Inferred Gene Regulatory Networks and Prostate Cancer

Michael Dempsey

Dec 16, 2024

1 Introduction

One of the primary motivating questions for this project was, how can we create a network based solely on gene expression data? What information can we glean from inferred networks, particularly when it comes to cancer research? The paper "Prostate Cancer Gene Regulatory Network Inferred from RNA-Seq Data", by Moore *et al.* [6], explores these questions in depth while offering detailed explanations about their methods, making it a useful paper for recreation.

Gene regulatory networks (GRNs) consist of genes as nodes and the regulatory relationships between the genes as edges. The edges in GRNs can be signed, weighted, directed, or undirected [2]. It is believed that the effects of "true" GRNs can be observed in data such as gene expression profiles, making inferred GRNs a possibility [2]. This is useful because it is much easier to make approximations using gene expression data compared to trying to find the "true" underlying GRN.

The Moore *et al.* paper sought to infer a gene regulatory network using gene expression data from patients with prostate adenocarcinoma. By accomplishing this, the authors hoped to gain a better understanding of prostate cancer and its processes. They looked at both hub genes and genes closely linked to cancer in their discussion, creating visualizations of various subnetworks and offering in-depth analysis on certain genes [6].

For this project, a gene regulatory network was inferred using gene expression data. Using this inferred network, a broad analysis was performed, comparing the results of the project to the findings of the original paper.

2 Methods

In the original paper, the researchers collected data from The Cancer Genome Atlas (TCGA), which has "molecularly characterized over 20,000 primary cancers and matched normal samples spanning 33 cancer types" [7]. The database from TCGA includes RNA-sequence data that has been used to produce gene expression values. The paper looked at prostate adenocarcinoma in their analysis. For this project, data collection was conducted on the cBioPortal website, which had "Prostate Adenocarcinoma (TCGA, Cell 2015)" available to download. This data source contained 333 samples, which was the same as the original paper. The download contained many files, including patient attributes, somatic CNA data, and protein expression data. However, for this project, only mRNA gene expression data was used.

Prior to running the inference algorithm, the data needed to be preprocessed. In the gene expression file, each row contained the "Hugo Symbol", the "Entrez Gene ID", and the gene expression values for each sample. The "Hugo Symbol" and the "Entrez Gene ID" were combined into one column, and the average gene expression values were found for each row. The data was sorted in ascending order based on the average values, and the lower quartile was then removed. This removed many of the rows that contained just zeros, which were not useful for inferring the network. After this step, the number of genes went

from 20,502 to 15,376, which was the same as the original paper.

To infer the GRN, the paper used the BC3NET algorithm, which comes from the paper "Bagging Statistical Network Inference from Large-Scale Gene Expression Data" by de Matos Simoes *et al.* [4]. BC3NET, or Bagging C3NET, essentially takes the C3NET algorithm and uses bootstrapping. The C3NET algorithm originates from the paper "Inferring the Conservative Causal Core of Gene Regulatory Networks" by Altay *et al*. [1]. This algorithm calculates the mutual information between every pair of genes and determines if the values are significant based on a certain threshold. Edges are formed according to whether or not the calculated mutual information value is significant. The C3NET algorithm takes as input gene expression data, and it outputs an adjacency matrix. The BC3NET algorithm builds upon this by resampling the inputted data many times with replacement. For every resample, the C3NET algorithm is employed, each time producing its own set of edges. In the end, the edges across the resamples are averaged, producing a "weight" column.

With the preprocessing done, the gene expression data was imported into the programming language R, and the "bc3net" package was used [10]. The authors in the original paper mentioned that they conducted 100 bootstrap resamples and used the Bonferroni correction. These were both default parameters for the R function, which is why this project ended up just running the algorithm with the default settings. If there were any differences in the parameters used, it is because the original authors did not specify. While the authors discussed the algorithm, they did not mention their implementation of it.

After the BC3NET algorithm finished running, the outputted adjacency matrix was converted into an edge list, which contained the columns "From", "To", and "Weight". For this analysis, the "Weight" column was dropped, simplifying the network. The "From" and "To" columns did not offer any meaningful information about directionality because direction cannot be determined from the BC3NET algorithm alone. Therefore, only an undirected network could be produced. For the network, the Python package NetworkX was used [9].

