OPEN ACCESS

Saudi Journal of Engineering and Technology
Abbreviated Key Title: Saudi J Eng Technol
ISSN 2415-6272 (Print) | ISSN 2415-6264 (Online)
Scholars Middle East Publishers, Dubai, United Arab Emirates
Journal homepage: https://saudijournals.com

Review Article

Community Detection in Network: Algorithmic Approaches with Python Programming

Tran Dang Hung[1*]

[1]Faculty of Applied Science, Ho Chi Minh City University of Industry and Trade, 140 Le Trong Tan Street, Tay Thanh Ward, Tan Phu District, Ho Chi Minh City, Vietnam

DOI: 10.36348/sjet.2024.v09i04.001 | **Received:** 23.02.2024 | **Accepted:** 05.04.2024 | **Published:** 08.04.2024

***Corresponding author:** Tran Dang Hung
Faculty of Applied Science, Ho Chi Minh City University of Industry and Trade, 140 Le Trong Tan Street, Tay Thanh Ward, Tan Phu District, Ho Chi Minh City, Vietnam

Abstract

Community detection is the identification of different communities or groups that exist within a network. This is useful in social network analysis (SNA) or what is great is performing whole network analysis (WNA), where humans interact with others as part of their various communities, but these approaches are not limited to the study of humans. These methods are to investigate any type of node that interacts closely with other nodes, whether those nodes are animals, hashtags, websites, or any other type of node in the network. In this work, we zoom in on communities that exist in a network. Community detection is a clear, concise, and appropriate name for what we are doing. Communities in the network would be worth exploring and understanding for further purposes. There are several methods and different approaches to detect community, but in this paper, I use two efficient methods to detect whole network which are named Louvain Method (LM) and Girvan-Newman Method (GNM). With LM, we can build a fast algorithm that is effective at community detection in massive networks and optimize the algorithm for better results. Using the GNM, a better approach that can identify the least number of edges that could be cut would result in a split network. We could do this by making an algorithm looking for the edges that the greatest number of shortest paths pass through.

Keywords: Community Detection, Social Network Analysis, Social Network, Network Analysis, Whole Network Analysis, Network, Networkx.

Copyright © 2024 The Author(s): This is an open-access article distributed under the terms of the Creative Commons Attribution **4.0 International License (CC BY-NC 4.0)** which permits unrestricted use, distribution, and reproduction in any medium for non-commercial use provided the original author and source are credited.

1. INTRODUCTION

It is useful in SNA, as humans interact with others as part of our various communities when we detect community, identifying the various communities or groups that exist in a network. We can also use these approaches to investigate any kinds of nodes that interact closely with other nodes, whether those nodes are animals, hashtags, websites, or any kind of nodes in a network. What communities would you be interested in exploring and understanding, and why? There are many good use cases for this. I can use it to understand the sentiment communities share about my product. I can use this to understand a threat landscape. I can use this to understand how ideas move and transform between different groups of people, etc. Be creative here. There are probably more uses for this than I can imagine.

In this paper, I will explore this in the context of human life, but it should not feel limited to only using this for social network analysis. This is very useful in SNA, but it is also useful in analyzing most network data, not just social network data. For instance, this can be useful in both cybersecurity, malware analysis and computational humanities, or in understanding how ideas move between groups and evolve.

Doing community detection, we have at least three different approaches, with the most frequently researched including the following: Node connectivity, Node closeness, Network splitting. *Node connectivity* has to do with whether nodes are part of the same connected component or not. If two nodes are not part of the same connected component, then they are part of a completely different social group, not part of the same community. *Node closeness* has to do with the distance between two nodes, even if they are part of the same

connected component. For instance, two people might work together in the same large organization, but if they are more than two handshakes away from one another, they may not be part of the same community. It would take several rounds of introductions for them to ever meet each other. Consider how many people you would have to go through to be introduced to your favorite celebrity. How many people would you need to be introduced to? *Network splitting* has to do with literally cutting a network into pieces by either removing nodes or edges. The preferred approach that I will explain is cuts on edges, but I have done something similar by removing nodes, shattering networks into pieces by removing central nodes.

That is at the end of discovery for community detection? I do not believe that. I hope that reading through this paper will give you some ideas for new approaches to identifying the various communities that exist in networks.

2. Getting Started with Community Detection

For getting stared, we will be using several different Python libraries: Networkx, pandas, scikit-network. These libraries should be installed. We also need a network to use. Let's use NetworkX's Les Miserables graph since it held several separate communities:

First, we load the network

```
Import networkx as nx
Import pandas as pd
G = nx.les_miserables_graph()
```

We do not need edge weights for this simple demonstration. So, I am going to drop it and rebuild the graph. Then we converted the Les Miserables graph into a pandas edge list, and we kept only the 'source' and 'target' fields, effectively dropping the weight field. Let's see how many nodes and edges exist in the network:

```
df = nx. to_pandas_edgelist (G) [['source',
'target']
# dropping 'weight'
G = nx. from_pandas_edgelist (df)
Print ('Number of nodes', len(G.nodes))
Print ('Number of edges', len(G.edges))
Print ('Average degree', sum (dict
(G.degree).values ())/len(G.nodes)
```

We have output:

```
Number of nodes 77
Number of edges 254
Average degree 6.597402597402597
```

Second, we need to build draw_graph function to draw graph instead of using draw function in the matplotlib.pyplot def draw_graph(G, show_names=False, node_size=1, font_size=10, edge_width=0.5):

```
Import numpy as np
From IPython.display import SVG
From sknetwork.visualization import
svg_graph
From sknetwork.data import Bunch
From sknetwork.ranking import PageRank
Adjacency = nx. to_scipy_sparse_matrix(G,
nodelist = None, dtype = None, weight =
'weight', format = 'csr')
Names = np. Array (list (G.nodes ()
Graph = Bunch ()
Graph. Adjacency = adjacency
Graph. Names = np. Array (names)
pagerank = PageRank()
Scores = pagerank. fit_transform (adjacency)
If show_names:
Image = svg_graph (graph.adjacency, font_size
= font_size, node_size-node_size, names =
graph.names, width = 700, height = 500, scores
= scores, edge_width = edge_width)
```

Else:

```
Image = svg_graph (graph.adjacency,
node_size = node_size, width = 700, height =
500, scores = scores, edge_width = edge_width)
Return SVG (image)
```

If we want to visualize the network in its entirety, by calling the draw_graph() like this:

```
draw_graph(G, show_names = True, font_size
= 12, node_size = 4,
edge_width = 1)
```

In the original Les Miserables graph, we should be able to see that there are several clusters of nodes that are very close to each other (communities), and there are a few critically important nodes. If those critically important nodes were removed, the network would shatter to pieces. We also have that there is no isolated node (nodes without edges), there are several nodes with a single edge (Figure 1).

