Assignment 3: Well-Connected Communities Using CM++

Laxmi Vijayan

June 25, 2023

1 Introduction

Communities, in networks, can be described as a set of nodes that are more closely connected to each other than to other nodes within the network. Detecting communities is useful in a variety of domains, such as biology, sociology, and computer science, where complex systems are modeled as graphs. [11] In these networks, communities might be indicative of shared attributes amongst nodes, such as common function in neural and metabolic networks [5], a group of individuals with a shared interest [4], and a group of web pages on a shared topic in the World Wide Web [1]. As such, there is a large body of work on the community detection problem.

The community detection problem can be modeled as a graph partitioning problem, wherein a graph is partitioned into disjoint subsets of nodes. Usually, each node in a partition belongs to one and only one set. This, however intuitively clear, is conceptually and practically challenging to accomplish.[2] First, the definition of 'community' is ambiguous and various definitions exist based on scope (i.e. attributes of nodes, nodes, clusters within a graph). Nodes in real-world networks can also belong to multiple clusters. Second, evaluating partitions and identifying *good* partitions from overwhelmingly large possibilities is necessary.[2] One approach to identifying good partitions is optimization through the use of quality functions.[3]

A popular optimization method is through the use of the Constant Pott's Model (CPM) quality function which is the default optimization criterion for the Leiden Algorithm.[13] This function uses a linear resolution parameter (γ). It can be written in several ways and one formulation is a summation of communities:

$$Q = \sum_c \left[m_c - \gamma \binom{n_c}{2} \right] \tag{1}$$

where m_c is the total number of edges inside the community c and $\binom{n_c}{2}$ represents the number of possible edges within the community (n_c represents the number of nodes in community c).[13] By summing the difference between the total number of edges and the total possible number of edges over all communities, the CPM function determines the difference between the actual number of edges and the expected number of edges in a random network. Maximizing this value helps identify more densely connected communities. The γ term allows for adjusting the resolution of the communities detected; positive values result in smaller and denser communities, while negative values yield larger and less dense communities. The internal edge density of communities is higher than γ, while the external edge density of communities is lower.

A community is considered internally dense if it takes several "edge cuts" to separate one community into two separate communities. The Leiden algorithm guarantees that for every cluster, a subset of nodes within the cluster is connected to remaining nodes with at least the density of the resolution parameter (γ):

$$E(S, C - S) \geq \gamma ||S|| \cdot ||C - S|| \tag{2}$$

where E represents the edge cut, S represents the subset of nodes, C represents the cluster, and the product of γ, the cardinality of S and $C - S$ is greater than or equal to the edge cut. While this may ensure sufficiently well-connected large clusters, this has been demonstrated to result in loosely connected small to moderate-sized clusters.[12]

In this report, we seek to apply the cm++ pipeline [8] to improve the connectivity of clusters. Two networks from the Stanford Network Analysis Platform's Large Network Dataset Collection [10] were cleaned, partitioned using the CPM quality function at different resolution values and modified through the use of the Connectivity Modifier++ (cm++).[8] The connectivity of the clusters after the initial partition and after the use of cm++ were evaluated to examine the change in connectivity of the clusters. A final aim was to evaluate the ease of use and accuracy of the cm++ pipeline.

2 Methods

2.1 Networks

Two .TXT edge lists were downloaded from the Stanford Network Analysis Platform's Large Network Dataset Collection.[10] These edge lists were converted into .TSV files and passed to the cm++ pipeline. A brief description of the networks is provided in the following sections.

2.1.1 High-Energy Particle Physics (citHepPh)

The Hep-Ph (high energy physics phenomenology) citation graph is a directed graph of papers published between January 1993 to April 2003.[9] It has 34,546 papers (nodes) with 421,578 citation edges. If a paper within the graph cites another paper within the graph, this edge is included. If the papers cite or are cited by papers outside the graph, this edge is not represented.

2.1.2 US Patents Citation (citPat)

The US Patents citation graph contains all utility patents granted between January 1, 1963, to December 30, 1999. It contains 3,923,922 patents and 16,522,438 citations. Of all the patents included, 1,803,511 patents have only an in-degree.

2.2 cm++

The cm++ pipeline is modular and requires several user-defined parameters. These parameters included the input file name, the algorithm, the resolutions, iterations, the parallel limit, the threshold, and the number of processes (nprocs). Edge lists for both networks' .TSV files were run at two resolutions: $[0.01, 0.5]$. The Leiden Algorithm with the CPM partition type was used. The default iteration for this algorithm is 2 and was left unchanged. For increased efficiency, the parallel limit was set to 2 (so the graph would be partitioned at both resolutions concurrently), and 32 nprocs were used. Finally, the threshold outlined in Park et al., 2023 was used ($\log_{10}(n)$).

