A3: Graphing Clusters

Elaina Wittmer

June 2022

1 Introduction

This is an exercise in clustering citation data using the Leiden algorithm, a popular algorithm used to identify community structure in large networks, first published in 2019 by Vincent Traag, Ludo Waltman, and Nees Jan van Eck.[1] The Leiden algorithm guarantees connected communities, meaning that members of a community have paths to reach other members that do not require passing through a different community. These communities form clusters in the network, as observed in this report.

2 Methods - Non-Randomized Network

The data used came from a PubMed search on publications relevant to histone research which were published between 1980 and 1990 and written in English. This search returned 6,185 publications, 4,508 which could be matched against the open_citations database. After extracting any citation in the open_citations database that contained a doi from my search in either the citing or cited column, the resulting citation data set contained 119,841 nodes and 249,919 edges. After removing any rows that contained null values, the data set was slightly reduced to 249,366 edges and 119,548 nodes. I then found all the papers which were cited more than 500 times (n=21) and only included rows where one of these papers was cited. This reduced citation network contained 16,327 edges and 15,430 nodes. Each node was then assigned an integer index and this data frame was exported as a .tsv file.

I clustered the network using the Leiden algorithm (Version 1.1.10) using five different resolution values: 0.01, 0.02, 0.03, 0.04, and 0.05. Each clustering attempt returned 21 clusters containing 100 (res. 0.01) to 20 clusters (res. 0.05). The remaining clusters for all resolutions contained only one node each. Figure 1. shows the distribution of nodes between clusters, with the first twenty-one clusters containing 25 nodes and the remaining containing only one. I selected resolution 0.04 as well as the first 10 clusters for graphing, as 250 nodes seemed like enough nodes to provide a meaningful output while also not containing so many nodes that the graph became meaningless.[2]

3 Methods - Randomized Network

The methods for generating a random network were identical in terms of the data set and the filtering methods used. However, the citations were randomized by year such that a paper which cited a paper published in 1990 was guaranteed to cite a paper from 1990. This was accomplished by grouping cited publications by the year they were published, shuffling them, and creating a new column in the data frame for these values. This method *does not* guarantee that all citing papers will be matched with a different cited paper following randomization. 45.14% of the citing-cited pairs were different in the data presented here. While this percentage is not very good, all other attempts at guaranteeing different pairs failed. Details on these can be found in Section 6: Additional Information.

[1]https://doi.org/10.1038/s41598-019-41695-z

[2]I think I may have done clustering and filtering in the wrong order. In the future I would look into clustering the network first, then graphing a select few clusters. I believe this may be why only twenty-one clusters contained more than one node.

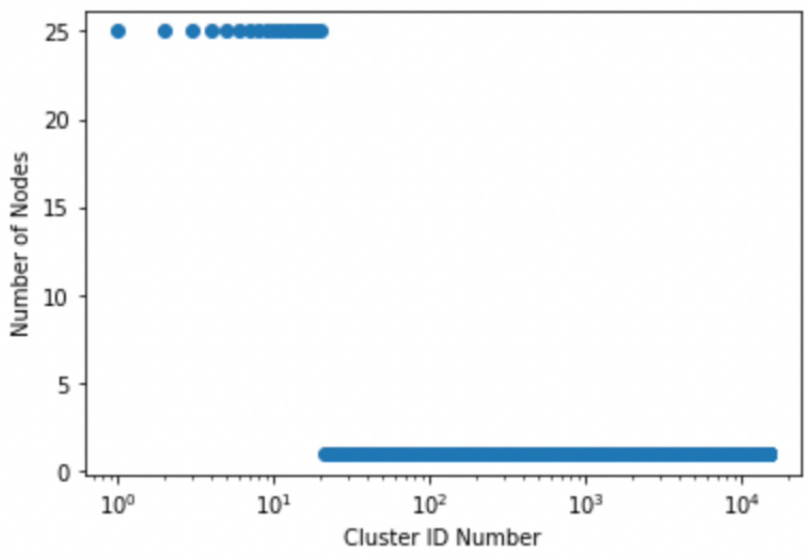

Figure 1: The scatter plot shows that the first twenty-one clusters contain twenty-five nodes each while the remaining clusters contain only one. Note that the x-axis is logarithmic.