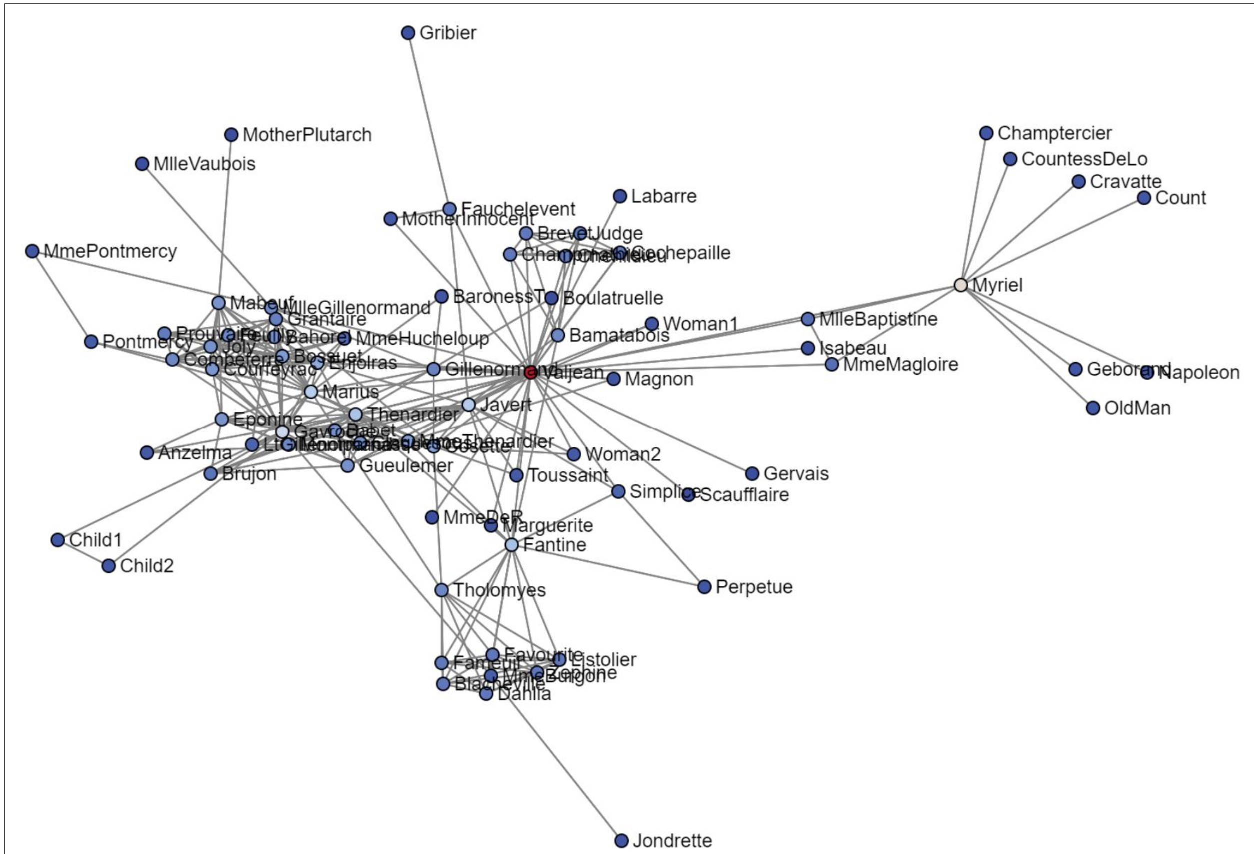

Figure 1: Les Miserables graph (original)

The last, we zoom in this graph a little, using k_core, only show nodes that have two or more edges, and not display labels so that we can get a sense of the overall shape of the network:

Draw_graph(nx.k_core(G, 2), font_size = 12, show_names = False, node_size = 4, edge_width = 0.5)

We will get the following visualization:

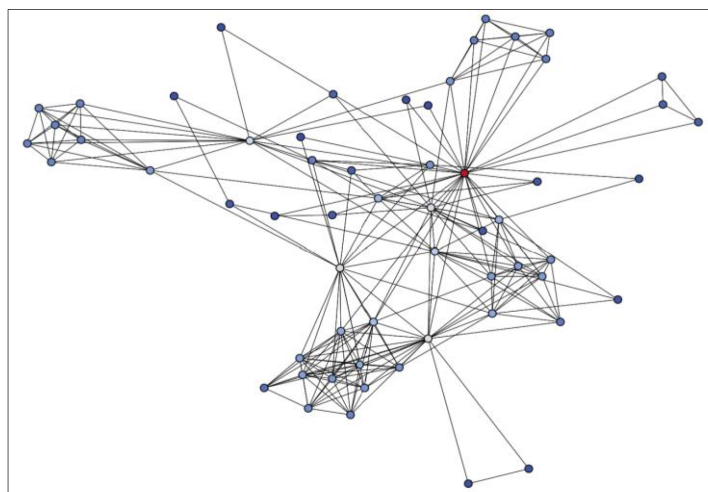

Figure 2: Les Miserables graph (zoom in)

The communitie graph should be a little clearer now (Figure 2). How many communities do there are? There are four, but there are smaller groups scattered around, and there is also likely a community in the center of the network. We begin our attempts at community detection.

3. Exploring Connected Components for Detecting Communities (Connected Components Method)

For understanding the various communities and structures that exist in a network is analyzing the connected components. As we can see, connected components are structures in networks where all nodes have a connection to another node in the same

component. Connected components can be useful for finding smaller connected components. Those can be thought of as communities as they are detached from the primary component and overall network, but the largest connected component is not typically a single community. It is usually made up of several communities, and it can usually be split into individual communities.

