establishing relationship between large scale graph and its features. Although there exists many approaches towards calculating approximate centrality measures, but complex and non-linear dependence among these values motivates the usage of Graph Neural Networks (GNN).

2.2.1 Convolutional GNNs:

Proposed originally by Niepert et al. [7], Convlutional GNNs have been found to be quite powerful in learning quality graph mappings, thereby, finding its application in many complex real-world networks. The idea is reminiscent of standard convlution that is in computer vision. In essence, it follows the idea of aggregate and process the information of an element's neighbors in order to update the particular element's value as shown in Figure 1.

The correspondence between graph and images is that - in graphs, the element is a node, and in images, the element is a pixel. However, the number of neighboring nodes in a graph can be variable, unlike in an image where each pixel has a set number of neighboring elements. By stacking message through convolution in GNN layers, a node can eventually incorporate information from across the entire graph.

3 Related Work

While the capability of GNNs to learn complex and non-linear dependence among these values motivates its usage for inferring centrality measures. There exists some attempts which have tried to look into a similar problem space.

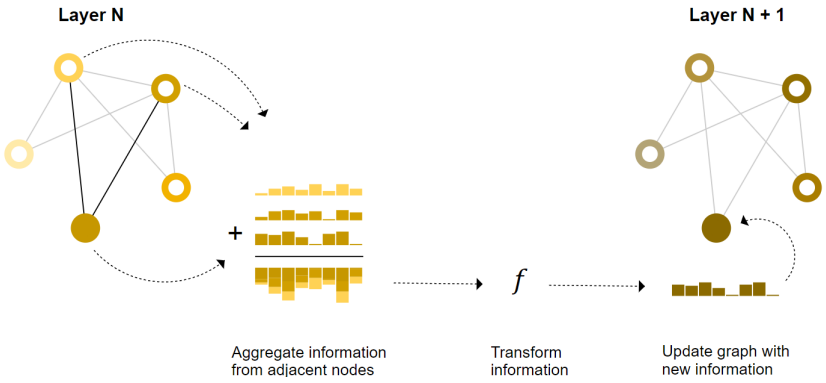

Figure 1: Convolution idea for graphs [9]

Specifically, Maurya et al. [5] were the first to follow a the idea of using Graph Neural networks to infer approximate *Betweenness* centrality. They proposed a contstrained message passing framework using GNNs to accomplish this. Even though the framework was a state-of-the-art at the time, it showed just acceptable results. Not only the analysis on other centrality measures like Closeness was left, the article was from 3 years ago, which leaves a large scope for study of more recent and capable approaches on GNNs [4, 1]

4 Dataset

GNNs like most other data-driven parametric models require huge amounts of data to work properly. Additionally, training a GNN for predicting centrality values will require dataset consisting of hundreds of networks with varying size, and possess diverse properties. Given that it's quite difficult to gather real-world networks at that scale, I have generated a synthetic dataset consisting of 450 graphs. The dataset comprises networks of diverse types including Erdős–Rényi(ER), Barabási–Albert(BA) and Gaussian Random Partition graphs, each sharing atmost one-third of the total. The definition of each of these types areas follows:

- **Erdős–Rényi(ER):** Random graphs in which edges are set between nodes with equal probabilities.
- **Barabási–Albert (BA):** Random graphs having preferential attachments having a heavy tailored degree distribution.
- **Gaussian Random Partition:** Random graphs created using k-partitions each with a size drawn from a normal distribution.

Each of the graphs have number of nodes which have been sampled randomly from 2,000 to 15,000. These different graph types and the statistics of nodes and edges ensures diversity in the dataset. These graphs are built in python using '*networkx*', '*networkit*' and '*SNAP*' scientific libraries.

For testing and evaluation, I've collected some real-world networks data from *Stanford Network Analysis Project* (SNAP) repository[1]. I've used several *Python* packages including *NetworkX, NetworkIt*, etc., to compute each of their centrality measures. The collected validation data includes survey from participants in Facebook, Theory collaborations in High Energy Physics and citation network in General Relativity and Quantum cosmology. The table below shows the network statistics of these datasets.

[1]https://snap.stanford.edu/data

Graph	No of Nodes	No of Edges
Facebook Survey	4039	88234
Physics Theory	9877	25998
General Relativity	5242	14496

Table 1: Graphs for test data

5 Methodology

The recent advancements in GNNs have made them capable to extract and learn complex relations in High-dimensional graphs as shown by [3, 8]. These GNNs utilizes the idea of message passing framework. This involves the aggregation of node features at different levels from it's neighbours. I've followed this similar idea of accumulating the feature vectors at different levels of GNN layers that are attached consecutively to form a output vector as shown in Figure 2.

