[CS552]: Network Science

Sankalp Acharya, Dakshraj Sadashiv Kashid

May 5, 2023

1 Problem Statement 1: Designing a Synthetic Social Network Generator

Social network generators use algorithms and statistical models to create networks that resemble real social networks, with nodes representing individuals and edges representing social relationships such as friendships, collaborations, or information flows.

Our aim is to design a synthetic social network generator which performs better on certain metrics or measures than the existing generators.

Applications

- *Testing network analysis methods:* Synthetic social networks can be used to test and validate network analysis methods, such as centrality measures, clustering algorithms, and community detection techniques. By generating networks with known properties, researchers can compare the performance of different methods and evaluate their strengths and limitations.

- *Evaluating social phenomena:* Synthetic social networks can be used to study the impact of network structures on social phenomena such as information diffusion, opinion formation, and social influence. By generating networks with different structural characteristics, researchers can analyze how different network properties affect social behavior and outcomes.

- *Generating training datasets:* Synthetic social networks can be used to generate datasets for training and testing machine learning algorithms, such as those used for link prediction, node classification, and community detection. These datasets can be generated to mimic real-world networks with specific properties, allowing researchers to develop and test algorithms under different network structures.

- *Simulating social processes:* Synthetic social networks can be used to develop simulations of social processes in virtual environments. By generating networks with specific properties and adding social dynamics and rules, researchers can simulate social processes such as the spread of rumors or the emergence of social norms.

- *Generating controlled experiments:* Synthetic social networks can be used to design controlled experiments to test hypotheses about social behavior. By generating networks with specific properties, researchers can manipulate the network structure to test the effects of different variables on social behavior and outcomes.Testing network analysis methods: Synthetic social networks can be used to test and validate network analysis methods, such as centrality measures, clustering algorithms, and community detection techniques. By generating networks with known properties, researchers can compare the performance of different methods and evaluate their strengths and limitations.

1.1 Related work

Classic Models

1. *Barabasi-Albert Model:* The Barabasi-Albert model generates a network by adding nodes one at a time and connecting each new node to existing nodes based on a preferential attachment rule, creating a rich-get-richer dynamic that leads to the power-law degree distribution. The Barabasi-Albert model has several important properties. First, it generates networks that have a small-world property, meaning that the average shortest path between any two nodes is relatively short. Second, it generates networks that have a high clustering coefficient, meaning that nodes tend to form tightly connected clusters. Third, it generates networks with a high degree of heterogeneity, meaning that some nodes have many connections while most have few.

2. *Erdos-Renyi Model:* The Erdős–Rényi model generates graphs that are characterized by their average degree, and the distribution of degrees follows a Poisson distribution. The Erdős–Rényi model has several important properties. First, it generates networks that have a low clustering coefficient, meaning that nodes are not strongly connected to their neighbors. Second, it generates networks that have a low diameter, meaning that the average shortest path between any two nodes is relatively small.

3. *Watts-Strogatz Model:* The Watts-Strogatz model generates a network by starting with a ring of n nodes, where each node is connected to its k nearest neighbors (k is typically a small integer). Then, with a certain probability p, each edge in the ring is rewired to a new random node. This rewiring process creates shortcuts in the network, allowing for long-range connections that increase the network's clustering coefficient and decrease its average path length. The Watts-Strogatz model generates networks that have a high clustering coefficient and a low diameter, which are properties of small-world networks. Small-world networks are characterized by

a high degree of local clustering and a low average path length between any two nodes. These properties make small-world networks efficient for information transfer, which is important in many real-world systems.

Recent Models

1. *R-MAT Model:* The RMAT model generates a network by recursively dividing the network into four quadrants and randomly selecting one of four rules to add edges to each quadrant. The four rules are defined by four parameters (A, B, C, D) that control the probability of adding an edge to each quadrant. These parameters can be adjusted to generate networks with specific topological properties. The RMAT model generates graphs that are characterized by their degree distribution, clustering coefficient, and community structure. It can generate graphs with a power-law degree distribution, high clustering coefficient, and strong community structure, making it useful for modeling real-world networks with these features.

