Community detection in a hashtags network

Lucia Depaoli Simone Mistrali

March 11, 2022

Contents

1 Introduction

Community detection is crucial when studying social media phenomena. In these situations, in general we look at all the possible contents shared by people, such as posts, tweets, pictures or videos, focusing, for example, on a particular time span, hashtag, keyword or social network. Usually we deal with a large amount of data that we do not know anything about, except that they are somehow correlated to the general topic we are investigating. Community detection gives us information about the relations between the nodes inside the network, in particular could highlight the sub-topics users are talking about, or shows the social impact of a particular event. For example, we want to investigate the sub-topics related to the hashtag #abuse inside Twitter. If, in the time span considered, some scandal comes out, it is probable that we will find a community related to this event, maybe containing the name of the person involved. Or even we could find a community about sexual abuse, and maybe another one about psychological abuse. Although the main topic remains the same, people talk about it using very different vocabulary.

In our particular case, we choose to query tweets containing those particular hashtags: #domesticviolence, #domesticabuse, #coercivecontrol, #abuse, #domesticviolenceawarenessmonth, #trauma, #domesticviolenceawareness, for the time span of: 01/09/2020 to 01/11/2020. Our network contains all the hashtags of the considered tweets except for the hashtags listed above, because they are non-informative: if they are retained in the network we clearly inject a bias in the algorithms given the fact that all the downloaded tweets are linked with those hashtags. At the end, we have a network of 11958 nodes and 85366 edges. What we want to understand is how different community detection algorithms work on our data, comparing the results obtained.

2 Tools

There are a large variety of community detection algorithms, each of them based on different metrics. Most focus on Modularity maximization, Betweeness centrality, hierarchical clustering.

2.1 Modularity

Modularity Q is a measure of the density between intra-community links as compared to inter-community links [1]. In simpler words, it quantify how much the structure of the network is different from a random one.

$$Q = \sum_{c=1}^{n_c} \left[\frac{W_c}{W} - \left(\frac{S_c}{2W} \right)^2 \right] \tag{1}$$

Where W is the total weight of all links in the network, the sum is over the number of the communities detected, S_c is the total strength of the nodes of community c, W_c is the total weight of the internal edges of community c.

An high value of Modularity means that nodes belonging to the same community have high connection between them, but low connection between nodes external to the community. Typically, community detection algorithm returns the community division that maximizes this value.

2.2 Random Walk

Random walks are the most fundamental types of stochastic processes. In this case they can be seen as a "walker" that jumps from nodes to nodes in the network according to a given probability. We can arrange those probability in the so called *transition matrix*, whose elements are computed as follows:

$$P_{ij} = \frac{A_{ij}}{d(i)} \tag{2}$$

Where $d(i)$ is the degree of the node.
Then we can compute the distances between all adjacent nodes according to:

$$r_{ij} = \sqrt{\sum_{k=1}^{n} \frac{\left(P_{ik}^t - P_{jk}^t \right)^2}{d(k)}} \tag{3}$$

3 Algorithms

In our case we have an undirected weighted graph which we do not anything about, so we can leave out algorithms that require the numbers of expected clusters.

3.1 Multilevel algorithm / Louvain algorithm

This is a hierarchical clustering method based on two steps repeated iteratively. In the first step, we start with each nodes belonging to a different community. Then we focus on a single node and we identify his two nearest neighbors. We compute the change in Modularity obtained if the node joins the community of one of the two neighbors, and we select the one with the highest gain (only if it is positive). This is done iteratively for each node until a local maximum is reached. In the second phase, we build a new network formed by the communities found above, considered as the new nodes. The weights between them are the sum of the weights of the edges between them in the previous network. At this point, we go back to the previous step, and the process is done iteratively until a global maximum is reached [3].

In our case, the algorithm returns 330 communities, the first 18 containing more than 90% of the nodes. The biggest community contains 15.17% of the nodes and the second biggest 9.25%. There are 21 communities that have between 10% and 1% of the nodes. The Modularity score is 0.63. In figure 1 is reported the network divided in communities.