This random network was also clustered using the Leiden algorithm, but only resolution 0.04 was used. The results of this clustering were the same as those in Figure 1: The first 21 clusters contained twenty-five nodes while the remaining clusters contained only one. The first ten clusters were selected for graphing.

4 Results

Figure 2 shows the clustering results for the non-randomized data. The figure shows 10 communities clustered around a central node, with three of these communities being detached from the others (5, 6, and 8), while the other 7 are connected to another community through at least one other node. For example, the clusters 9 and 10 have three shared nodes which cite papers in both communities. Cluster 3 could be said to be at least somewhat connected to clusters 2, 4, and 5, as there is a path connecting cluster 3 to each of these other clusters. In this way, it appears that some communities are more closely related to each other than others, while some are more detached.

Figure 3 shows the clustering results for the randomized data. The randomized data show less uniform distribution between clusters, though I'm not currently sure why, as there should be 25 nodes to a cluster according to the Leiden algorithm output. This may be a result of how I assigned edges, (See Graphing Clusters in A3_RandomNetwork.ipynb) where both nodes needed to be present in a cluster for an edge to be drawn. However, I'm not sure yet how this would impact the number of nodes present.

Regardless, the random network has a more scattered appearance. There are two isolated clusters present, 1 and 3. Clusters 5 and 6 are connected through one instance of co-citation, while clusters 2, 4, 7, 8, 9, and 10 are connected. At least one node is connected to clusters 2, 8, 9, 10 an instance of quad-citation that is not observed in the original network. The communities represented by the random network appear more scattered and fragmented than the original network does.

Returning to Figure 2, two of these clusters are connected by up to three nodes co-citing a paper in these clusters (clusters 2 and 3, clusters 9 and 10), three are connected by one instance of co-citation, and the remaining three form independent connected communities. This shows that these clusters are fairly independent but a few may have some commonalities.