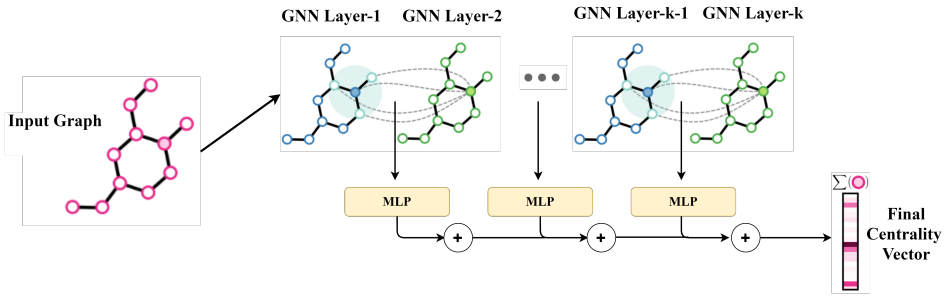

Figure 2: Model Diagram

5.1 Loss Criterion

Centrality values of a network are ordinal in nature making them ordinal variables. In machine learning ordinal variables refers to those which have a relative ordering between them that is of high significance. Therefore, standard regression criterion such as *Mean-Square Error (MSE)* would be not be good objective function.

Instead, I plan to use one of the ranking loss, i.e *Margin-Ranking loss (MR Loss)* [2] as the objective function for training.

$$\text{MR Loss}(x, y) = max(0, -y \times (S_i^{pred} - S_i^{actual})$$

$$A = \begin{cases} 1, & \text{if } S_i^{pred} \geq S_i^{actual} \text{ in rank} \\ -1, & \text{otherwise} \end{cases} \tag{1}$$

Here, S_i corresponds to the rank of node i based on the selected centrality measure. In PyTorch, this can be found here.

5.2 Experimental Settings

I have used a total of 6-stacked GNN layers and each of the MLPs (Multi-Layer Perceptrons) had a depth of 3 and each having 32 neurons respectively. Each GNN layer has a dropout-rate of 0.30. The model was trained for a total of 20 epochs, out of which the configuration with best test accuracy is considered for validation. The model configuration are same for all the three cases: Degree, Betweenness, Closeness centrality except for the dropout rates which is higher for the case of Closeness with 0.5. The model is built completely on PyTorch framework. For training NVIDIA-RTX 2080-Ti (12 GB) GPU along with an Intel Xeon Silver (x12 cores) processor was harnessed.

6 Results

As a part of evaluation, I have considered both the accuracy in predicting the centrality values and the speed-up in time between our approach and the deterministic algorithm.

6.1 Prediction Accuracy

In network analysis, since we are concerned about the influential nodes when we compute centrality measures, therefore, I use the hit rates as a metric for performance of the model. The measure of $k-$hit rate here refers to the percentage of the nodes that the model retrieved/ predicted corectly in the top k percent influential nodes having the highest centrality values.

Centrality Measure	Top 5% Hits	Top 10% Hits	Top 20% Hits
Degree	89.05 ± 1.57	93.14 ± 1.20	99.11 ± 0.68
Betweenness	83.20 ± 1.64	90.58 ± 1.47	98.25± 2.40
Closeness	78.92 ± 4.16	85.07 ± 3.84	89.25±4.62

Table 2: Obtained Hit Rates along with its standard deviation on the test data.

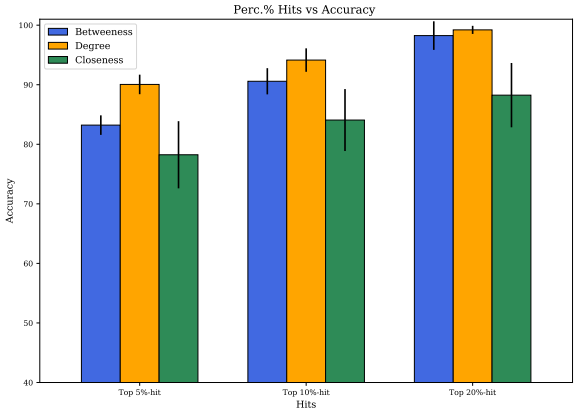

Figure 3: Plot of the obtained Hits vs Accuracy on the test data.

Table 2 and Figure 3 shows the obtained percent hit rates on the test data. The error bars shown in Figure 3 represent the 95 percent confidence. It has been computed using the marhin of error described as:

$$CI = \overline{x} \pm z\frac{s}{\sqrt{n}}$$

where $\overline{x}$ is the mean of obtained values, z is 0.95 (Confidence), s is the standard-deviation and n is the number of observations.

The model is able to predict the degree and betweenness centrality measure with a good accuracy of more than 90 percent on top-10% hits. Closeness centrality on the other hand have a little lower but fair accuracy of more than 85% in top-10% hits in the actual network.

6.2 Time Comparison

Inference in Graph Neural Network (GNN) have a time complexity of order $O(E)$, where E refers to the number of edges in the graph. Solving both *Betweenness* and *Closeness* centrality measures deterministic ally take time in the order ON^3, where N is the number of nodes of the network. Therefore, GNN-based model should theoretically take much lower time than the deterministic algorithm. *Degree* centrality on the other hand, has time complexity of order $O(E)$, which is same as that of GNN-based model. However, due to higher number of computations in the GNN-based approach, it should take a longer time.