In the Les Miserables network, there is only one connected component. There are no islands or isolates. There is just one single component. It takes away a bit of the usefulness of inspecting connected components for this graph. There is a way around that, if we remove a few critically important nodes from a network, that network tends to shatter into pieces by two following steps:

Step 1: Remove five very important characters (['Valjean', 'Marius', 'Fantine', 'Cosette', 'Bamatabois'] from the network, and visualize the network again:

```
G_copy = G.copy()
G_copy.remove_nodes_from(['Valjean', 'Marius', 'Fantine', 'Cosette', 'Bamatabois'] )
draw_graph(G_copy, font_size = 12, show_names = True, node_size = 4, edge_width = 1)
```

We have the following visualization:

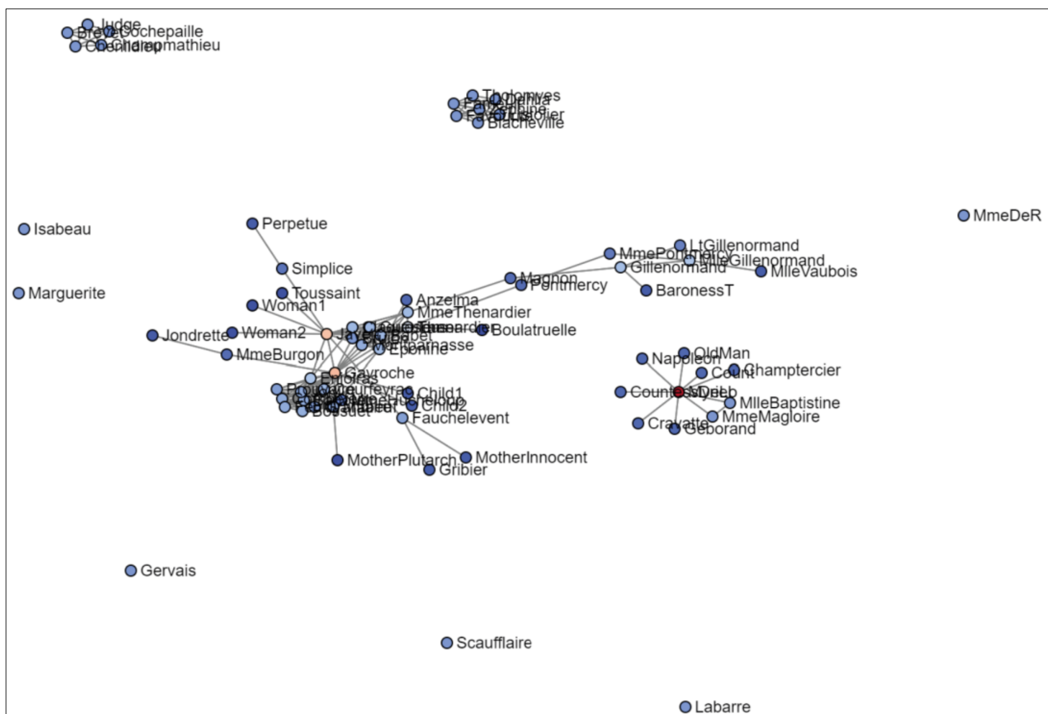

Figure 3: Shattered Les Miserables network

This network is much closer to how many real-world networks look. There's still one primary connected component (continent), there are three smaller connected components (islands), and there are six isolated nodes. There is no threshold for deciding that an island is a continent. It is just that most networks contain one super-component (continent), lots and lots of isolated nodes, and several connected components (islands). This helps to do more analyst.

What I just did could be used as a step-in community detection. Removing a few key nodes can break a network apart, pulling out the smaller communities that exist. Those critically important nodes held one or more communities together as part of the larger structure. Removing the important nodes allowed the communities to drift apart. It is not usually ideal. However, other actual approaches to community detection work similarly, by removing edges rather than nodes.

Step 2: Remove isolated nodes and inspect each one.

After shattering the network there are 10 connected components left, but 6-isolates are not connected to anything other than possibly themselves. We remove them before looking into connected components:

```
G_copy = nx.k_core(G_copy, 1) # Remove isolated nodes
Community = components [0]
G_community =
G_copy.subgraph(community)
draw_graph(G_community, show names =
True, node_size = 5)
```

Let's look at the visualization:

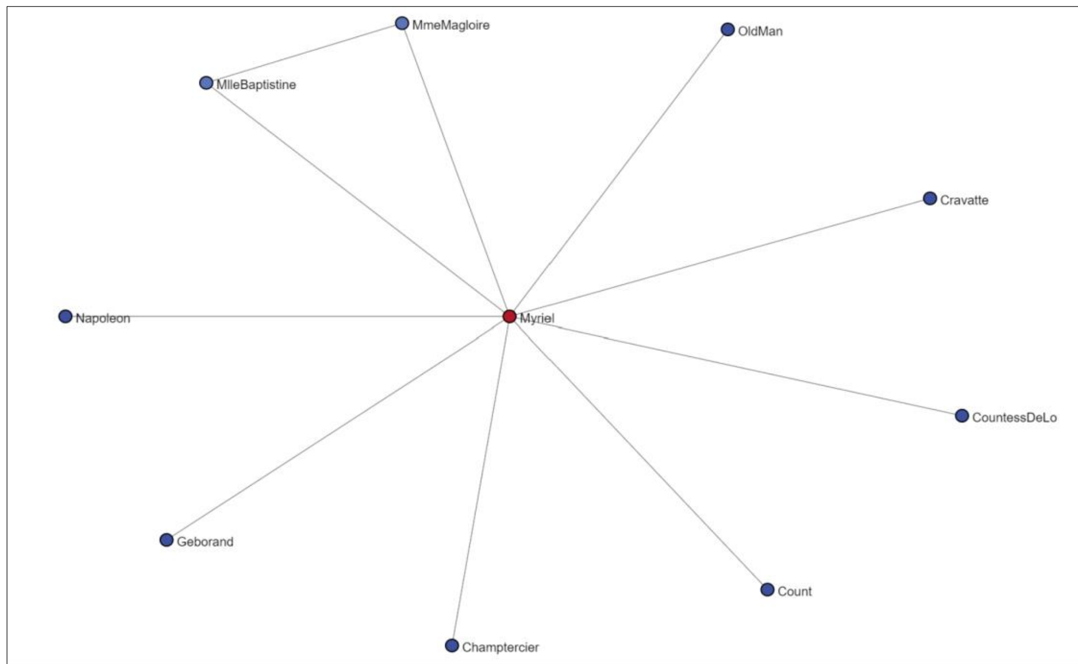

Figure 4: Component 0 subgraph of the shattered Les Miserables network

That is very interesting. The first connected component is almost a star network, with all nodes connecting to one central character, Myriel. However, if you look at the top left, you should see that two characters also share a link. That relationship could be worth investigating.