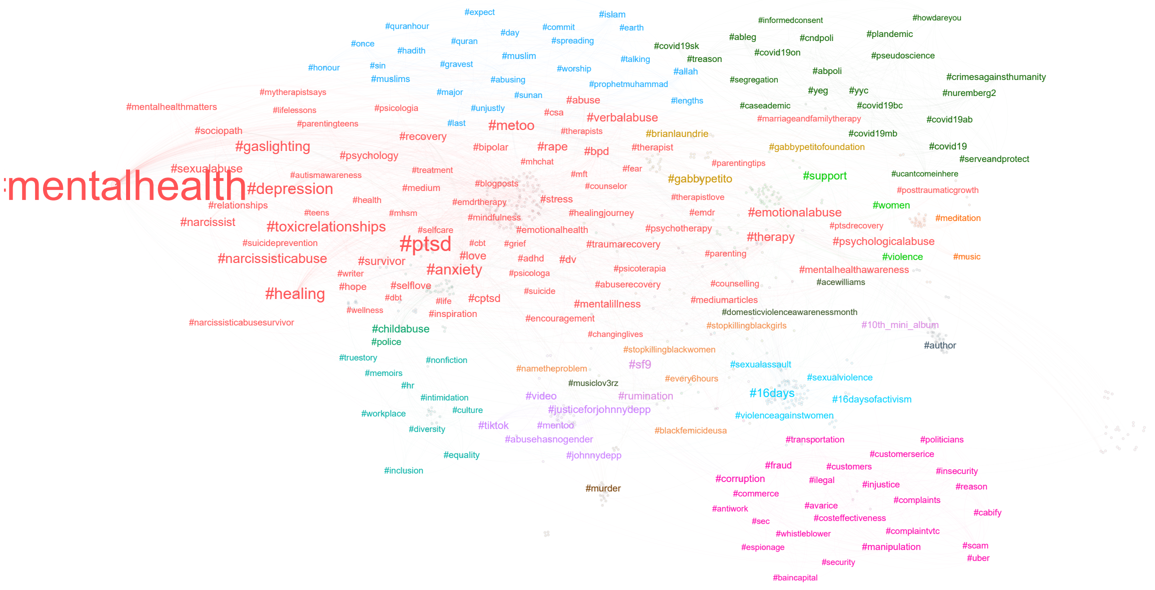

Figure 1: Network divided in communities using Multilevel algorithm.

3.2 Fast greedy algorithm

This algorithm is based on Modularity maximization, and it is an agglomerative hierarchical clustering method. It starts with each node belonging to a different community. At each step, communities are merged together according to the maximization of Modularity. This produces a dendrogram that is possible to cut at the highest Modularity level [2].

In our case, the algorithm returns 352 communities, the first 10 containing more than 90% of the nodes. The biggest community contains 23.88% of the nodes, the second biggest 20.08%, and the third biggest 14.51%. There are 8 communities that have between 10% and 1% of the nodes. The Modularity score is 0.61. In figure 2 is reported the network divided in communities.

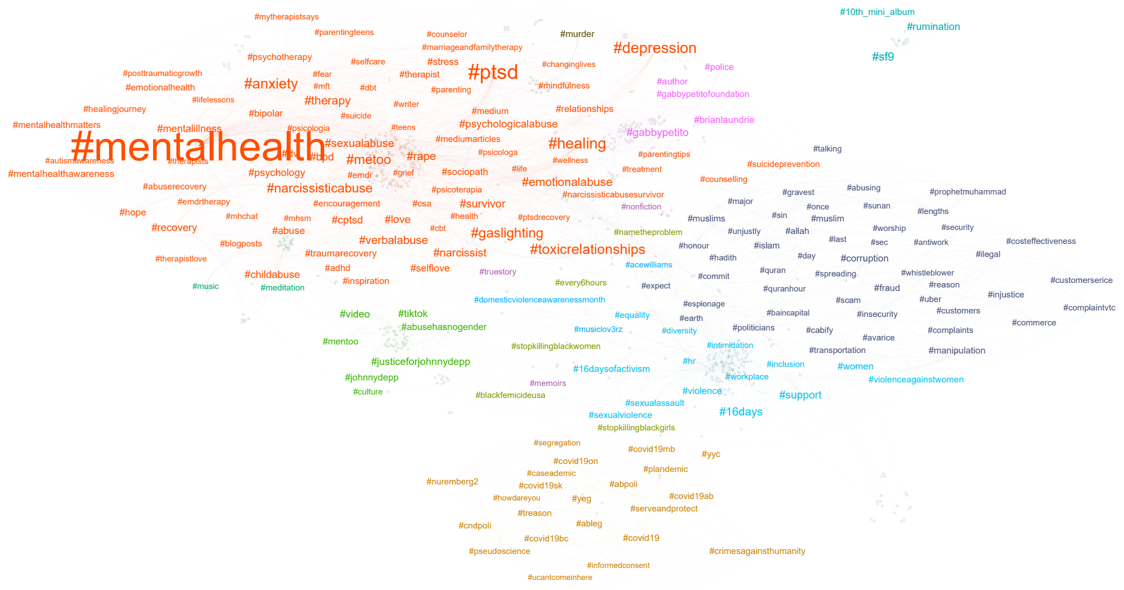

Figure 2: Network divided in communities using Fast greedy algorithm.