For the analysis portion, properties of the network were first noted or calculated, including the number of nodes, the number of edges, the mean degree, the clustering coefficient, the mean geodesic distance, the degree distribution, the edge density, and the max degree. Then, the top degree nodes were found. A subnetwork was created for the node with the highest degree.

From the Catalogue of Somatic Mutations in Cancer, the Cancer Gene Census (CGC) catalogs "genes which contain mutations that have been causally implicated in cancer…" [3]. Tier 1 genes are genes that are most closely linked to cancer, with evidence of mutations and cancer-relevant activity. The list of genes from the Tier 1 group were downloaded and used to highlight cancer-relevant genes in the network visualizations.

For the final part of the analysis, the question was posed, how much of the structure of the gene regulatory network was driven purely by degree structure? To answer this question, null modeling was conducted, using the configuration model to produce 150 random networks. The Python package ConfigModel_MCMC was used for this step [12]. While NetworkX has its own built-in configuration model function, it is restricted in that it produces networks with self-loops and multi-edges [8]. In this project, the random networks should not have any multi-edges because the inference algorithm cannot produce any. Self-loops can be included because genes can regulate themselves, and the BC3NET algorithm does produce self-loop edges. The

ConfigModel_MCMC package offered more control, allowing the creation of random networks with or without self-loops and multi-edges while avoiding introducing bias or altering the degree structure [12].

After creating 150 random networks using the configuration model, the clustering coefficient for each network was calculated and plotted alongside the clustering coefficient calculated from the inferred network. Based on this visualization, it was concluded whether or not the degree structure was able to explain the empirical value.

3 Results

Running the gene expression data through the BC3NET algorithm and converting the resulting adjacency matrix into a network, the number of nodes or genes was 15,371, the number of edges was 81,068, the mean degree was 10.55, the clustering coefficient was 0.08, and the mean geodesic distance was 4.93.

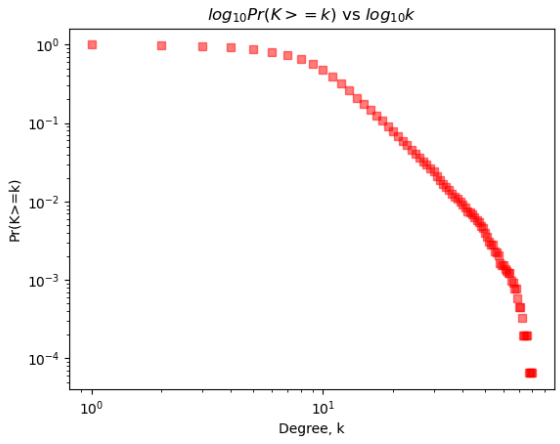


Figure 1: This plot depicts the degree distribution of the network. Most of the nodes tend to have a lower degree, with the distribution remaining fairly high until trending downward at about a degree of 10. The nodes with the highest degrees do not exceed 100.

	Original Paper	Recreation
Nodes	15,376	15,371
Edges	82,579	81,068
Mean Geodesic Distance	4.98	4.93
Edge Density	6.99×10^{-4}	6.86×10^{-4}
Max Degree	96	80

Table 1: This table shows a comparison between the attributes of the inferred network from the original Moore *et al.* paper and this project's results.

The values of the recreation tended to be a little lower than the original paper, but they were mostly similar to each other. While this project has about 1,500 less edges than the original paper, this difference is not too significant due to the size of the network (about 80,000 edges)..

Gene	Degree
HELZ	80
TRIP6	76
CCDC124	76
TAOK1	72
NDUFB7	72
GSTP1	71
BPTF	71
FAM156A	69
CCNL2	69

Table 2: This table shows the top-degree nodes from this project's inferred network.

HELZ had the highest degree in the network with a value of 80. This differs from the original paper, which had PIK3C2A as its top-degree node with a value of 96. Interestingly, the top-degree nodes between the original paper and the recreation were all different from each other. PIK3C2A only had a degree of 56 in the recreation.

Gene	Original Paper	Recreation
PIK3C2A	96	56
SCAF11	90	31
AURKAIP1	86	57
RIF1	85	56
NDUFA13	84	57
NOSIP	79	47
NAA10	78	53
CLTB	75	31
ASXL2	75	38

Table 3: This table highlights a comparison between the top-degree nodes from the original paper and their degrees in the recreation. In the recreated network, all the original top-degree nodes had a much lower degree.