Let's look at the next component.

```
Community = components [1]
G_community = G_copy. subgraph
(community)
draw_graph (G_community, show_names =
True, node_size = 4)
```

This gives us the following output:

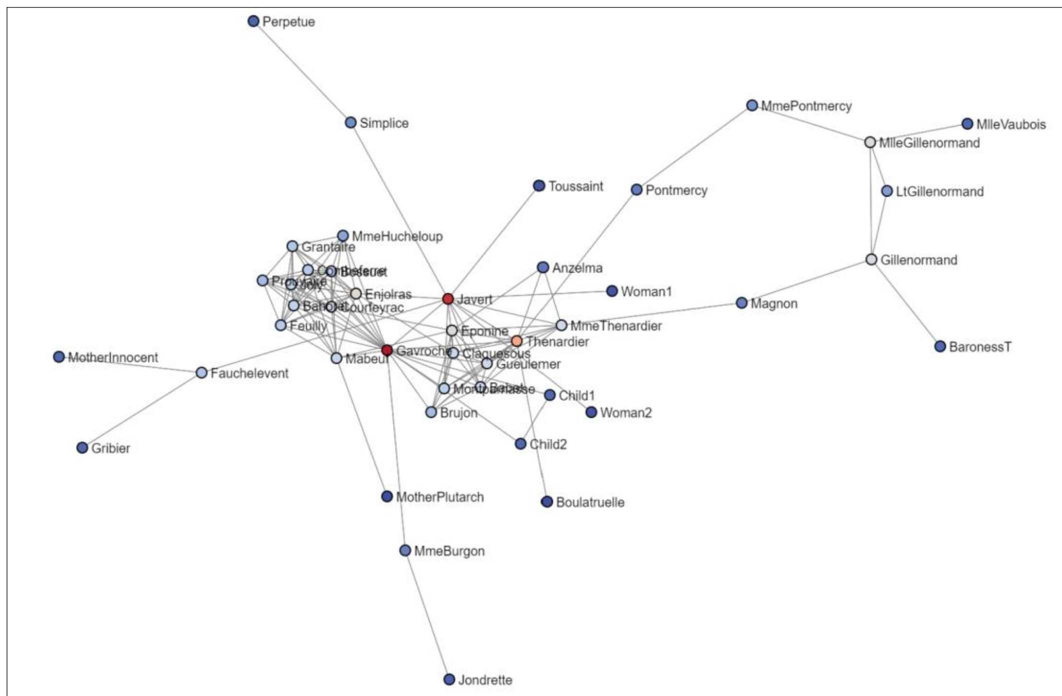

Figure 5; Component 1 subgraph of the shattered Les Miserables network

© 2024 | Published by Scholars Middle East Publishers, Dubai, United Arab Emirates

This is the primary component. It's the largest connected component in the shattered network. However, connected components are not ideal for identifying communities. Look slightly left of the center in the network - we should see two clusters of nodes ['Gavroche', 'Javert'], two separate communities. There is also at least one other community on the right. If two edges or nodes were removed, the community on the right would split off from the network.

Let's continue with the third component:

```
Community = components [2]
G_community =
G_copy.subgraph(community)
draw_graph(G_community, show_names =
True, node_size = 4)
```

We have following output:

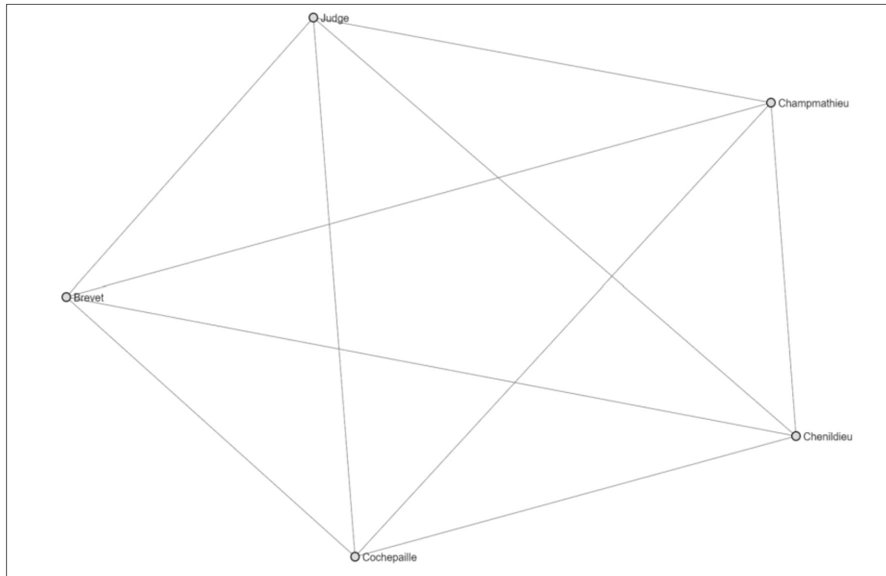

Figure 6: Component 2 subgraph of the shattered Les Miserables network

This is a strongly connected component. Each node has a connection to the other nodes in this network. If one node were removed, this network would remain intact. From a network perspective, each node is as important or central as each other node.

Let's move to the final component:

```
Community = components[3]
G_community =
G_copy.subgraph(community)
draw_graph(G_community, show_names =
True, node_size = 4)
```

We have the following output. It is a densely connected network. Each node is equally important or central. If one node were to be removed, this network would remain intact.

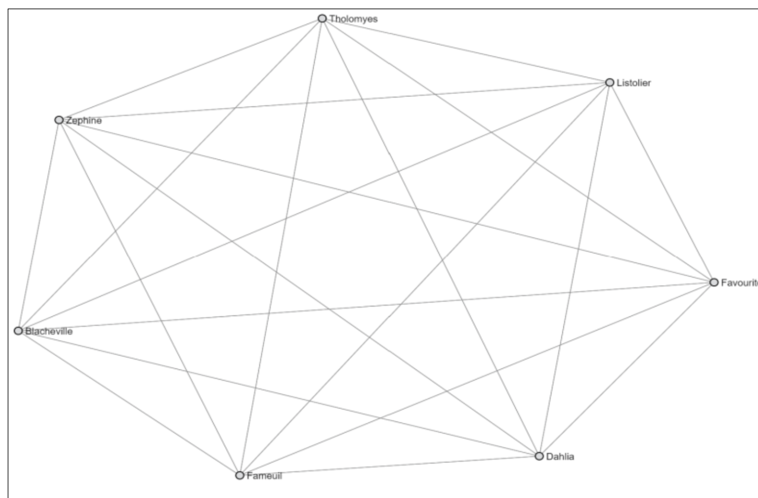

Figure 7: Component 3 subgraph of the shattered Les Miserables network

We were able to find three communities by looking at the connected components, but connected components did not draw out the communities that exist in the larger primary component. If we wanted to draw those out, we would have to remove other important nodes and then repeat our analysis. Removing away nodes is one way to lose information, so I do not recommend that approach, but it can be useful for quick ad hoc analysis.

Communities can be found while investigating connected components, I also do not consider investigating connected components to be community detection. I consider this one of the first steps that should be taken during any network analysis, and the insights gained are valuable, but it's not sensitive enough for community detection.