The modules of the cm++ pipeline, along with a brief description are provided below:

- **Stage 1. Clean-Up**: Any self-loops and parallel edges were discarded.
- **Stage 2. Leiden Clustering**: The cleaned edge list was partitioned using Leiden CPM and given resolutions.
- **Stage 3. Statistics**: For each cluster, the number of nodes, edges, modularity, cpm score, connectivity, normalized connectivity, and conductance were calculated.
- **Stage 4. Pre CM Filtering**: All trees (which, by definition in graph theory, are poorly connected) and stars were filtered out of the clusters. Also, all clusters that were smaller than or equal to size 10 were discarded.
- **Stage 5. CM**: Before modifying, each cluster was evaluated to see if met the user-defined connectivity threshold and whether it was an extant cluster from the Leiden Clustering. Extant here meant that this cluster was unchanged by the filtering process. Then, clusters were passed through the connectivity modifier. Clusters that did not meet the user-defined threshold were split into separate clusters and re-evaluated until all clusters met the threshold.

- **Stage 6. Post CM Filtering**: Some modified clusters fell below the size threshold of greater than 10. These clusters were discarded.
- **Stage 7. Stats**: As in Stage 3, the number of nodes, edges, modularity, cpm score, connectivity, normalized connectivity, and conductance were calculated for each cluster.
- **Stage 8. Analysis**: The number of clusters, node coverage, node count, minimum, median, and max cluster size were calculated for Stage 2, Stage 4, Stage 5, and Stage 6.

2.3 Analysis

The cm++ pipeline returned several files for each stage, including two stats.csv files for Stage 3 and Stage 7, and JSON files for clusters at Stage 5, among others. These files were loaded into a dataframe using the pandas library.[15] Then, descriptive statistics were captured using the NumPy library [6] and plotted using matplotlib.pyplot [7] and seaborn [14] packages. Scripts for the analysis are attached separately.

3 Results

The cm++ pipeline took four minutes and forty-eight seconds to run both resolutions of the HepPh network and eight minutes and forty-five seconds to run both resolutions of the Patents network. The discrepancy in time is commensurate with the difference between the two networks. The stage-by-stage breakdown of the execution time is presented in Figure 1. Stage 8, the analysis stage, took the most amount of time of all the stages, and it took the most amount of time for the Patents network.

3.1 HepPh Network

After Clustering, Pre CM Filtering, CM, and Post CM Filtering, some descriptive statistics were generated for the clusters. These results for the HepPh network are summarized in Table 1 for both resolutions. At resolution 0.01, 310 clusters remained after all stages of the CM pipeline. The median cluster size for these clusters was 41 and the maximum cluster size at this stage was 744. The maximum size remained consistent through all stages.

The cm++ pipeline also tracked how many clusters were extant, meaning how many clusters were formed by the Leiden clustering algorithm and remained unchanged by any of the filtering stages and the cm modifications. For the 0.01 resolution, 40 clusters were extant. For the 0.5 resolution, 346 clusters were extant.

The cluster counts for the HepPh network were examined for both resolutions and plotted against the cluster size. This can be seen in Figure 2. It reinforces the data summarized in Table 1. At resolution 0.01, several large clusters were broken up and several clusters below the threshold were discarded. One tree was also discarded at the pre-CM filtering stage. However,

