Community Detection on a sample of articles.

Carlos García Sánchez
Università di Bologna, Complex Networks Course
(Dated: September 20, 2024)

Abstract: In this project we have used the software MATLAB to reproduced the results of M. E. J. Newman and M. Girvan's article about community detection [1] (concretely Zachary's Karate Club example) and then applied the program with some modifications (made it valid for weighted networks and faster for bigger networks) to a network of words created with a sample of abstracts obtained from the web page database www.scopus.com.

I. INTRODUCTION

There are a great variety of community detection articles in the vast Complex Networks' bibliography. In this work we focused on one of the first articles, it is the one of M. E. J. Newman and M. Girvan [1] published on 2004. In this article they introduced a new way of thinking about connections between distant vertices: *betweenness*. *Betweenness* is a measure that is based on the fact that through the edges between communities there will be more paths that connect vertices belonging to different communities than edges that are in the communities themselves. So *betweenness* is some kind of measure that favors those edges between communities and disfavors those that lie within communities. They also introduced a definition of *modularity* based on an earlier measure of assortative mixing proposed by Newman itself on a preceding article ([2]).

In the following sections it is presented an implementation of the results showed in [1] via MATLAB. Concretely the created code follows the *short-path betwenneess* method (as indicated in [1] it yields the best results in lesser time) and it is applied to the example B of section V in [1]. Then we will present the procedure carried out to obtain a weighted adjacency matrix of a network of words between abstracts of articles along with the modifications we made in order to make a faster algorithm from Newman and Girvan's one. Finally, in the last subsection we will apply our code to detect communities in the network of the chosen articles and discuss the results. It is important to say that our code is thought to work on undirected networks (weighted or unweighted) and its proper functioning in directed networks has not been tested. The GitHub repository [3] contains all necessary documents to run the code on MATLAB. Also in [4] you can find the data obtained through [5] used for this work.

II. IMPLEMENTING NEWMAN AND GIRVAN'S ARTICLE

The structure of the algorithm is easy to follow. As mentioned before it is based on *betwenness*. The idea is to remove those edges with greater *betweenness* until all edges have been erased. This makes this algorithm to be a divisive algorithm such that we don't start from the vertices alone but from the giant component of the network. The steps to follow are presented in [1] and they are the following:

1. Calculate betweenness scores for all edges in the network.
2. Find the edge with the highest score and remove it from the network.
3. Recalculate betweenness for all remaining edges.
4. Repeat from step 2.

There is an important feature in this algorithm and it is step 3; the recalculation of betweenness after each edge removal. It is a step that adds complexity to the algorithm (we will see in the next subsection what's its repercussion on the computational time order) but its effect on the results makes it a worth paying step. This happens because the betweenness of the network changes when one edge is removed.

This is really important because if we don't recalculate, the algorithm's accuracy drops drastically. Imagine that two communities in a network are connected via two edges and for whatever reason one of the edges presents a greater betweenness than the other. Then an algorithm without the recalculation step would remove first the one with greater betweenness and leave the other. This would mean that the algorithm wouldn't be able to detect that division and so it will skip it resulting on a meaningless set of divisions of the network.

A. Short-Path Betweenness method

To compute the betweenness it is necessary to choose the type of betweenness one wants to use (random-walk betweenness or short-path betwennnes for expample).As said before the implementation has been carried out using the *Short-Path Betwenness* method. This method, as its name infers, is based on short-paths. A short-path between two vertices is the path defined as those edges through which one passes if arriving crossing the minimum number of edges is one's objective.

Now, with that definition in mind and following [1] we have created a MATLAB function that first, using a breadth-first search (see pgs 311 to 316 in [6]), computes for a starting vertex "s" the short-path distance to all other vertices in the network. For each neighbor i of vertex s we assign a distance of $d_i = 1$ then we continue assigning a $d_j = d_i + 1$ distance for each neighbor j of each vertex i only if d_j has not yet been set. In this first part of the function is obtained an array of length equal to the total number of nodes whose entries are the corresponding distance (i.e. the number of edges passed through) to the network's vertices. The element 1 will correspond to the distance from s to the node 1, element 2 to the distance from s to node 2 and so on. Of course distance of node s is set to be 0.

Also in the previous breadth-first search we have to add some weights to each vertex. This is the part necessary to obtain betweenness; we will need those weights to compute the short-path betweenness in the following steps. For vertex s and each neighbor of vertex s we assign a weight equal to $w_i = w_s = 1$. Then, as the breadth-first search executes, if we have i and its neighbor j if j has already a distance assigned and $d_j = d_i + 1$ then we set $w_j = w_i + 1$, in the case where j doesn't have a distance assigned we make $w_j = w_i$. Given a weighted adjacency matrix we must set $w_j = w_i/A_{ij}$ [1]. In practice, we can set $w_j = w_i/A_{ij}$ given a weighted or an unweighted matrix because the entries of an unweighted adjacency matrix are ones when there's an edge between two connected vertices. On the other hand, if j's distance doesn't satisfies neither of the previous we do nothing. The idea is to obtain how many times each path passes through each vertex (without counting the times it passes through s that would be equal to n = number of vertices). Now having the distance array and the weights array one can compute the betweenness for each edge.