If your network contained no super-cluster of a connected component, then connected components would be pretty adequate for community detection. However, you would have to treat the super-cluster as one community, and the cluster contains many communities. The connected component approach becomes less useful with larger networks.

So, we move on to more suitable methods. First, dabbling in community detection by starting with the Louvain method, and then picking up other approaches. It's good to know that there are options.

4. Using the Louvain Method for Detecting Communities

The Louvain method works through a series of passes, where each pass contains two phases. The first phase assigns different communities to each node in the network. Initially, each node has a different community assigned to it. Then, each neighbor is evaluated, and nodes are assigned to communities. The first step concludes when no more improvements can be made. In the second phase, a new network is built, with nodes being the communities discovered in the first step. Then, the results of the first phase can be repeated. The two steps are iterated until optimal communities are found. Using the Louvain method, we have a fast algorithm that is effective at community detection in massive networks, and we can optimize the algorithm for better results.

The Louvain method has been included in more recent versions of Networkx, so if you have the latest version of Networkx, you will not need to use the community Python library. Your version will be different, if you use the older version, you need to use the community library approach. First, we build code that will help us draw Louvain partitions:

```
Import community as community_louvain
Import networkx as nx
G = nx.les_miserables_graph()
def draw_partition(G, partition):
Import matplotlib.cm as cm
Import matplotlib.pyplot as plt
# draw the graph
plt.figure(3, figsize = (12,12))
pos = nx.spring_layout(G)
# color the nodes according to their partition
cmap = cm.get_cmap('jet',
max(partition.values()) + 1)
nx.draw_networkx_nodes(G, pos,
partition.keys(), node_size = 40,
cmap = cmap, node_color = list
(partition.values()))
nx.draw_networkx_edges(G, pos, alpha = 0.5,
width = 0.3)
Return plt.show()
```

Starting to use the best partition function to identify the optimal partition using the Louvain method. During the testing, I found resolution = 1 to be ideal, but with other networks, you should experiment with this parameter:

```
Partition =
community_louvain.best_partition(G,
resolution=1)
draw_partition(G, partition)
```

This code has created a visualization:

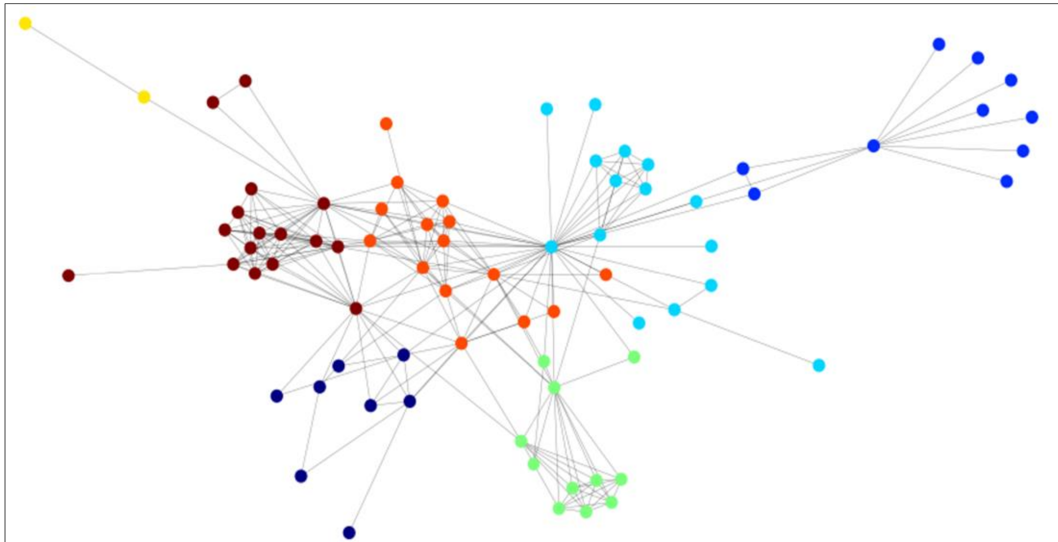

Figure 8: Louvain method community detection of the Les Miserables network

The draw partition () function in step 1 will color nodes by the communities that they belong to. The important thing is that separate communities have been detected, and each community of nodes is identified with a different color. Each node belongs to a different partition, and those partitions are the communities.

Second, we look at what is inside the partition variable:
Partition

{'Napoleon':1,
'Myriel':1,
'MlleBaptistine':1,
'MmeMagloire'1,
'CountessDeLo':1,
'Geborand':1,
…
'Grantaire':0,
'Childl':0,
'Child2':0,
'BaronessT':2,
'MlleVaubois':2,
'MotherPlutarch': 0}

To save space, I cut out some of the nodes and partitions. Each node has an associated partition number, and that's the community that it belongs to. I want to get a list of nodes that belong to an individual community, so I do:

[Node for node, community in partition. Items () if community == 2]

The Louvain method is so exciting. For one thing, it can scale to massive networks, allowing for research into the largest networks, such as the internet. Second, it's fast and practical. There is not a lot of point to an algorithm that is so slow as to only be useful on tiny networks. Louvain is practical with massive networks. This algorithm is fast and efficient, and the results are very good. This is an algorithm for community detection.

We have another option for community detection by using Girvan-Newman method.

5. Using Girvan-Newman Method for Detecting Communities

At previous, we noticed that the Les Miserables network consisted of a single large, connected component and that there were no isolates or smaller "islands" of communities apart from the large, connected component. To show how connected components could be useful for identifying communities, we shattered the network by removing a few key nodes. That approach is not typically ideal. While there is information in both nodes (people, places, things) and edges (relationships), in my experience, it is typically preferable to throw away edges than to throw away nodes.

A better approach than what we did previously would be to identify the least number of edges that could be cut that would result in a split network. We could do this by looking for the edges that the greatest number of shortest paths pass through - that is, the edges with the highest edge_betweenness_centrality. That is precisely what the Girvan-Newman algorithm does.

In a network, there are several nodes on two different sides connected by a few edges. It almost looks like a few rubber bands are holding the two groups together. If you snip the rubber bands, the two communities should fly apart, similar to how networks shatter into pieces when key nodes are removed. This is more surgically precise than removing nodes. There is less loss of valuable information. Losing information on certain relationships is a drawback.

Through a series of iterations, the Girvan-Newman method identifies edges with the highest

edge_betweenness_centrality scores and removes them, splitting a network into two pieces. Then, the process begins again. If not repeated enough, communities are too large. If repeated too many times, communities end up being a single node. So, there will be some experimentation when using this algorithm to find the ideal number of cuts.