3.3 Leiden algorithm

The Leiden algorithm is an improvement of the Louvain algorithm. We can divide it in 3 steps:

- The first phase, as in the Louvain algorithm, is the optimization of the Modularity function.
- The second phase is a refinement of the partitions.
- The third phase is the community aggregation part (as in the Louvain).

The difference between the Louvain and Leiden algorithms is that after the first stage, the refinement of the partitions is performed in each generated small community. Then, the community achieves a local optimum.

In our case, the algorithm returns 10128 communities. The first one has only 0.68% of nodes. Its Modularity score is 0.34. This algorithm has different parameters to tune: probably a grid-search on the parameters could leads us to a highest Modularity score. In figure 3 is reported the network divided in communities.

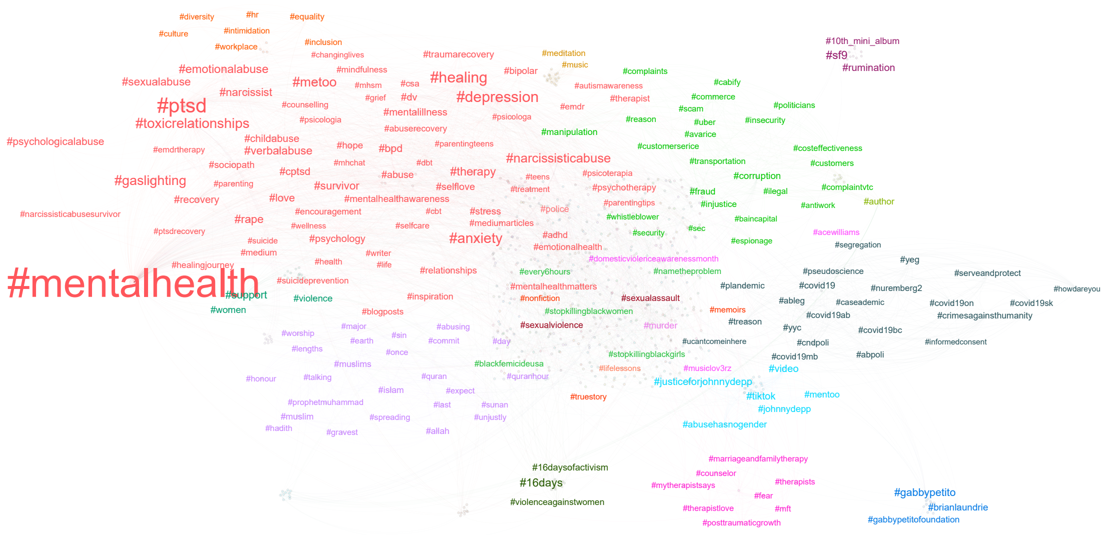

Figure 3: Network divided in communities using Leiden algorithm.

3.4 Eigenvectors algorithm

The idea behind this method is the Modularity matrix $\mathcal{B}$, which is defined as $\mathcal{B} = \mathcal{A} - \mathcal{P}$, $\mathcal{A}$ being the adjacency matrix of an undirected network, and $\mathcal{P}$ being the matrix that contains the probability that a certain edge is present according to the configuration model. To make it easier, a $\mathcal{P}[i, j]$ element of $\mathcal{P}$ is the probability that there is an edge between vertices i and j in a random network in which the degrees of all vertices are the same as in the input graph. The leading eigenvector method works by calculating the eigenvector of the Modularity matrix for the largest positive eigenvalue and then separating vertices into two communities based on the sign of the corresponding element of the eigenvector. If all elements in the eigenvector are of the same sign, that means that the network has no underlying community structure.

In our case, the algorithm returns 308 communities, the first 8 containing more than 90% of the nodes. The biggest nodes contains 24.99% of the nodes and the second biggest 24.81%. There are 7 communities that have between 10% and 1% of the nodes. The Modularity score is 0.51. In figure 4 is reported the network divided in communities.