2. *Kronecker-Graph Model:* The Kronecker graph model generates a network by starting with a small seed graph and iteratively applying a Kronecker product operation to generate a larger graph. The Kronecker product operation takes two matrices and produces a larger matrix whose entries are the products of the corresponding entries in the two matrices. The seed graph is typically a small graph with a few nodes and edges, and the Kronecker product operation is applied to this graph to generate a larger graph with a specific structure. The Kronecker graph model generates graphs that are characterized by their community structure and degree distribution. It can generate graphs with power-law degree distributions and strong community structure, making it useful for modeling real-world networks with these features.

1.2 Structural properties of real world social networks

Real-world social networks exhibit a number of structural properties that distinguish them from other types of networks. Here are some of the key structural properties of real-world social networks:

1. *Small-world phenomenon:* Real-world social networks tend to exhibit the small-world phenomenon, which means that any two individuals in the network are separated by a relatively small number of intermediaries. This property is often characterized by a low average path length and high clustering coefficient.

2. *Power-law degree distribution:* The degree distribution of real-world social networks often follows a power-law distribution, which means that there are a few highly connected nodes (hubs) and many nodes with relatively few connections. This property is also known as scale-free networks.

3. *Community structure:* Real-world social networks tend to have a modular or community structure, where nodes are organized into clusters or communities based on shared attributes or interests. This property is often characterized by a high modularity index or clustering coefficient.

4. *Homophily:* Real-world social networks exhibit homophily, which means that individuals tend to associate with others who are similar to themselves in terms of demographics, interests, or other attributes.

5. *Triadic closure:* Real-world social networks exhibit triadic closure, which means that nodes tend to form triangles or other closed loops in the network. This property is related to the small-world phenomenon and contributes to the high clustering coefficient in social networks.

1.3 Our Solution

Our approach will focus on improving the *clustering coefficient*, while maintaining the *community structure* as well as the *scale-free* nature of the network. This will be achieved through applying *dynamic homophilicity* and *transitivity* in our algorithms. The overview of our solution is as follows:

1. Design a **Stochastic Block Model (SBM)** to create an initial model of the desired network, using a dynamic homophily approach.

2. Run a **Triadic Closure** algorithm on the obtained model.

3. Add the remaining number of nodes using variants of the **Preferential-Attachment-based** algorithm (we have designed 2 variants).

1.3.1 Stochastic Block Model

The stochastic block model is a type of generative model for random graphs that are characterized by community structure. It assumes that the nodes in the network can be partitioned into K communities, and that the probability of an edge between two nodes depends on their community memberships. Specifically, the stochastic block model assumes that the probability of an edge between nodes i and j is given by a $K \times K$ matrix P, where P_{mn} represents the probability of an edge between nodes i and j in communities m and n respectively.

1. Initially, we assign the probabilities in the matrix P using two uniform random distributions: one for inter-community links $[P_{mn}]$ and the other for intra-community links $[P_{nn}]$.

2. However, with the addition of each link in the network, the probability matrix changes. If an inter-community link is added between the communities m and n, P_{mn} decreases with some *threshold value.* If an intra-community link is added in the community n, P_{nn} increases with the same *threshold value.* This is done with the idea to induce new nodes to connect more

with the communities which they belong to, if their community is itself well connected.

$$Threshold\ Value\ [th] = \frac{Average\ degree}{Number\ of\ nodes}$$

1.3.2 Triadic Closures

As discussed before, it refers to the phenomenon of transitivity in social networks, i.e. the tendency for two people who have a mutual friend to become friends themselves. It is one of the most fundamental properties of social networks and can be seen as a mechanism for the formation of clusters or communities within a network.

We apply a triadic closure algorithm where the tendency/probability of nodes i & j to have a link between them (if they didn't prior) is proportional to the *number of common neighbors* they have.

1.3.3 Preferential Attachment algorithms

Preferential attachment is a mechanism of network growth in which new nodes in a network preferentially attach to nodes that are already highly connected. This process leads to the formation of scale-free networks, in which a few highly connected nodes (hubs) coexist with many nodes with relatively few connections.

Preferential attachment is a common mechanism of network growth in many real-world systems, including social networks, biological networks, and the World Wide Web. It has been used to model the growth of scientific collaboration networks, the spread of information in social networks, and the evolution of online communities, among other phenomena.