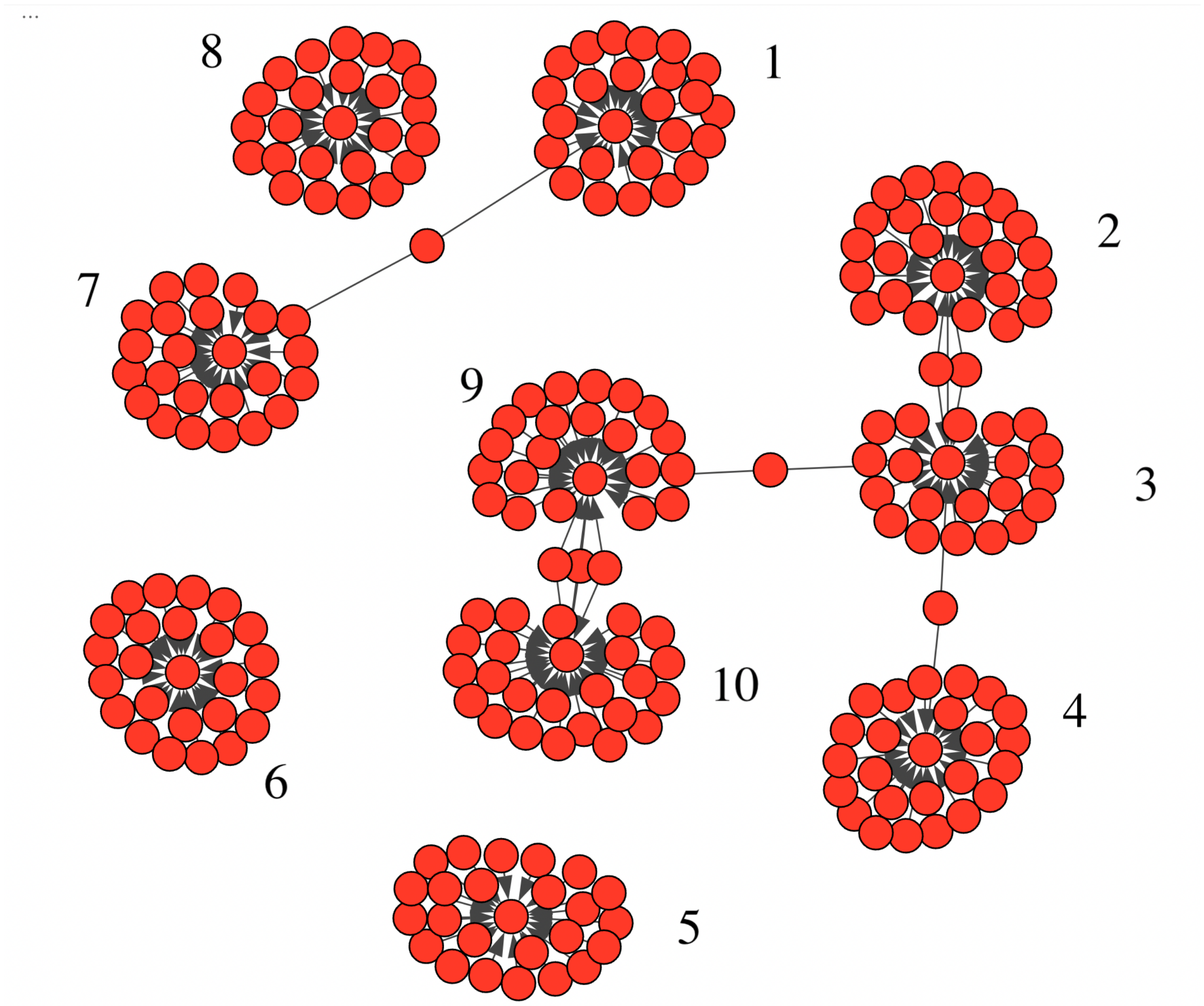

Figure 2: Clustering of the histone data using the Leiden algorithm (res. 0.04) and displaying the first ten clusters returned. The clusters have been labeled with integers for convenience of reference and are not associated with their identification number as assigned by the Leiden algorithm.