In the original paper, a subnetwork of the top-most degree node, PIK3C2A, was created. The subnetwork not only depicted all of its connections, but it also highlighted the genes that were part of the tier 1 group of the CGC. To recreate this part of the paper, a subnetwork was created for the HELZ gene, which was found to be the highest-degree node in this project. Additionally, a subnetwork was created for the PIK3C2A gene, allowing for comparisons between the paper and the recreation.

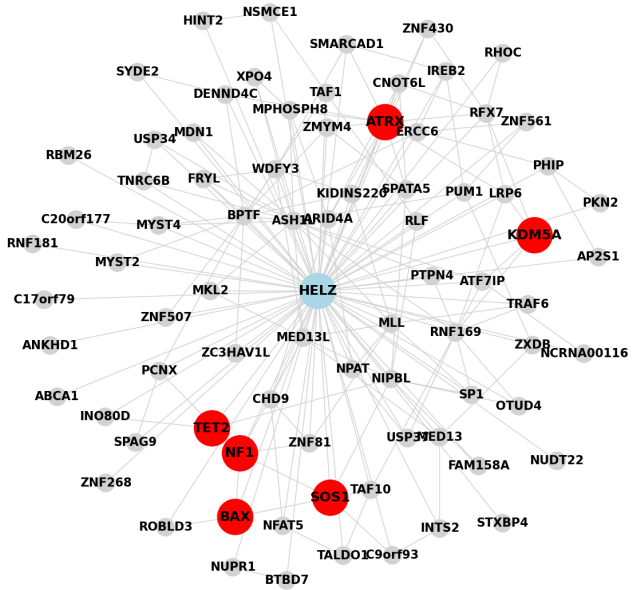
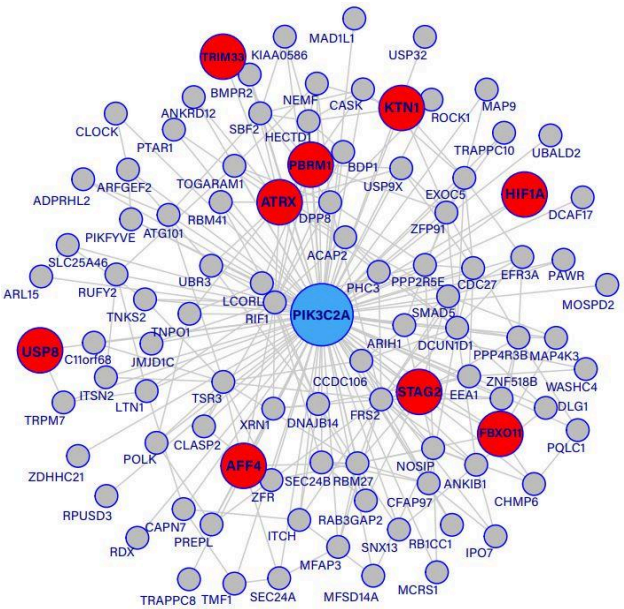
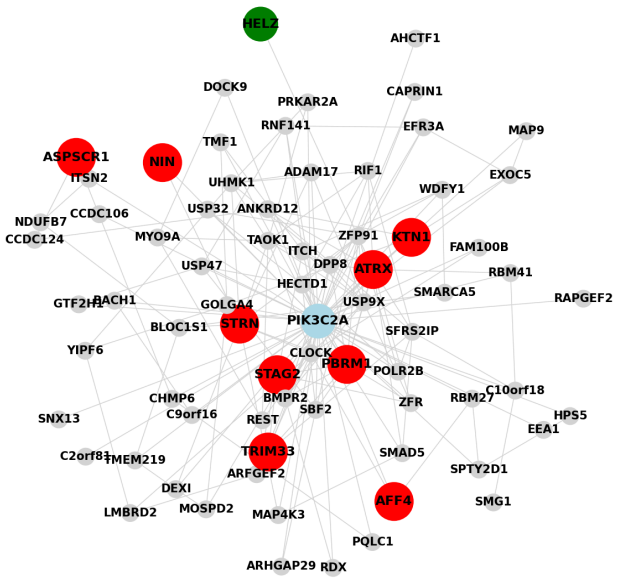


Figure 2: This subnetwork contains the HELZ gene and all of its neighbors. Highlighted in red are genes from the tier 1 group. These include ATRX, KDM5A, TET2, NF1, BAX, and SOS1.