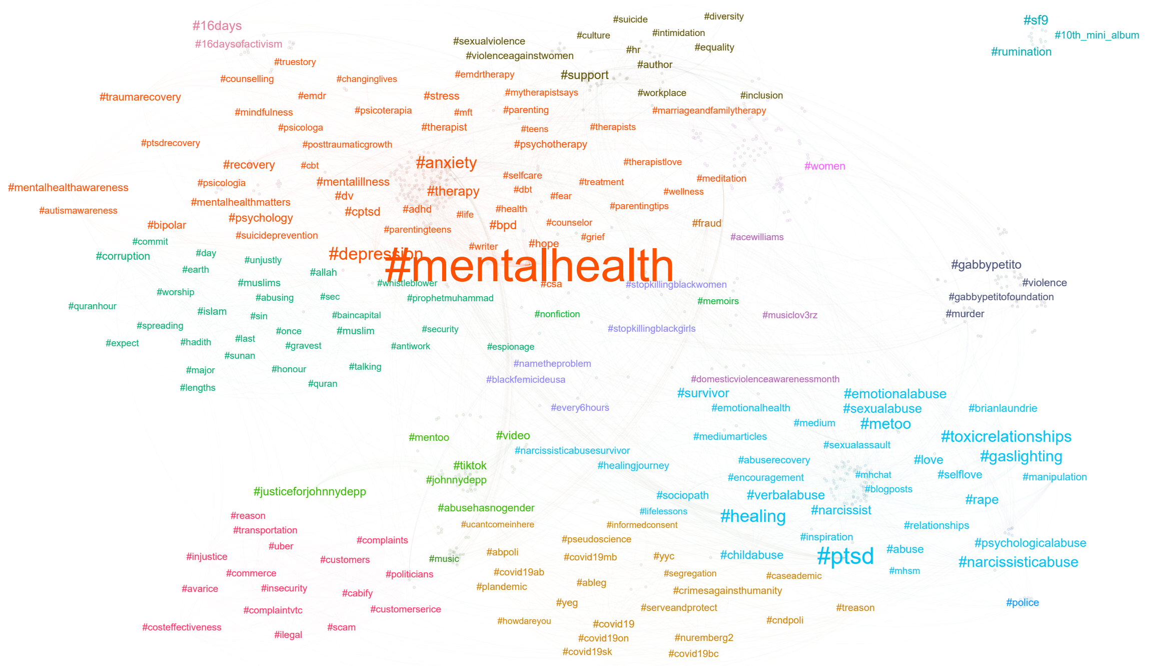

Figure 4: Network divided in communities using Eigenvectors algorithm.

3.5 Info map algorithm

It finds the community structure of the network according to the Infomap method of Martin Rosvall and Carl T. Bergstrom. It is based on Huffman coding. The basic idea is the following: we have a random walker that walks among the nodes and we have to decode the random path of the walker. If no community is considered, each node has a different code and the random path code is very large (no compression in information). If we divide the nodes in groups where each group has a different code, the random path code becomes smaller and we have a compression in information. The algorithm find the best partitioning in order to have the biggest compression in the random path information.

Why having too few communities does not necessarily lead to a minimization in the information? Because a random walker spends more time inside a set of nodes that are somehow correlated. In this way, using the method showed above, we only have the information of when the walker enters and exits the community. If we have only few communities that are not related to each other, the entering-exiting message will be random and we will not have relevant information. This is particularly useful when the connections between nodes represent a flow of a given quantity and not only the similarity between those.

In our case, the algorithm returns 1082 communities, the biggest one containing 4.66% of the nodes, the second biggest 1.33%. All other communities are below 1% for number of nodes. The Modularity score is 0.57. In figure 5 is reported the network divided in communities.