The downside of this algorithm is that it is not fast. Calculating edge_betweenness_centrality is much more computationally expensive than the computations being done for the Louvain method. As a result, this algorithm ceases to be useful very quickly, as it becomes much too slow to be practical. However, if your network is small enough, this is a very cool algorithm to explore for community detection. It's also intuitive and easy to explain to others.

I tried this out with the Les Miserables graph. The graph is small enough that this method should be able to split it into communities quickly:

First, we import the algorithm, and need to pass the graph to the algorithm as a parameter. When we do this, the algorithm will return the results of each iteration of splits, which we can investigate by converting the results into a list:

```
From networkx.algorithms.community import girvan_ newman
Communities = girvan_newman(G)
Communities = list(communities)
Print ('Maximum number of iterations that the algorithm consisted of a single node:', len(communities))

# Output: 76
```

Second, I could investigate the various levels of splits and find the one that looks best for my needs. We have 76 iterations of splits kept in a Python list. It could be very early in the process, in the first 10 splits, or it might be a bit later. This part requires some analysis, further making this a bit of a hands-on algorithm.

For continuously, I assume that we found that the tenth iteration of splits yielded the best results. Let's set the tenth iteration results as our final group of communities, and then visualize the communities as we did with the Louvain method.

```
Communities = communities [9]
```

I will be keeping the tenth iteration results and dropping everything else. If I did not want to throw away the results, I could have used a different variable name. To see what these communities look like so that we can compare them against the other algorithms we discussed:

```
Community = communities [0]
G_community = G.subgraph (community)
draw_graph(G_community, show_names = True, node_size = 5)
```

This code has created a visualization:

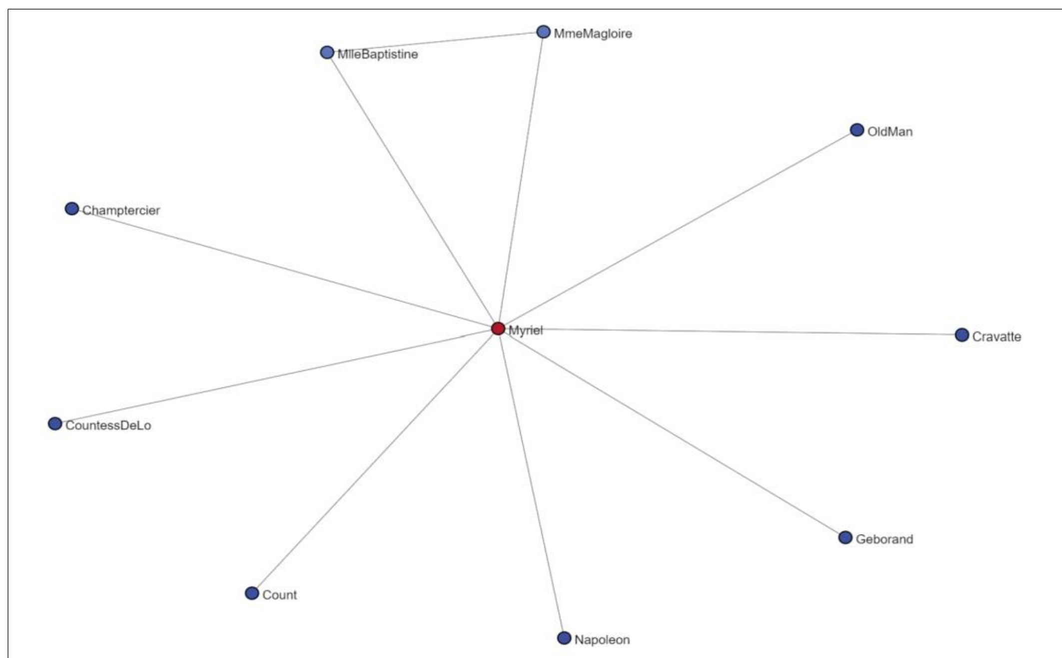

Figure 9: Girvan-Newman community detection of the Les Miserables network, community 0

This subgraph (community 0) is familiar the subgraph that I shattered the network by nodes and then visualized connected components. This algorithm split the network using edges and managed to find the same community.

We go forward with another communities:

Community = communities [1]
G_community = G.subgraph (community)
draw_graph(G_community, show_names = True, node_size = 5)

We have the following output:

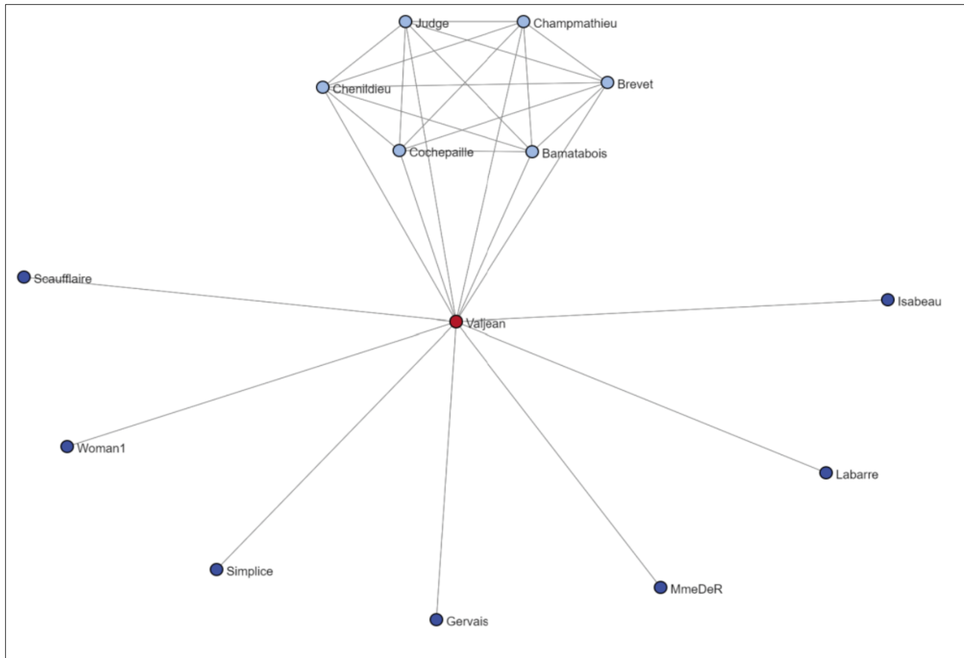

Figure 10: Girvan-Newman community detection of the Les Miserables network,

Community 1
It is not uncommon when community 1 has a densely connected group, as well as some less connected nodes.