We have proposed two variants: *Constant Preferential Attachment* & *Variable Preferential Attachment* algorithms.

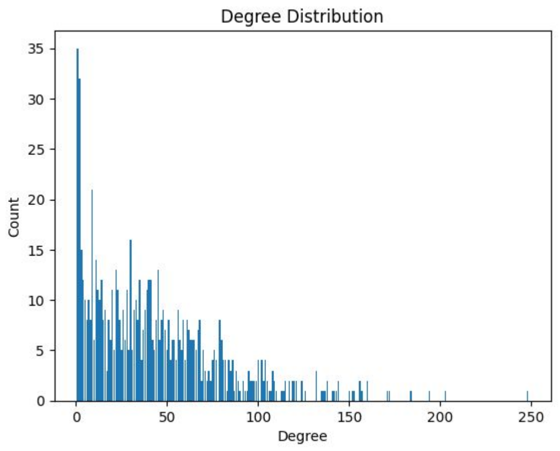
1. **Constant Variation:** Choose an existing node at random to link with (as per the preferential attachment algorithm in the Barabasi-Albert model), and link with it. Link to all neighboring nodes of the chosen node, until you either run out of links, or you run out of neighbors. In the latter case, repeat the above steps until the links are exhausted.

2. **Variable Variation:** Choose an existing node at random to link with (as per the preferential attachment algorithm in the Barabasi-Albert model), and link with it. Link to one of the neighboring nodes of the chosen node at random, with higher chance of being chosen being those nodes with higher degrees. Repeat the above steps until the links are exhausted.

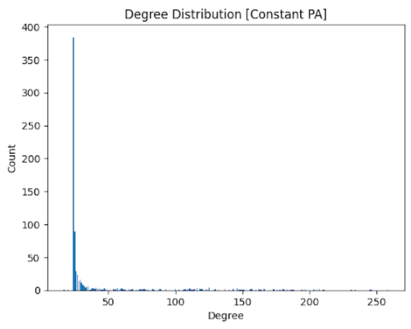
1.4 Results & Conclusion

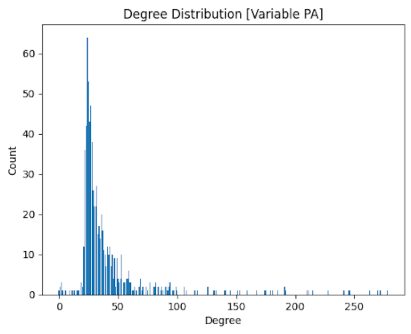
To test our approach, we used **CalTech dataset** with the following property:

- Nodes: 769
- Average Degree: 43.31
- Average Clustering Coefficient: 0.41
- Number of Communities: 11

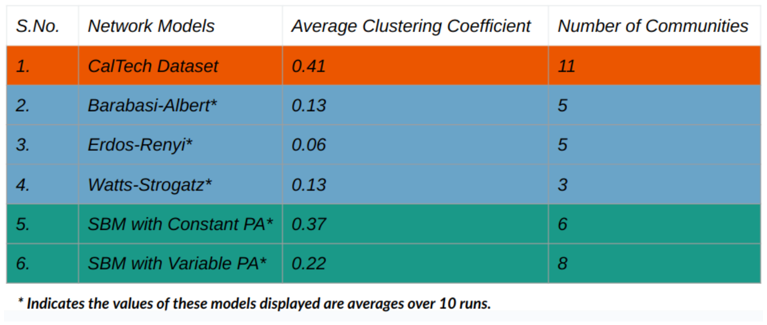


This is how the degree distributions of our approaches look as compared with the real world dataset.





The performance analysis with respect to related models is as follows:



S.No.	Network Models	Average Clustering Coefficient	Number of Communities
1.	CalTech Dataset	0.41	11
2.	Barabasi-Albert*	0.13	5
3.	Erdos-Renyi*	0.06	5
4.	Watts-Strogatz*	0.13	3
5.	SBM with Constant PA*	0.37	6
6.	SBM with Variable PA*	0.22	8

** Indicates the values of these models displayed are averages over 10 runs.*

2 Problem Statement 2: Finding optimal edge for maximum reduction in stress centrality

The stress centrality is number of shortest paths through a node such that the node is not an endpoint of the path

$$SC(u) = \sum_{\{s \in V | s \neq u\}} \sum_{\{t \in V | t \neq s \,\&\, t \neq u\}} \sigma_{st}(u)$$

Now it is the measure of load/importance of a node. Our aim is to find an edge whose addition will result to maximum reduction in the stress centrality of a node.