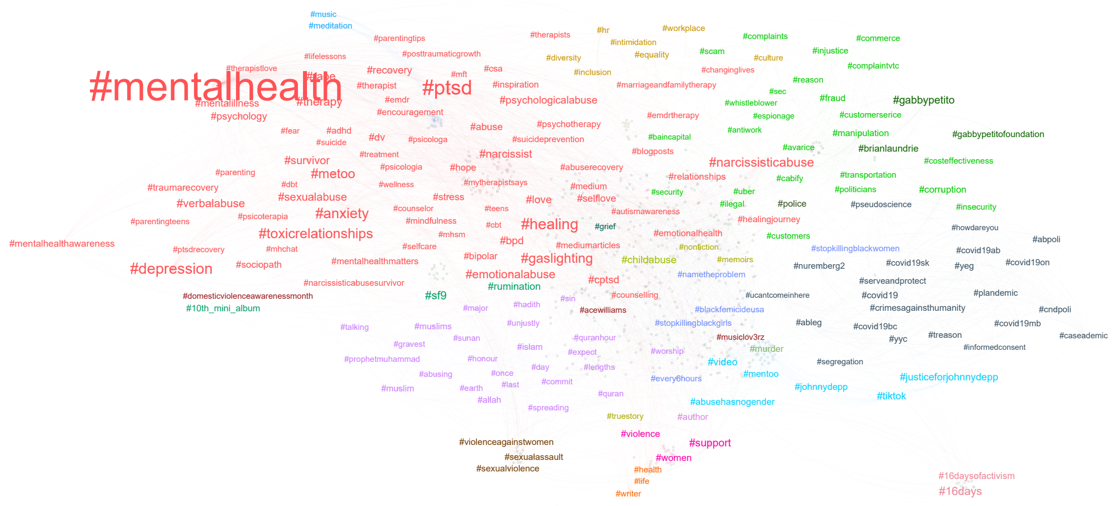

Figure 5: Network divided in communities using Info map algorithm.

3.6 Walktrap algorithm

It is a hierarchical clustering method. At first, each node is a community and we have a partition made by n communities (all the nodes). Then we compute the distances between all adjacent nodes according to equation 3.

At this point, we merge two communities belonging to the same partition that minimize the mean σ_k of the squared distances between each vertex and its community.

$$\sigma_k = \frac{1}{n} \sum_{C \in P_k} \sum_{i \in C} r_{iC}^2 \tag{4}$$

Then we have a new partition identical to the one before, but with the merged communities [4].

In our case, the algorithm returns 2030 communities. The biggest community contains 19.25% of the nodes and the second biggest 12.15%. There are 2 communities that have between 10% and 1% of nodes, all others have less than 1% of nodes. The Modularity score is 0.53. In figure 6 is reported the network divided in communities.

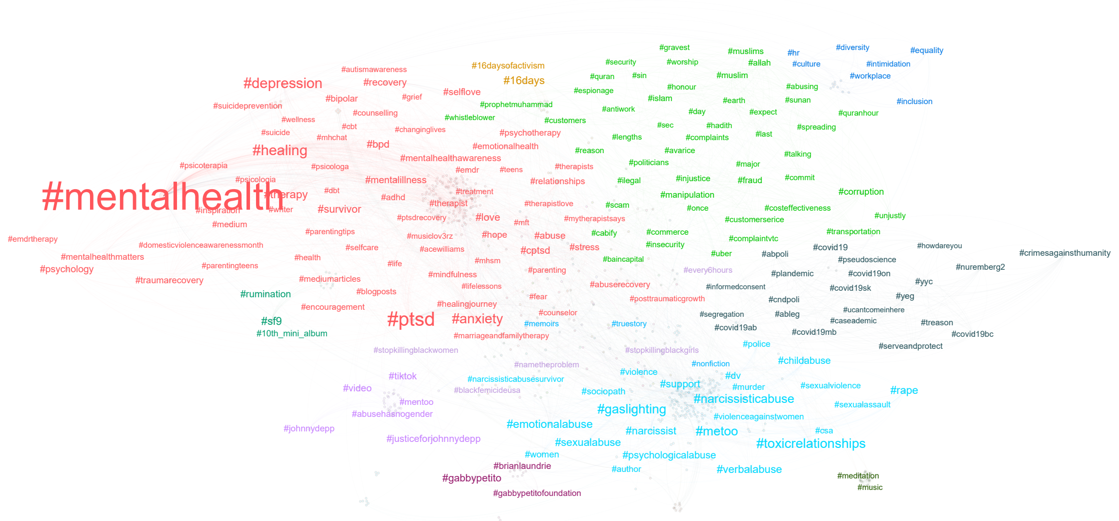

Figure 6: Network divided in communities using Walktrap algorithm.