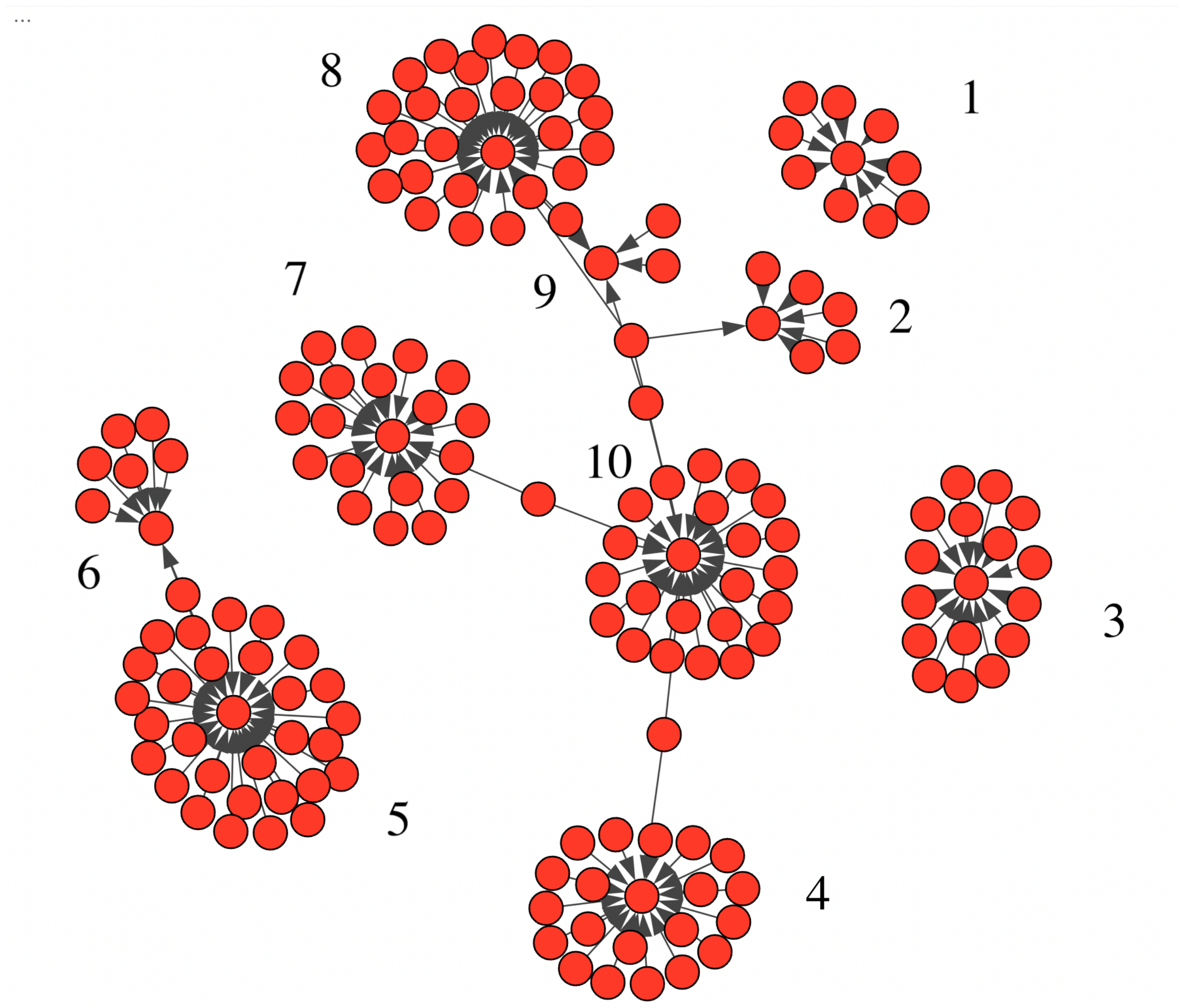

Figure 3: Clustering of the randomized histone data using the Leiden algorithm (res. 0.04) and displaying the first ten clusters returned. The clusters have been labeled with integers for convenience of reference and are not associated with their identification number as assigned by the Leiden algorithm.

5 Conclusions

While it was interesting to be able to graph clusters and see the groups of citations and the way papers interacted with each other, I'm not sure how much I trust the results, given how heavily I filtered the data before clustering. Going back to Figure 1, only 21 clusters contained more than 1 node, which doesn't seem right. In the future, I would cluster the entire network, then filter by cluster, rather than the other way around.

The in-degree was maintained for each cited node and the publication year was maintained, but only 45% of the citations were different. This could depend on the distribution of citations. If ten papers were published in 1990 and paper A was cited 80 times, paper B was cited 12 times, and the other 8 were cited 1 time each, it would be impossible to maintain both publication year and in-degree and also make sure that each cited paper is different than it was originally. Given that citations generally have a power-law distribution, I would expect a similar pattern. Therefore, 100% difference is unlikely to occur. However, 45% could maybe be improved.

6 Additional Information

My first attempt at randomization included saving all the citations in a dictionary by year such that the dictionary had the form {publication_year: {'citation1', 'citation2'...}} where the citations were stored in a set. For each row in the data frame, I would use the publication year to retrieve the set of citations for that year, make a copy of the set, delete the entry that matched the item I wanted to replace, then pull a random item from the set as the replacement. This approach worked for maintaining the publication year, but not for the in-degree of each node.

My second approach substituted the set with a list which stored every citation for a given year, allowing duplicates for papers cited more than once. It would pull a random item out from the list, and if the list item matched the item I wanted to replace, then it would run recursively until it picked an item that did not match. If it found an item that didn't match, it would delete the item from the list so it couldn't be used again. This approach caused many indexation errors, as well as infinite loops in the case where the options in the list all matched the item to be replaced.