Application:

- Load balancing: If a particular node has high stress centrality, then it depicts it has higher load as comparative to others, finding such an edge would balance the load.
- System Design: To an existing system, we can calculate such an edge which would result in overall reduction of stress centrality. This would enable to lower the cost of infrastructure along with high efficiency.

2.1 Calculation of stress centrality

Algorithm to find stress centrality of nodes

1. For each node we will apply a BFS. We will maintain a parent array and distance vector for each node during BFS. If from a current node u in the BFS, we get a shorter distance path to the adjacent node v, then we will update v's parent list to current node($\{u\}$) and update the minimum distance for v. Else if the we get distance equal to that of previous distance, then we will add this vertex into the parent list of the vertex v. *(This denotes multiple shortest paths between pair of nodes).*

2. Now from the parent array of each vertex we will recursively recover all the shortest paths. And update the number of shortest paths for each vertex.

As for the calculation of stress centrality we have to iterate through all the shortest paths, it is optimal to find the stress centrality of all node at once as there is very low overhead. A single edge addition can affect the stress centrality of all the nodes.

2.2 Brute Force Approach

We can try all the possible n^2 edges. We will add one edge at a time and calculate the stress centrality of the node, then remove it and repeat the process. We will choose the edge which will give minimum stress centrality.

Complexity: $O(n^2 \cdot C(SC))$ where $C(SC)$ denotes the complexity of calculating stress centrality.

In real world network there are millions-billions of vertices i.e $n \sim 10^6$. Thus, the above algorithm is computationally costly and not feasible. In the following section we will try to reduce this complexity.

2.3 Our Solution

2.3.1 Greedy Approach

Consider a node with degree d. Now we want to reduce the stress centrality of the given node v i.e we have to reduce the number of shortest paths through it $\sigma_{st}(v)$.

1. Consider two distinct neighbours of v let it be p and q. Now the number of shortest paths from $v \rightarrow p$ and $v \rightarrow q$ be n_1 and n_2 respectively then the total number of shortest paths from these paths will be $n_1 \cdot n_2$.*(shortest path from $i \rightarrow j$ and shortest path from $j \rightarrow k$ forms the shortest path $i \rightarrow k$).* To cutoff these shortest paths we can add an edge pq. Thus, we will try this for every pair of neighbouring vertices. Complexity: $O(d^2 \cdot C(SC))$.

2. Now other case is one neighbour p is part of too many shortest paths through pv let it be n_1. Now the total number of shortest paths it will contribute to the concerned node v is $\sum_{ver \neq v \in N(v)} n_v * n_{ver}$. Thus, to reduce this we will find the node which has maximum number of shortest paths through pv. To find this we will perform the $BFS(v)$, and maintain the number of shortest paths for each node. The total number of shortest paths for any node is the sum of number of shortest paths from its already visited nodes in the BFS. Once a node is removed from BFS queue, we would have recorded the total number of shortest paths from the removed node to v. We will sort the nodes according to its number of shortest paths through pv in descending order. Now we will select the first node which forms a non-existing edge with v from this list.
Complexity: $O(n\,log(n) \cdot C(SC))$.

Overall complexity:

$$O((d^2 + n\,log(n)) \cdot C(SC))$$

2.3.2 Parallel Computing Approach

In this approach we would make use of parallel computing architecture. In our project we used the *CUDA* programming for parallel execution of the algorithm. It is GPU programming framework developed by Nvidia.

In parallel computing, we make use of GPU's ability to perform high computation. It has multiple streaming multiprocessors (SM) which contain 32 CUDA thread. Each thread is an execution unit. We designed a CUDA kernel which is basically a parallel version of the brute force approach. The algorithm used is as follow:

- For a source node u we assign the task of calculating the reduction in stress centrality for the edge addition uv is allotted to different thread i.e. each thread corresponds to different node v.
- As each thread are assigned to a different vertex, for higher number of vertices we can divide them into blocks of multiple threads. In case we exceed the GPU capacity we will perform the computation in batches.