Let's see community 2:

Community = communities [2]
G_community = G.subgraph (community)
draw_graph(G_community, show_names = True, node_size = 5)

We have the following output:

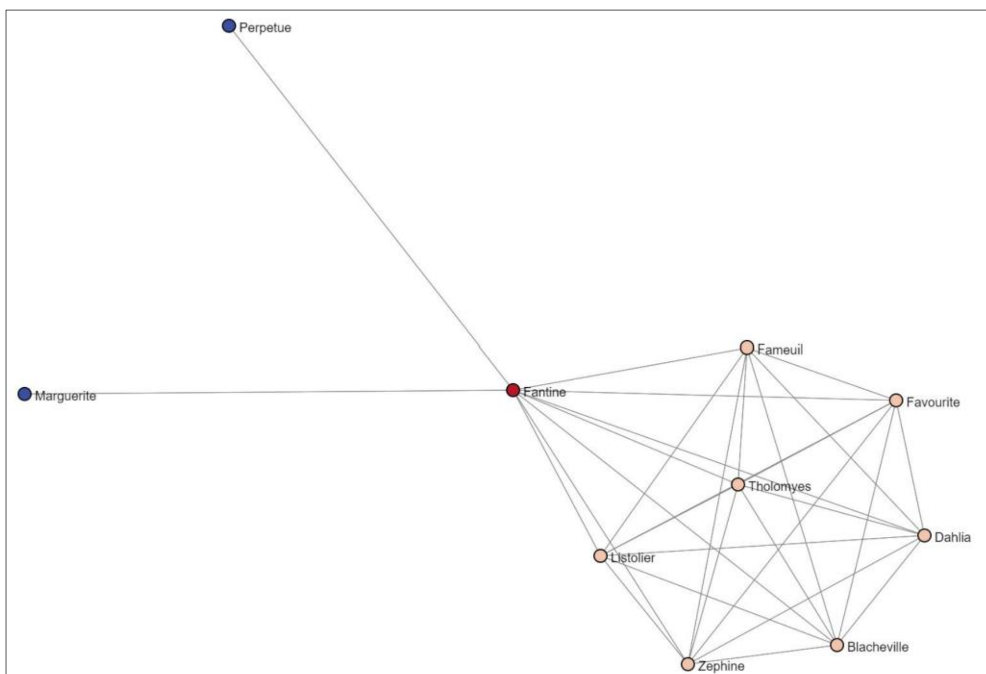

Figure 11: Girvan-Newman community detection of the Les Miserables network,

Community 2

Let's move to the community 3:

Community = communities [3]

G_community = G.subgraph (community)

draw_graph(G_community, show_names = True, node_size = 5)

We have the following output:

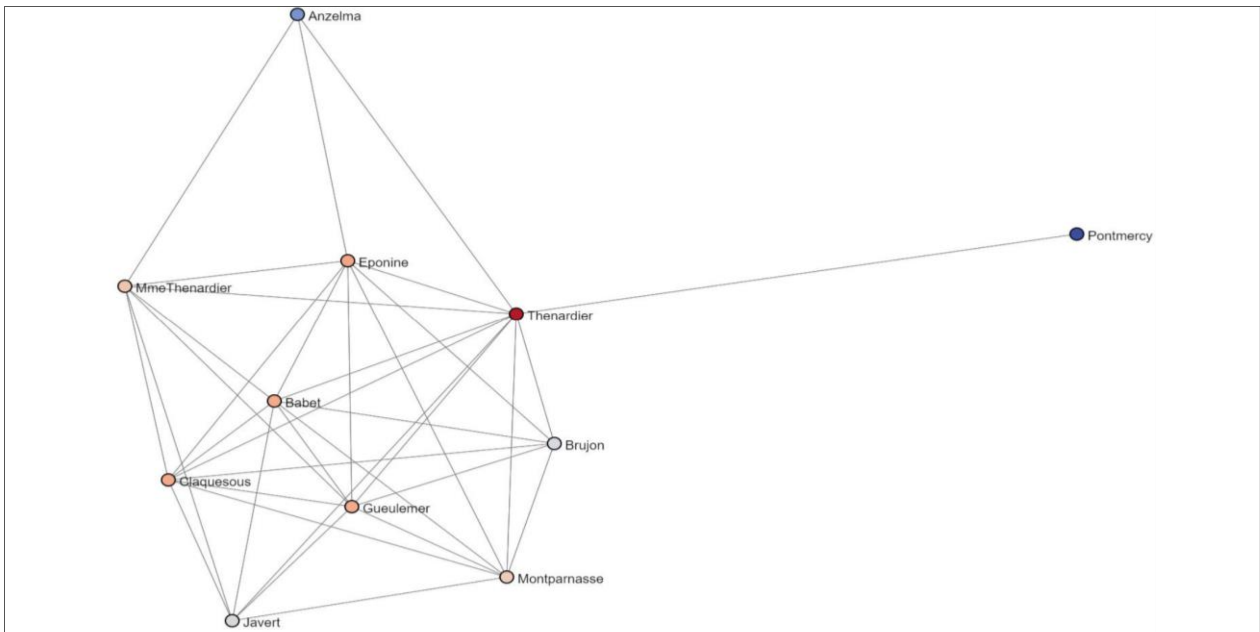

Figure 12: Girvan-Newman community detection of the Les Miserables network, Community 3

Community 2, community 3 is similar to community 1. They have a densely connected group of nodes and some nodes with a single edge. This looks great.

And the next one – the community 4:

Community = communities [4]

G_community = G.subgraph (community)

draw_graph(G_community, show_names = True, node_size = 5)

We have the following network:

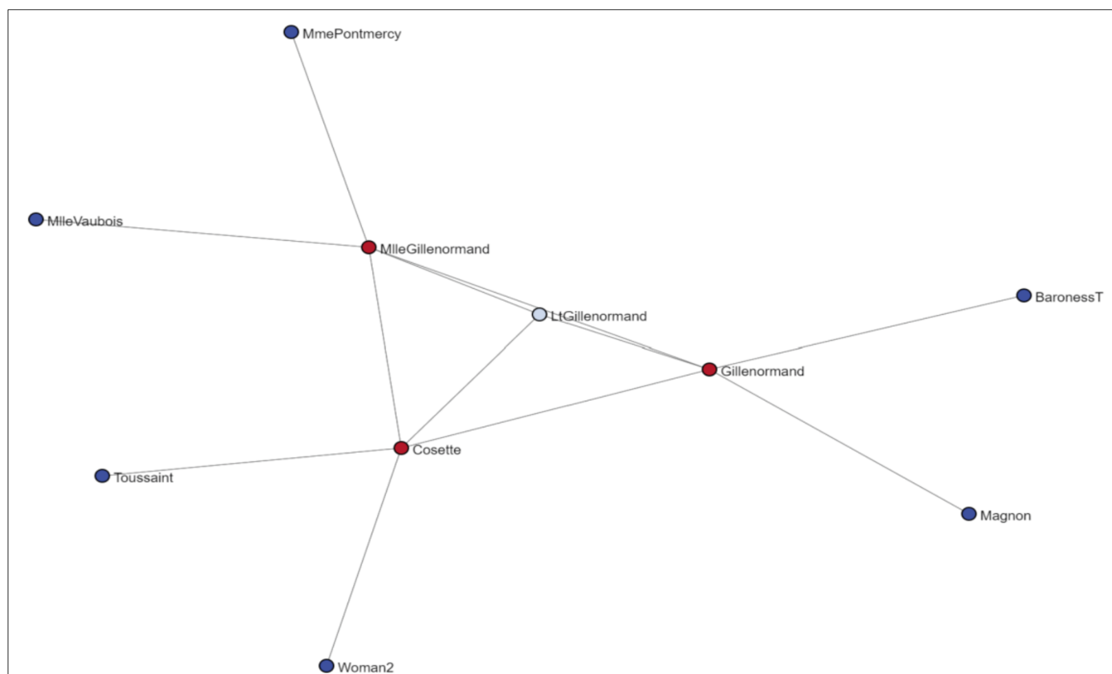

Figure 13: Girvan-Newman community detection of the Les Miserables network,

Community 4

While this may be less visually appealing to look at, this impresses me more than the other network visualizations. This is a less obvious community, found only by cutting edges with the highest edge_betweenness_centrality scores. There is a slightly more connected group of nodes in the center, surrounded by nodes with a single edge each on the outskirts.

The Girvan-Newman method can give really good and clean results. The only downside is its speed. Calculating edge_betweenness_centrality and shortest_paths is time-consuming, so this method is much slower than the others that we discussed, but it can be very useful if your network is not too large.

All these algorithms that we have just explored were ideas that people had on how to identify communities in networks, either based on nearness to other nodes or found by cutting edges. However, these are not the only approaches. If we are continuing with an approach before learning about the Girvan-Newman algorithm that cuts nodes rather than edges. When I studied about the Girvan-Newman approach, I found that to be more ideal and gave up on my implementation. But that got me thinking, what other approaches might there be to better identify communities in networks? Let's try to discover other ways of identifying communities later more.

6. CONCLUSION

We have just studied several different algorithmic approaches for community detection. First, we use the connected components method, it can be useful for identifying communities, but only if the network consists of more than just one single primary component. To use connected components to identify communities, there need to be some smaller connected components split off. It's very important to use connected components at the beginning of your network analysis to get an understanding of the overall structure of your network, but it is less than ideal as a standalone tool for identifying communities. Next, we used the Louvain method. This algorithm is extremely fast and can be useful in networks where there are hundreds of millions of nodes and billions of edges. If your network is very large, this is a useful first approach for community detection. The algorithm is fast, and the results are clean. There is also a parameter you can experiment with to get optimal partitions. Finally, we used the Girvan-Newman algorithm, which is an algorithm that finds communities by performing several rounds of cuts on edges with the highest edge_betweenness_centrality scores. The results were very clean. The downside of this algorithm is that it is very slow and does not scale well to large networks. However, if your network is small, then this is a very useful algorithm for community detection.

Community detection is one of the most interesting areas of network analysis. It is one thing to analyze networks as a whole or explore ego (connections between nodes) networks but being able to identify and extract communities is another skill that lies between whole network analysis and ego-centric network analysis.

Community detection techniques will be helpful in supervised machine learning and unsupervised machine learning research. These techniques would be code-heavy, not math-heavy. There are tons of community detection techniques out there that have an emphasis on math but do not show actual implementation very well, or at all. I hope these methods have effectively bridged the gap, giving a new skill to coders, and showing programmatic ways to take the network analysis to new heights.

REFERENCES

- Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network Flows: Theory, Algorithms, and Applications* (Prentice–Hall, Upper Saddle River, NJ).
- Arenas, A., Fern´andez, A., & G´omez, S. (2008). *N. J. of Phys*. 10 053039.
- Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., ... & Wiener, J. (2000). Graph structure in the web. *Computer networks*, *33*(1-6), 309-320.
- Clauset, A., Newman, M. E., & Moore, C. (2004). Finding community structure in very large networks. *Physical review E*, *70*(6), 066111.
- Danon, L., D´ıaz-Guilera, A., Duch, J., & Arenas, A. (2005). *Community analysis in social networks J. Stat. Mech.* P09008.
- Fortunato, S., & Barth´elemy, M. (2007). *Resolution limit in community detection Proc. Natl. Acad. Sci. USA* 104, 36.
- Girvan, M., & Newman, M. E. J. (2002). *Proc. Natl. Acad. Sci*. USA 99 7821.
- Hoerdt, M., & Magoni, D. (2003, October). Completeness of the internet core topology collected by a fast mapping software. In *Proceedings of the 11th International Conference on Software, Telecommunications and Computer Networks* (pp. 257-261).
- Newman, M. E. (2001). The structure of scientific collaboration networks. *Proceedings of the national academy of sciences*, *98*(2), 404-409.
- Newman, M. E. (2004). Detecting community structure in networks. *The European physical journal B*, *38*, 321-330.
- Newman, M. E. (2004). Fast algorithm for detecting community structure in networks. *Physical review E*, *69*(6), 066133.
- Newman, M. E. (2006). Modularity and community structure in networks. *Proceedings of the national academy of sciences*, *103*(23), 8577-8582.
- Newman, M. E. J., & Girvan, M. (2004). *Phys. Rev,* E 69 026113.

© 2024 | Published by Scholars Middle East Publishers, Dubai, United Arab Emirates

- Newman, M. E. J., Barab´asi, A. L., & Watts, D. J. (2006). *The Structure and Dynamics of Networks* (Princeton University Press, Princeton).
- Palla, G., Der´enyi, I., Farkas, I., & Vicsek, T. (2005). *Uncovering the overlapping community structure of complex networks in nature and society Nature,* 435-814.
- Pons, P., & Latapy, M. (2006). Computing communities in large networks using random walks. *J. Graph Algorithms Appl.*, *10*(2), 191-218.
- Ravasz, E., & Barabási, A. L. (2003). Hierarchical organization in complex networks. *Physical review E*, *67*(2), 026112.
- Scott, J. (2000). *Social Network Analysis: A Handbook* (Sage, London), 2nd Ed.
- Wasserman, S., & Faust, K. (1994). *Social Network Analysis* (Cambridge Univ. Press, Cambridge, U.K.).