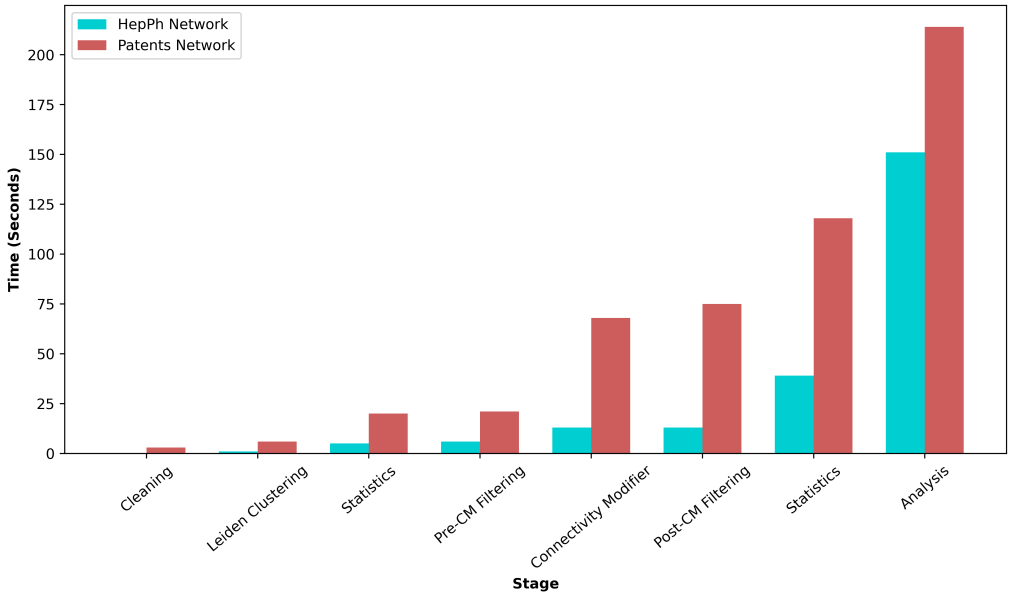

Figure 1: Execution time for each stage in seconds. The execution time in seconds is given for each stage for both networks, HepPh, and Patents. Each stage includes two edge lists, one for each resolution (0.01 and 0.5). The final stage, analysis, which involves calculating descriptive statistics for all clusters took the greatest amount of time. Further, it was more time-consuming for the Patents network, the larger of the two networks.

at resolution 0.5, all of the loss of node coverage occurred at the first filtering stage. This was largely due to the many clusters smaller than the threshold size which were discarded. No trees or stars were pruned from the clusters at this resolution and no changes were made to the clusters aside from this pre-CM filtering.

Connectivity, in this context, referred to the minimum number of edge cuts necessary to break a cluster into two separate parts. This was normalized by dividing the value by $\log_{10}(n)$ where n refers to the cluster size. This was then plotted as a log-log plot by cluster size to examine the change in connectivity between Leiden clustering and post-CM filtering (Figure 3). Figure 3 shows that at resolution 0.01, a large number of Leiden clusters had a normalized connectivity of less than 1 and were smaller than the threshold size. At resolution 0.5, all clusters had normalized connectivity greater than 1, but many clusters were below the threshold size.

3.2 Patents Network

After Clustering, Pre CM Filtering, CM, and Post CM Filtering, some descriptive statistics were generated for the clusters. These results for the Patents network are summarized in Table 2 for both resolutions. At resolution 0.01, 53,869 clusters remained after all stages of the CM pipeline and represented 54.5% node coverage. The median cluster size for these clusters was 24 and the maximum cluster size at this stage was 804. The maximum size remained consistent through all stages. For the 0.01 resolution, 587 clusters were extant. For the 0.5 resolution, 2,653 clusters were extant.

Stage & Resolution	Clusters	Node Coverage (%)	Nodes	min	med	max
Clustering 0.01	810	96	33,176	2	9	744
Pre CM Filtering 0.01	379	91.4	31,588	11	37	744
CM 0.01	402	85.2	29,447	3	27.5	744
Post CM Filtering 0.01	302	83	28,669	11	41	744
Clustering 0.5	7,569	85	29,359	2	3	55
Pre CM Filtering 0.5	346	16.3	5,617	11	14	55
CM 0.5	346	16.3	5,617	11	14	55
Post CM Filtering 0.5	346	16.3	5,617	11	14	55

Table 1: Summary statistics of the HepPh network at two resolutions. The table provides an overview of the characteristics of the HepPh network at two different resolutions: 0.01 and 0.5. The HepPh network is analyzed at several stages of the cm++ pipeline, including clustering, pre-CM filtering, CM, and post-CM filtering. The table displays the number of clusters, node coverage percentage, number of nodes, and minimum, median, and maximum cluster size values for each stage and resolution. At the resolution 0.01, the node coverage is 96% upon clustering and drops to 83% after post-CM filtering. At the resolution 0.5, the node coverage is 85% after clustering and drops to 16.3% after post-CM filtering. The minimum cluster size for both resolutions after clustering is 2. This increases to 11 for both after small clusters are discarded during the filtering stage. The maximum cluster size is unchanged for both resolutions regardless of stage.