Figures 3 and 4: These are two subnetworks for the PIK3C2A gene. The first figure comes from the Moore *et al.* paper, while the second figure was created using this project's inferred network. The subnetworks share many of the same genes despite the different degrees.

While the recreated version of the PIK3C2A subnetwork does have a lower number of connected nodes (hence the lower degree), it still contains many of the same genes as the subnetwork from the original paper, including many of the same tier 1 group genes. Some of these genes include TRIM33, ATRX, AFF4, PBRM1, KTN1, and STAG2. This demonstrates that the BC3NET algorithm did do its job despite the varying degree values. The algorithm still managed to find closely linked genes and formed edges between them. Potential reasons why there were differences in the degree-values are listed in the Discussion section.

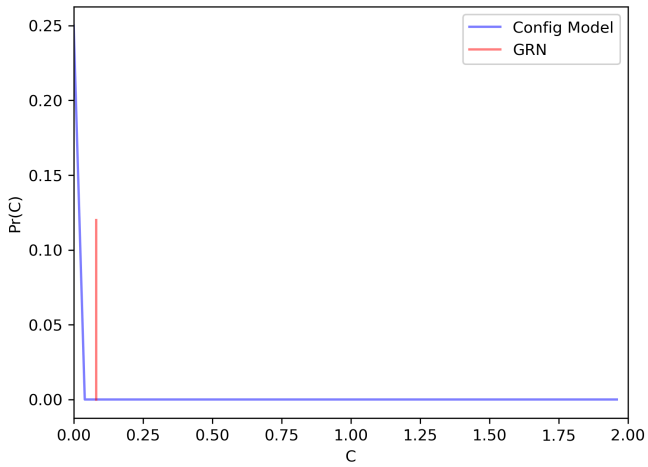


Figure 5: This figure shows the distribution of the clustering coefficient after 150 random networks were created using the configuration model. The red line represents the clustering coefficient of the inferred network.

From the figure above, the red line is positioned to the right of the other clustering coefficient values. Virtually all of the 150 random networks yielded a clustering coefficient of about 0.001, while the actual clustering coefficient for the inferred network was 0.08. From this, it can be concluded that the degree structure alone cannot explain the clustering coefficient.

4 Discussion

Overall, this project managed to recreate many of the findings from the original Moore *et al.* paper with some exceptions. The inferred networks shared very similar qualities, looking at the number of edges, the mean geodesic distance, the edge density, and the max degree. Additionally, nodes had many of the same neighbors across the original paper and the recreation. The main difference was the varying degrees for each gene, which affected the top-degree nodes.

As for the reason why the degree values varied quite a bit from the original paper, it could be due to various factors. For one, there might have been differences in the gene expression data.

The original paper took data directly from TCGA while this project extracted data indirectly from the cBioPortal website. While the number of samples were the same, there is always the potential for differences in the data collection or the preprocessing steps. Additionally, there might have been differences in the parameters for the BC3NET algorithm. This project made sure to include parameters that the original authors provided, but there is no guarantee that all the parameters were the same. The authors did not describe their specific implementation of the BC3NET algorithm, opting to provide more of an overview of how the algorithm works. Finally, there is plenty of room for human error in this project's research. While some of the findings inspire confidence, like the similar network attributes and genes having the same neighbors in both papers, the different degree values suggest the possibility of some kind of error.

Comparing the HELZ subnetwork to the PIK3C2A subnetwork from the original paper, HELZ had a lower degree and a smaller amount of tier 1 cancer genes. The authors in the original paper noted that PIK3C2A may play a role in cell survival, and proliferation has been shown to decrease by lowering the gene levels [6]. Interestingly, a study found that HELZ may be indirectly associated with proliferation while PIK3C2A might have a more direct correlation [11]. This particular study noted that decreasing the levels of PIK3C2A inhibited cell growth while decreasing the levels of HELZ did not have any significant effect. One interesting aspect to note from the recreated PIK3C2A subnetwork is that PIK3C2A is closely connected to HELZ (highlighted in green).