The kernel can be optimized, by developing a GPU kernel for the task of stress centrality calculation and incorporating the new kernel into our existing kernel.The overall of complexity is:

$$O(n \cdot C(SC))$$

2.3.3 Graph Neural Network (GNN) approach

The graph neural network is machine learning framework which forms the node embedding using the feature relations of the node and its neighbour. We used *GraphSAGE*, it calculates the node embedding by considering neighbouring nodes and neighbours of neighbours. It is used for representative learning of large graphs. It creates a low dimensional vector for nodes which capture the feature relations.

In this we cannot train the model with various graphs for learning as it is purely based on node embedding, which depends on individual structural properties of graphs. Thus, instead of training on several graphs we will train the model using the graph which we want to analyse. And this model can be used for finding edge for any node in the graph.Upon training the model, it will have an understanding of how to create a node embedding (node representation) from its neighbourhood. Model can directly be used to assess the stress centrality without actually calculating the stress centrality.

For the learning phase we will keep the node feature as stress centrality. Now we will train the model with the node features. Then for the node for which we want to find an optimal edge we will try all non-existing edges and calculate the node embedding. The edge which results in minimum node embedding will be chosen. Complexity:

$$o(n^2)$$

Note it is 'small-O' notation.

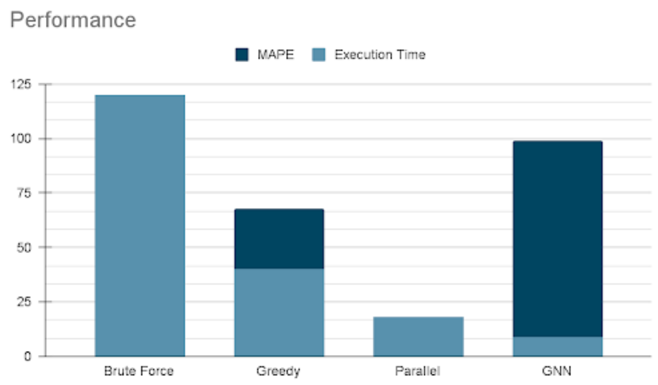
2.3.4 Proposed model

GNN+RL: We propose a more robust approach for the above problem which is GNN + RL. Here Reinforcement Learning is used to make the edge selection based on reward as negative stress centrality *(as we want to minimize it)*. Now to capture the node embedding GNN can be used to store the structural information about the graph. For training data we create BTER –(Block Two level Erdos-Renyi) model which would create a graph which maintain given degree distribution and clustering coefficient for low number of nodes. This model would generate small graphs which are representative for large real-life network.

2.4 Results & Conclusion

To test our approach, we used **CalTech dataset** with the following property:

- Nodes: 769
- Average Degree: 43.31
- Average Clustering Coefficient: 0.41
- Number of Communities: 11



As it can be seen that from our solution the parallel computing approach is the best. As for the brute force and parallel solution we have $MAPE = 0$ as they produce the optimal result. In greedy approach which is better than the brute force and worse than parallel have $MAPE = 28\% - 40\%$, but it generates about 45% average reduction in the stress centrality of nodes. GNN approach

have very low execution time even compared to parallel solution but it has very high MAPE. The source of the error can be because of approximation learning it will not work as global solution. As every neighbourhood can be different and number of shortest paths through a node depends on global structure and not on node's neighbourhood. Thus increasing the standard deviation from the local inference of the GNN. A properly modelled GNN solution with tolerable MAPE can become the best solution.
For now, the best solution is parallel solution. But it has infrastructural limitation as GPUs have limited memory which creates an execution limit. We cannot process a very large data into the GPU because of memory limitation, so we have to use the classical CPU memory (RAM) copy it to the GPU memory batch-wise. However, for network of large size this method of data replication and offloading creates an overhead to the algorithm and become the bottleneck to the solution. Currently the strongest Nvdia GPU is RTX 4090 which provide 24 GB of kernel memory, whereas big network data easily scale up-to 100 GB.