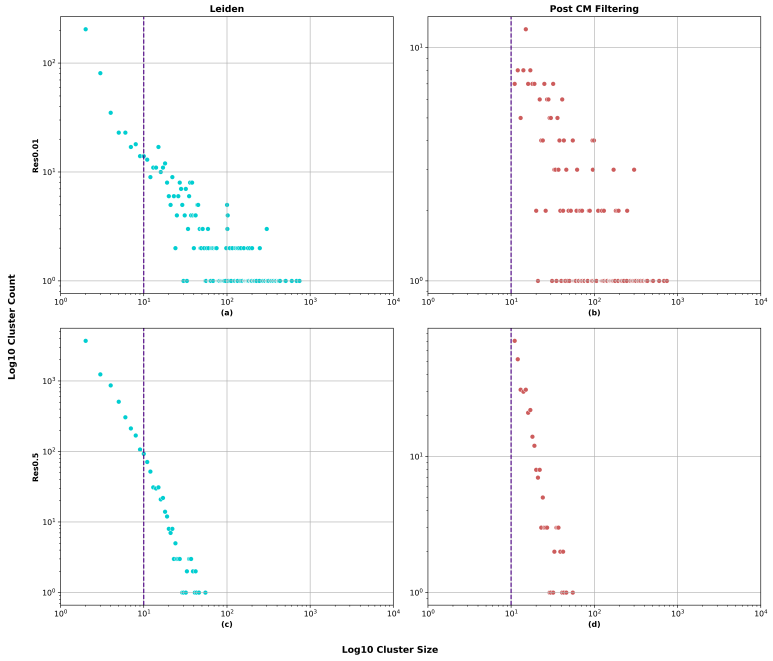

Figure 2: Log-log cluster counts by cluster size for HepPh network. The figure shows the log-log scatter of cluster count by cluster size for two resolutions (0.01 and 0.5) at the Leiden clustering stage (a, c) and after post-CM filtering (b, d) for the HepPh network. The indigo line at 10^1 is added as a reference to show the size threshold which cm++ enforces.

The cluster counts for the Patents network were examined for both resolutions and plotted against the cluster size. This can be seen in Figure 4. It reinforces the data summarized in Table 2. At resolution 0.01, several large clusters were broken up in addition to clusters below the threshold being discarded. A total of 22,483 trees and 415 stars were also discarded at the

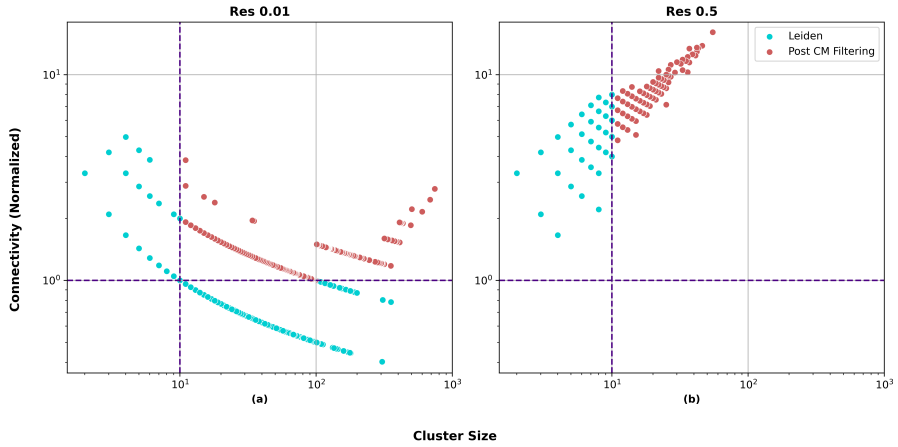

Figure 3: Log-log normalized connectivity by cluster size for HepPh network. The figure shows the log-log scatter of normalized connectivity by cluster size for the HepPh network. Two indigo lines, one at $x = 10^1$ and $y = 10^0$ are included for reference. The scatter in blue shows the clusters at the Leiden clustering stage and the scatter in red shows the clusters after the post CM filtering stage. No cluster at the post-CM clustering stage is smaller than 10 or has connectivity of less than 1. Comparatively, a majority of the clusters at the Leiden clustering stage at resolution 0.01 have a normalized connectivity below 10^0.