For the null modelling portion of the analysis, the empirical value that was used was the clustering coefficient. As noted previously, based on the figure, the degree structure cannot explain the clustering coefficient alone. This implies that there is some other factor contributing to the clustering you see in the network. This might have to do with the way genes regulate other genes, with genes serving similar functions clustering together [5].

Along with the clustering coefficient, the mean geodesic distance could have also been calculated for each of the 150 random networks. From this, it could be concluded whether or not the degree structure can explain the mean geodesic distance on its own. However, the computation time to do this was unfortunately really high, so this project ended up not including it. With proper computational power or a more clever means of calculating the mean geodesic distance, this could be an interesting avenue to explore in future projects.

Future works could include analysis on different cancers or involve a broad overview of biological processes. This paper focused on prostate cancer, but the methods can be applied to any gene expression dataset.

5 References

[1] Altay, G., & Emmert-Streib, F. (2010). Inferring the conservative causal core of gene regulatory networks. BMC Systems Biology, 4(1). https://doi.org/10.1186/1752-0509-4-132

[2] Badia-i-Mompel, P., Wessels, L., Müller-Dott, S., Trimbour, R., Ramirez Flores, R. O., Argelaguet, R., & Saez-Rodriguez, J. (2023). Gene regulatory network inference in the era of single-cell multi-omics. Nature Reviews Genetics, 24(11), 739–754. https://doi.org/10.1038/s41576-023-00618-5

[3] Cosmic. (2024, November 19). Cancer gene census. Cancer Gene. https://cancer.sanger.ac.uk/census

[4] de Matos Simoes, R., & Emmert-Streib, F. (2012). Bagging statistical network inference from

large-scale gene expression data. PLoS ONE, 7(3). https://doi.org/10.1371/journal.pone.0033624

[5] Eisen, M. B., Spellman, P. T., Brown, P. O., & Botstein, D. (1998). Cluster analysis and display of genome-wide expression patterns. Proceedings of the National Academy of Sciences, 95(25), 14863–14868. https://doi.org/10.1073/pnas.95.25.14863

[6] Moore, D., de Matos Simoes, R., Dehmer, M., & Emmert-Streib, F. (2019). Prostate cancer gene regulatory network inferred from RNA-Seq Data. Current Genomics, 20(1), 38–48. https://doi.org/10.2174/1389202919666181107122005

[7] National Cancer Institute. (n.d.). The cancer genome atlas program (TCGA). NCI. https://www.cancer.gov/ccg/research/genome-sequencing/tcga

[8] NetworkX. (n.d.-a). configuration_model. configuration_model - NetworkX 3.4.2 documentation. https://networkx.org/documentation/stable/reference/generated/networkx.generators.degree_seq.configuration_model.html

[9] NetworkX. (n.d.). NetworkX documentation. https://networkx.org/

[10] Rdocumentation. (n.d.). Bc3net: Bc3net gene regulatory network inference. RDocumentation. https://www.rdocumentation.org/packages/bc3net/versions/1.0.4/topics/bc3net

[11] Schepeler, T., Holm, A., Halvey, P., Nordentoft, I., Lamy, P., Riising, E. M., Christensen, L. L., Thorsen, K., Liebler, D. C., Helin, K., Ørntoft, T. F., & Andersen, C. L. (2011). Attenuation of the beta-catenin/TCF4 complex in colorectal cancer cells induces several growth-suppressive micrornas that target cancer promoting genes. Oncogene, 31(22), 2750–2760. https://doi.org/10.1038/onc.2011.453

[12] UpasanaDutta98. (n.d.). Upasanadutta98/CONFIGMODEL_MCMC. GitHub. https://github.com/UpasanaDutta98/ConfigModel_MCMC