The betweenness will be stored on a matrix of length equal to the adjacency matrix's length, i.e. equal to n = number of vertices, and on each entry non-zero valued we will have the betweenness' value corresponding to the related edge.

Now, in order to compute it we will have to find the "leaf" vertices we will reefer to by t's (their weight will be w_t). These are the vertices such that no paths from s to other vertices go through them. We find them during the breadth-first search by saving the position of those vertices for whom the weight array stays the same in the iteration. Also they are obtained only for the last sweep of the reading index (i.e. the last sweep before the reading index becomes equal to the writing index). Leafs only have neighbors for whom we have already set a distance that is smaller and belong to the last breadth-first search writting index iteration i.e. they are the leafs of a tree if we see paths as the branches of said tree.

We will work from the leafs towards the trunk (vertex s). The first desirable step is to start setting a betweenness value for the edges that connect the leafs with their neighbors. We do that by setting w_i/w_t as the betweenness value of the mentioned edges. From this we can compute the betweenness of the rest of the edges.

The betweenness of the remaining edges is calculated by a double "while" loop (although it could be implemented also via a double "for" loop). The array we traverse via the first loop is composed by all the vertices (except for the leaf vertices) sorted in decreasing order. Now, for each remaining vertex i we obtain his "below neighbors" and his "above neighbors" i.e. neighbors with distance equal to $d_i + 1$ and neighbors with distance $d_i - 1$

[1] This case is not treated in [1] but we have supposed that the more connected two articles are the shorter the path between them is.

respectively (only if they are in the array we are traversing). Then we apply the next formula to compute the betweenness of each "above edge" with the second while loop (because we already have the one corresponding to each "below edge"):

$$B_{ij} = B_{ji} = (\sum_{k \in \downarrow Neighbor} B_{ik} + 1)\frac{w_j}{w_i} \quad (1)$$

Where B stands for the betweenness matrix whose entries represent the betweenness value of edges. i stands for the vertex for which we are computing his above edge betweenness and j is the above vertex. On the other hand, the sum over k is made over all below neighbors. Of course w_i and w_j represent the i and j weights respectively.

Note that we have included all this method in a function called *ShortPathBetw* and we will use it on a father function that implements the community detection algorithm we denoted by *DendrogramAndQComp* (which is in turn a daughter function of *CommunityDet*). An scheme showing the structure of our MATLAB program is presented in Figure 1.

At this point we have created a program to compute the short-path betweenness of each edge being s the starting point. This yields a complexity of order $O(m)$ where m is the number of edges. As we need to perform this for each vertex of the network and then perform the recalculation step for each edge's removal we obtain a final complexity of $O(m^2 n)$. This is a high order complexity and it will be the most important disadvantage of Newman and Girvan's algorithm. Nowadays there exist new and faster ways to realise community detection such as the ones mentioned in [7]. Also Newman published in the same year another article with a faster algorithm, see [8]. As we said in the beginning we are just interested on implementing the results of [1].

B. Modularity and Dendrogram

At the moment we have the way to measure betweenness and the idea is to remove the edge with the greatest betweenness on each iteration. The key to continue on implementing Newman and Girvan's algorithm is to be aware that for every edge removal we don't necessary have the network divided on different communities. How do we know when we have two unconnected components on our network? easy, using the short-path method described in the previous section. If we take as an output of *ShortPathBetw(s,A)* the array containing the distance to every vertex and find out if there are vertices to which there is no short-path then we have identified a community. As the short-path is computed from an initial vertex s, then all those vertices that present a distance belong to the same component as s and the rest of the vertices to other components.

In order to implement the previous we create an array "q" of length n and made out of -1's that will assign a number to each vertex depending on the community they belong to. For every edge removal we find out if there is a network division (computing distances of, for example, vertex number 1) and if it is the case then we assign in q a number $k = 1$ to all the vertices that have a distance. Next, we increase k in 1 unit and move in q to the first vertex that has no community and perform the same until all vertices have a number assigned. Then we save q on a cell we will call "*Dendrogram*" and continue till every edge of the network has been removed. We call it dendrogram because this cell will contain the divisions of the network from the giant component to each vertex alone and a great way to see this divisions is via a dendrogram (see e.g. figure 6 of [1]).

At the end of the removals we obtain the cell *Dendrogram* but our objective is to detect communities. A set of the network's divisions doesn't help us much on finding the network's division into its natural communities. At this point Newman and Girvan introduced the concept of "*modularity*" (section IV in [1]).

Modularity is a meassure of how much close is a division of a network to be a random division. If its value is close to 0 then the division is near to be random and if it is close to 1 then the division is far from being random. Obviously our objective is to find the divisions with high modularity to find a good community structure of the network.