Stage & Resolution	Clusters	Node Coverage (%)	Nodes	min	med	max
Clustering 0.01	134,380	98.3	3,710,613	2	19	804
Pre CM Filtering 0.01	91,410	86.4	3,261,279	11	24	804
CM 0.01	92,909	63.8	2,408,822	3	13	804
Post CM Filtering 0.01	53,869	54.7	2,063,694	7	24	804
Clustering 0.5	1,143,221	72.2	2,725,458	2	2	116
Pre CM Filtering 0.5	2,653	1.1	39,912	11	13	116
CM 0.5	2,653	1.1	39,912	11	13	116
Post CM Filtering 0.5	2,653	1.1	39,912	11	13	116

Table 2: Summary statistics of the Patents network at different stages and resolutions. The table provides an overview of the characteristics of the Patents network at two different resolutions: 0.01 and 0.5. The Patents network is analyzed at several stages of the cm++ pipeline, including clustering, pre-CM filtering, CM, and post-CM filtering. Displayed are the number of clusters, node coverage percentage, number of nodes, and minimum, median, and maximum cluster size values for each stage and resolution. At resolution 0.01, the node coverage is 98.3% upon clustering and drops to 54.7%. At resolution 0.5, the node coverage is 72.2% upon clustering and drops to 1.1% after post-CM filtering. The minimum cluster size for both resolutions after clustering is 2. This increases to 11 for both after small clusters are discarded during the filtering stage. Surprisingly, one cluster of size seven survives post-CM filtering. The maximum cluster size is unchanged for both resolutions regardless of stage.

pre-CM filtering stage. Surprisingly, one cluster that did not meet the threshold survived the post-CM filtering (Figure 4).

However, at resolution 0.5, all of the loss of node coverage occurred at the first filtering stage. A drastic drop in node coverage, from 72.2% to 1.1% can be seen at this resolution. All the clusters smaller than the threshold size were discarded, but no trees or stars were pruned from these clusters. No further changes were made to the clusters, aside from this pre-CM filtering.

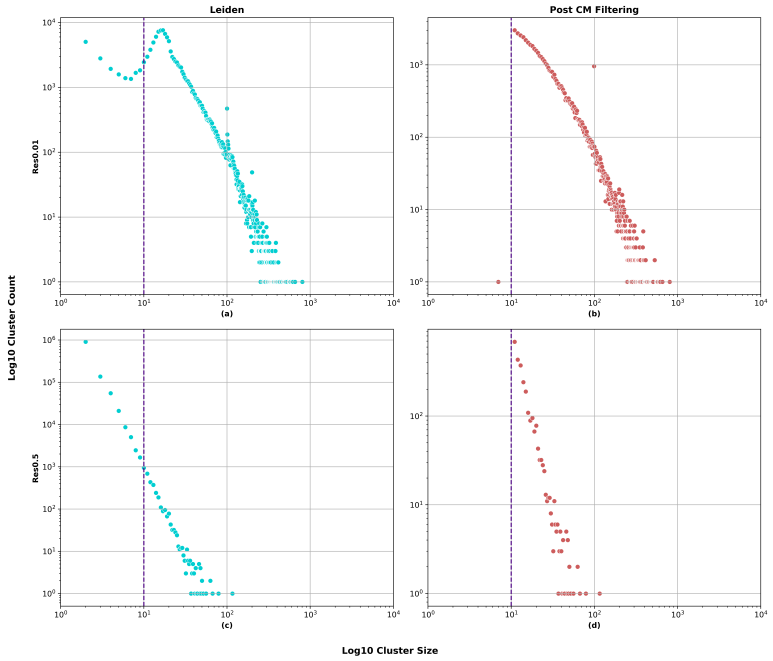

Figure 4: Log-log cluster counts by cluster size for Patents network. The figure shows the log-log scatter of cluster count by cluster size for two resolutions (0.01 and 0.5) at the Leiden clustering stage (a, c) and after post-CM filtering (b, d) for the Patents network. The indigo line at 10^1 is added as a reference to show the size threshold which cm++ enforces. Notably, one cluster smaller than this threshold is included after post-CM filtering at the resolution 0.01 (b).

As before with the HepPh network, a log-log plot of normalized connectivity by cluster size was examined to see the change in connectivity between Leiden clustering and post-CM filtering (Figure 5). The results were consistent with those seen for the HepPh network. At resolution 0.01, a large number of clusters had a normalized connectivity of less than 1 and many were smaller than 10 nodes. At resolution 0.5, most clusters were well-connected, but many were below the size threshold.

4 Discussion

4.1 Connectivity

Two SNAP citation networks, HepPh, and Patents, were clustered at two resolutions, 0.01 and 0.5. These clusters were then processed through the cm++ pipeline to examine changes in the connectivity of the clusters. Results for both networks at a resolution of 0.01 showed that while the Leiden algorithm guaranteed well-connected clusters, this was not often the case. Few of the clusters from the original clustering were extant after post-CM filtering.