6 Code

```
import networkx as nx
import numpy as np
import matplotlib
import matplotlib.pylab as plt
import pandas as pd

%matplotlib inline
# Pre-processing
genes = pd.read_csv('data_mrna_seq_v2_rsem.txt', sep='\t')
genes["Hugo_Symbol"] = genes["Hugo_Symbol"] + "_" + genes["Entrez_Gene_Id"].astype(str)
genes['Hugo_Symbol'] =
genes.groupby('Hugo_Symbol').cumcount().add(1).astype(str).radd('_').radd(genes['Hugo_Symbol'].values)
genes_avg = genes.drop(columns=[genes.columns[1]])
genes_avg['Average'] = genes_avg.iloc[:,2:].mean(axis=1)
genes_avg = genes_avg.sort_values(by='Average', ascending=True)
lower_quartile = genes_avg["Average"].quantile(0.25)
filtered_genes = genes_avg[genes_avg["Average"] > lower_quartile]
gene_column = filtered_genes.iloc[:,0]
numeric_columns = filtered_genes.iloc[:,1:]
numeric_columns = numeric_columns.applymap(lambda x: np.log(x) if x > 0 else 0)
log_transformed_df = pd.concat([gene_column, numeric_columns], axis=1)
filtered_genes_drop_col = log_transformed_df.drop(columns=[log_transformed_df.columns[-1]])
filtered_genes_drop_col.to_csv("filtered_genes.csv", index=False)

# Create network from edge-list
edge_list = pd.read_csv("edge_list_final.csv")
edge_list = edge_list.drop(columns=['weight'])
edge_list['from'] = edge_list['from'].str.split('_').str[0]
edge_list['to'] = edge_list['to'].str.split('_').str[0]
G = nx.from_pandas_edgelist(edge_list, source='from', target='to')

def plot_CCDF(kis):
    kmax = max(kis)
    counts, bins = np.histogram(kis, bins=[i for i in range(kmax+2)], density=True)
    cumcounts = np.cumsum(counts)
    cumcounts = np.insert(cumcounts,0,0)
    plt.loglog(bins[1:-1], 1-cumcounts[1:-1], 'rs', alpha=0.5)
    plt.xlabel('Degree, k')
    plt.ylabel('Pr(K>=k)')
    plt.title(r"$log_{10}Pr(K>=k)$ vs $log_{10}k$")
    plt.savefig("Degree_Distribution.png")
    plt.show()
    return

def compute_MGD(G):
    total_length = 0
    Z = 0
    for node in nx.all_pairs_shortest_path_length(G):
        sum_length_for_node = 0
        for length in node[1].values():
            if length != 0:
                sum_length_for_node += length
                Z += 1
        total_length += sum_length_for_node
    return total_length / Z
```

```
# Calculate Network Attributes
n = G.number_of_nodes()
m = G.number_of_edges()
kmean = (2*m) / n
C = nx.transitivity(G)
ellmean = compute_MGD(G)
plot_CCDF(list(dict(nx.degree(G)).values()))
num_possible_edges = (n * (n-1)) / 2
edge_density = m / num_possible_edges

max_gene = None
max_degree = -1
for node, degree in G.degree():
    if degree > max_degree:
        max_gene = node
        max_degree = degree

sorted_genes = sorted(G.degree, key=lambda x: x[1], reverse=True)
sorted_genes_df = pd.DataFrame(sorted_genes[:9], columns=["Gene", "Degree"])

paper_genes = []
paper_genes.append(("PIK3C2A", G.degree("PIK3C2A")))
paper_genes.append(("SCAF1", G.degree("SCAF1")))
paper_genes.append(("AURKAIP1_54998_1", G.degree("AURKAIP1")))
paper_genes.append(("RIF1", G.degree("RIF1")))
paper_genes.append(("NDUFA13", G.degree("NDUFA13")))
paper_genes.append(("NOSIP", G.degree("NOSIP")))
paper_genes.append(("NAA10", G.degree("NAA10")))
paper_genes.append(("CLTB", G.degree("CLTB")))
paper_genes.append(("ASXL2", G.degree("ASXL2")))
paper_genes_df = pd.DataFrame(paper_genes[:9], columns=["Gene", "Degree"])

cgc_data = pd.read_csv('Census_allMon Dec  9 23_38_33 2024.csv')
known_cancer_genes = cgc_data["Gene Symbol"].tolist()

subnetwork_genes = [max_gene] + list(G.neighbors(max_gene))
subnetwork = G.subgraph(subnetwork_genes)
node_labels = {}
node_colors = []
node_sizes = []
node_font_sizes = {}
for node in subnetwork.nodes:
    if node == max_gene:
        node_colors.append('lightblue')