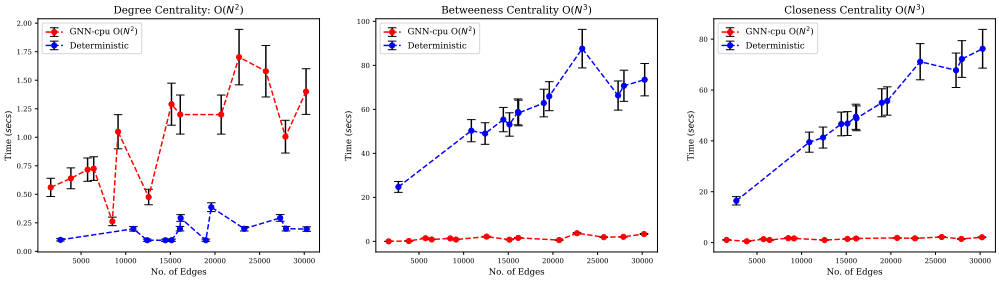

Figure 4: Comparison of the time taken in CPU to compute centrality measures, when the number of edges are increased in a network. The black bars show the std-deviation in time taken.

Figure 4 shows the comparison of the time taken by both GNN-based model and the deterministic algorithm to compute each of the centrality values as the number of edges are increased in a network. For a fair comparison in terms of hardware, the comparison shown in Figure 4 is performed entirely on CPU. These results match our theoritical expectation. Note here the speed-up that GNN-based model offers when *Betweenness* and *Closeness* centrality are computed. For around $30,000$ edges, the GNN-based model offers almost 80 times speed-up which is tremendous considering the fact that it gives more around 90% accuracy on top-10% nodes in the network.

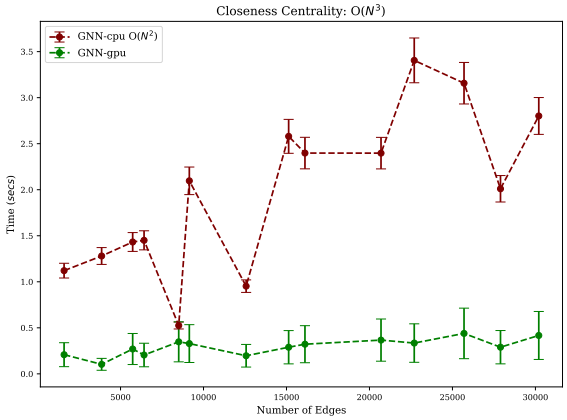

Figure 5: Comparison of time taken for computing *Closeness* centrality on GPU (NVIDIA-2080Ti) and CPU (Intel Xeon Silver).

Still, One of the other main feature of using GNNs are that frameworks such PyTorch and Tensorflow enables its computation over GPUs which should be theoretically much faster as compared to the computations over CPU. Figure 5 shows the time comparison while computing on GPU and CPU. The plot shows that we can obtain further speed-ups when the computations are done over GPU.

7 Conclusion

The huge time speed-up that is provided by the Graph Neural Network (GNN) based model in computing centrality values justifies the advantage of exploring this field. GNNs are a very active line of research and in this project. I have used a simpler version of GNN-architecture which still performed fairly-well. This further motivates applying the recently proposed and far advanced GNN-based models for computing centrality values on the network. This whole approach can also be extended further to other Network measures like Page-Rank and clustering coefficients etc, which again are computationally very expensive to compute. On

the limitation side, GNN still cannot take graphs which have higher nodes as compared to its model-size. Therefore, work still needs to be done for generality of GNN for larger networks.

All the codes and the data used are present in public domain[2]

References

[1] Sami Abu-El-Haija, Amol Kapoor, Bryan Perozzi, and Joonseok Lee. N-gcn: Multi-scale graph convolution for semi-supervised node classification. In *uncertainty in artificial intelligence*, pages 841–851. PMLR, 2020.

[2] Wei Chen, Tie-Yan Liu, Yanyan Lan, Zhi-Ming Ma, and Hang Li. Ranking measures and loss functions in learning to rank. *Advances in Neural Information Processing Systems*, 22, 2009.

[3] Nima Dehmamy, Albert-László Barabási, and Rose Yu. Understanding the representation power of graph neural networks in learning graph topology. *Advances in Neural Information Processing Systems*, 32, 2019.

[4] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 639–648, 2020.

[5] Sunil Kumar Maurya, Xin Liu, and Tsuyoshi Murata. Fast approximations of betweenness centrality with graph neural networks. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 2149–2152, 2019.

[6] Mark EJ Newman. A measure of betweenness centrality based on random walks. *Social networks*, 27(1):39–54, 2005.

[7] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023. PMLR, 2016.

[8] Kazuya Okamoto, Wei Chen, and Xiang-Yang Li. Ranking of closeness centrality for large-scale social networks. In *International workshop on frontiers in algorithmics*, pages 186–195. Springer, 2008.

[9] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B. Wiltschko. A gentle introduction to graph neural networks. *Distill*, 2021. doi: 10.23915/distill.00033. https://distill.pub/2021/gnn-intro.

[10] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[2] `https://github.com/cryptonymous9/GNN-for-Network-Centrality`