Similarly, results for both networks at a resolution of 0.5 showed that while the Leiden algorithm's guarantee was met, many clusters were smaller than 10 nodes. As such, the drop in node coverage (85% to 16.3% for HepPh and 72.2% to 1.1% for Patents) was drastic. The remaining clusters were sufficiently large and met the low connectivity threshold we set ($\log_{10}(n)$). While

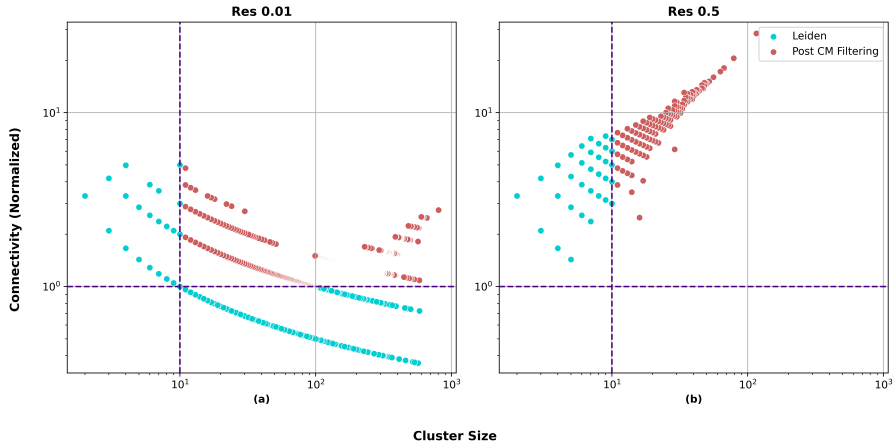

Figure 5: Log-log normalized connectivity by cluster size for Patents network. The figure shows the log-log scatter of normalized connectivity by cluster size for the Patents network. Two indigo lines, one at $x = 10^1$ and $y = 10^0$ are included for reference. No cluster at the post-CM clustering stage is smaller than 10 or has a connectivity of less than 1. Comparatively, a large majority of the clusters at the Leiden clustering stage at resolution 0.01 have a normalized connectivity below 10^0.

this is not concerning if the real-world networks being analyzed have large communities, it is concerning if the networks have small to moderate-sized natural communities.

These trials were done with the default Leiden iterations of two runs per clustering, but it is important to note that Traag, Waltman, and van Eck, 2019 state that the Leiden algorithm runs to convergence with more iterations. It would be an interesting experiment to see if increased iterations of clustering have any impact on how much change the cm++ pipeline makes. Another consideration here might be a resulting change in the number of clusters that are discarded as a result of the size threshold. It might be possible that with increased iterations of the algorithm, there might be fewer small clusters.

5 Conclusion

Comparing the results of the Leiden clustering and the cm++ was an interesting way of exploring the challenges associated with finding 'good' partitions. This assignment was particularly challenging because it required the application and use of all the tools we have been learning till now. The next step would be to run more experiments with the Leiden algorithm using different parameters, such as the number of iterations, and different networks of different sizes.

References

[1] Gary William Flake, Steve Lawrence, C. Lee Giles, and Frans M. Coetzee. Self-organization and identification of web communities. *Computer*, 35(3):66–71, mar 2002.

[2] Santo Fortunato and Claudio Castellano. *Community Structure in Graphs*, pages 490–512. Springer New York, New York, NY, 2012.

[3] Santo Fortunato and Mark E. J. Newman. 20 years of network community detection. *Nature Physics*, 18(8):848–850, jul 2022.

[4] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[5] Roger Guimerà and Luís A. Nunes Amaral. Functional cartography of complex metabolic networks. *Nature*, 433(7028):895–900, 2005.

[6] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[7] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[8] Vidya Kamath, Vikram Ramavarapu, Fabio Ayres, and George Chacko. Connectivity modifier pipeline. `https://github.com/illinois-or-research-analytics/cm_pipeline`, 2023.

[9] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, pages 177–187, New York, NY, USA, 2005. Association for Computing Machinery.

[10] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[11] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[12] Minhyuk Park, Yasamin Tabatabaee, Baqiao Liu, Vidya Kamath Pailodi, Vikram Ramavarapu, Rajiv Ramachandran, Dmitriy Korobskiy, Fabio Ayres, George Chacko, and Tandy Warnow. Well-connected communities in real-world networks, 2023.

[13] V. A. Traag, L. Waltman, and N. J. van Eck. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific Reports*, 9(1):5233, 2019.

[14] Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021.

[15] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.