        node_labels[node] = node
        node_sizes.append(2000)
        node_font_sizes[node] = 17
    elif node in known_cancer_genes:
        node_colors.append('red')
        node_labels[node] = node
        node_sizes.append(2000)
        node_font_sizes[node] = 17
    else:
        node_colors.append('lightgray')
        node_labels[node] = node
        node_sizes.append(500)
        node_font_sizes[node] = 15
plt.figure(figsize=(12, 12))
pos = nx.spring_layout(subnetwork, seed=42, k=0.05)
```

```
nx.draw(subnetwork, pos, with_labels=False, node_color=node_colors, node_size=node_sizes,
font_weight='bold', edge_color='lightgray')
for node, label in node_labels.items():
    if label:
        plt.text(pos[node][0], pos[node][1], label, fontsize=node_font_sizes.get(node, 12), ha='center',
va='center', fontweight='bold')
plt.title("Subnetwork of HELZ (Topmost Connected Gene)")
plt.savefig("HELZNetwork.png")
plt.show()

target_gene = "PIK3C2A"
neighbors = list(G.neighbors(target_gene))
neighbors_of_neighbors = []
for neighbor in neighbors:
    neighbors_of_neighbors.extend(G.neighbors(neighbor))
filtered_neighbors_of_neighbors = []
for neighbor in neighbors_of_neighbors:
    if G.degree(neighbor) > 50:
        filtered_neighbors_of_neighbors.append(neighbor)
nodes_in_subnetwork = [target_gene] + neighbors + filtered_neighbors_of_neighbors
subnetwork = G.subgraph(nodes_in_subnetwork)
node_colors = []
node_labels = {}
node_sizes = []
node_font_sizes = {}
for node in subnetwork.nodes:
    if node == target_gene:
        node_colors.append('lightblue')
        node_labels[node] = node
        node_sizes.append(2000)
        node_font_sizes[node] = 17
    elif node == max_gene:
        node_colors.append("green")
        node_labels[node] = node
        node_sizes.append(2000)
        node_font_sizes[node] = 17
    elif node in known_cancer_genes:
        node_colors.append('red')
        node_labels[node] = node
        node_sizes.append(2500)
        node_font_sizes[node] = 17
    else:
        node_colors.append('lightgray')
        node_labels[node] = node
        node_sizes.append(500)
        node_font_sizes[node] = 15
plt.figure(figsize=(12, 12))
pos = nx.spring_layout(subnetwork, seed=20, k=1)
nx.draw(subnetwork, pos, with_labels=False, node_color=node_colors, node_size=node_sizes,
font_weight='bold', edge_color='lightgray')
for node, label in node_labels.items():
    if label:
        plt.text(pos[node][0], pos[node][1], label, fontsize=node_font_sizes.get(node, 12), ha='center',
va='center', fontweight='bold')
plt.title("Subnetwork of PIK3C2A")
plt.savefig("PIK3C2A.png")
plt.show()

self_loops = list(nx.selfloop_edges(G))
```

```
print("Self loops:", self_loops)

import ConfigModel_MCMC as CM
allow_loops = True
allow_multi = False
is_vertex_labeled = True
mcmc_object = CM.MCMC(G, allow_loops, allow_multi, is_vertex_labeled)
G2 = mcmc_object.get_graph()

def plot_nullAndEmpirical(nulls,emp,qlabel,nlabel,elabel,xlim,ylim):
    counts, bins = np.histogram(nulls,bins=50,range=(0,xlim), density=True)
    nreps = len(nulls)
    fig = plt.figure()
    ax1 = fig.add_subplot(111) # put multiple
    plt.plot(bins[:-1], counts/100, 'b-', alpha=0.5, label=nlabel)
    plt.plot([emp, emp], [0, ylim], 'r-', alpha=0.5, label=elabel)
    plt.xlabel(qlabel)
    plt.ylabel('Pr('+qlabel+')')
    plt.xlim(0, xlim)
    plt.legend(loc='upper right');
    plt.savefig("clustering.png", dpi=300, bbox_inches='tight')
    plt.show()
    return

Cs   = []  # store the null values of C here
ells = []  # store the null values of ellmean here
for i in range(150):
    gen_G = mcmc_object.get_graph()
    Cs.append(nx.transitivity(gen_G))
plot_nullAndEmpirical(Cs,C,'C','Config Model','GRN',2,0.12)
```