4 Results

All the graphs above show the same nodes and the same edges, the only difference between them is the division in communities. In order to allow a better visualization, the networks are filtered by edge weights (> 5) and node weights (> 100), the latter applied only for the labels. The algorithm that produces the highest Modularity score is Multilevel (which is an improvement of the Louvain algorithm), and the one that produces the lowest one is Leiden. Given a network, the Modularity score depends only on the partitioning of the nodes, so it is right to compare the score of different algorithms, but not every algorithm is based on Modularity maximization, even if all the dendrograms cuts are done at the highest score. Because of this, we can compare the results in this way, but this should not be the only parameter considered: maybe the algorithm with the highest score tends to merge together different communities (and this could fit a user-type analysis), whereas we are interested in identify the sub-topics.

The most relevant difference between the outputs of the algorithms is that the number of communities is around 300 for Multilevel, Eigenvectors and Fastgreedy, around $1000 - 2000$ for Infomap and Walktrap (the only two algorithms based on random walk), and around 10000 for the Leiden algorithm. These lasts three algorithms tend to produce hundreds and thousands number of communities with $2 - 3$ nodes inside, while the others place these nodes in bigger communities. For example, for the Leiden algorithm, the produced graph reported in figure 3 is similar to the others, but it leaves alone a large amount of "small" nodes outside the main communities: we can see that from the cloud of grey small dots in the center of the graph.

In any case, we have the same largest community, which contains the hashtags #mentalhealth, #ptsd (not for Eigenvectors), #healing (not for Eigenvectors), #anxiety, #psychology, #psychotherapy, #health, #recovery, #traumarecovery, #therapy, #stress, #mentalhealthawareness, #mentalillness and so on. Eigenvectors algorithm places in the same community the hashtags #ptsd and #healing, together with hashtags #toxicrelationship, #gaslighting, #metoo, #emotionalabuse, #sexualabuse, #narcissisticabuse. For the others algorithm, all these hashtags are in the biggest community, except for Walktrap which puts them separated from the others. There are some communities that are always the same for every algorithms: for example the #rumination community, the #johnnydepp community and the #covid19 community. Two other large communities are: the first one formed by main hashtags #muslim, #allah, #abusing, #islam, and the second one formed by main hashtags #uber, #insecurity, #commerce, #cabify, #scam, #injustice. These two communities are separated for every algorithms except for Walktrap and Fastgreedy, which merge them. The hashtags #16days and #16daysofactivism are either alone or together with hashtags #violenceagainstwomen, #sexualassault, #sexualviolence. A part from Eigenvectors algorithm, hashtags #support, #women and #violence are always together, whether alone or merged with the previous two set of hashtags.

We can conclude that all the algorithms identify the same "topic" communities, but sometimes they place them in a macro-community or sometimes they merge them together (for example #women, #16days and #sexualassault communities, as said before). The merge process is typical of the hierarchical clustering methods. It is important to remember that, for every of these methods, we cut the dendrogram at some suggested level (the one with the highest Modularity score): it is obvious that cutting at a lowest level could separate the merged communities. Anyway, even if some algorithms identify thousands of communities, the biggest ones remains the same. This is an important result because it stated that, even if we have some very different metrics, the main detected topics remain the same.

About the computational time of the community detection, it is obvious that it is possible to find a partition that gives the best Modularity score possible, but it would require a lot of computational effort, especially for a large network. All the algorithms showed above exist in order to reduce the time complexity. In our case, every one of the previous algorithm requires only some seconds to compute, so the time complexity of our algorithms was not taken in consideration.

What is obvious from our results is that there exists an overlapping of communities. This is clear because sometimes important hashtags (such as #ptsd or #emotionalabuse, #toxicrelationship) belong to different well-defined communities. There exist various algorithms to investigate the overlapping community detection, but the results are difficult to visualize (an interactive visualization would be better) and it is necessary to adapt the Modularity function to this problem.

References

[1] Bisma S. Khan, Muaz A. Niazi (2017). Network Community Detection: A Review and Visual Survey. Available at: https://arxiv.org/ftp/arxiv/papers/1708/1708.00977.pdf

[2] M. E. J. Newman (2003). Fast algorithm for detecting community structure in networks. Available at: https://arxiv.org/pdf/cond-mat/0309508.pdf

[3] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, Etienne Lefebvre. Fast unfolding of communities in large networks. Available at: https://iopscience.iop.org/article/10.1088/1742-5468/2008/10/P10008/pdf

[4] Pascal Pons, Matthieu Latapy (2005). Computing communities in large networks using random walks. Available at: https://arxiv.org/abs/physics/0512106