In mathematical terms the modularity is defined via a symmetric matrix called $\mathbf{e}$ which is a square matrix $k \times k$ where $k = \#\ of\ communities$ and whose elements are the total fraction of edges shared between communities. Also its diagonal terms represent the fraction of edges within communities (e_{ii} = total fraction of edges within community i). Now, having that in mind, one could say that the trace of $\mathbf{e}$ is a good measure of modularity because the more connections within communities the more strongly connected they are and therefore the better the partitioning is. But that doesn't work well because if we have only one community then $Tr\ \mathbf{e} = 1$ and every network will have maximum modularity when the giant component is the only one community. So in order to establish a good measure of modularity we define $a_i = \sum_j e_{ij}$. In a network in which edges fall between vertices without regarding the communities they belong to, we would have $e_{ij} = a_i a_j$ and this gives us the ability to define modularity as described in the previous paragraph using $a_i a_j = \parallel \mathbf{e}^2 \parallel$ [2]

$$Q = Tr\ \mathbf{e} - \parallel \mathbf{e}^2 \parallel \quad (2)$$

Now that we have a way to test the goodness of a division we only need to compute it for all divisions on

[2] $\parallel \mathbf{e}^2 \parallel$ reefers to the sum over all elements of $\mathbf{e}^2$ i.e. $\sum_{ij} \sum_k e_{ik} e_{kj}$

the Dendrogram and see the peaks. In theory the better division is the one of maximum modularity but it is interesting to see also other peaks that might appear because they might represent also good community divisions. For networks with strong community structure a normal range in modularity is from 0.3 to 0.7.

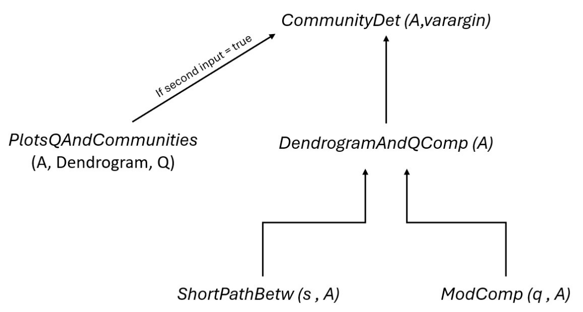

FIG. 1. Tree structure of our community detection MATLAB program. The inputs of each function are in brackets; A is the adjacency matrix, Dendrogram is the cell containing the different divisions obtained in the algorithm. Q is the array containing the modularity value for each division in the dendrogram and q in *ModComp* represents a partitioning of the network. *PlotsQAndCommunities* create the desired figures to analyse the results of the algorithm. All functions are uploaded into the GitHub repository [3].

In the following subsection we will show the obtained results applying our MATLAB code to the Zachary's Club network.

C. Applying to Zachary's Club network

In order to put in application the created code we try to reproduce results of [1] in the case of Zachary's Club network (see section V example B). We only need its adjacency matrix which we obtained from [9]. For a network of this size the algorithm doesn't take much time (around 4 seconds on a computer bought in 2018) and the results are in great concordance with those presented in [1].

Seeing figure 2 and comparing with the same plot in [1] (see figure 9 in [1]) we notice (apart from realizing how equivalent they are to each other) that there are two pronounced peaks. This means that, as we have already said, we have two good partitions of the network. The one of maximum modularity corresponds to a partitioning of the network into 5 groups (one of them composed only of one node see figure 3) and our results (members of each group) are the same as in [1] (again see figure 9 in [1]). The interpretation of this result could be that the network presents an underlying community structure with more groups than 2 (which is the real community structure and was the way Zachary's karate club split) but as we can't relate this structure with anything in our social network (at least at first sight) we stick to the peak with less modularity.

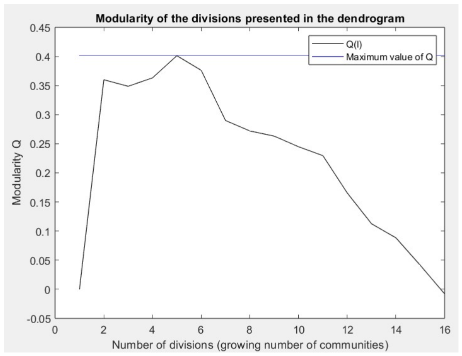

FIG. 2. Modularity values for the different partitioning of Zachary's Club network.

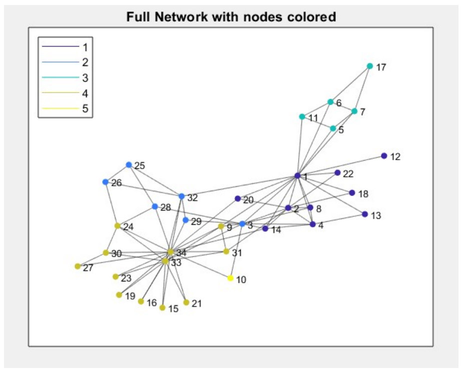

FIG. 3. Zachary's Club network with nodes colored in function of the community they belong to. Here is a representation corresponding to the maximum peak in figure 2.

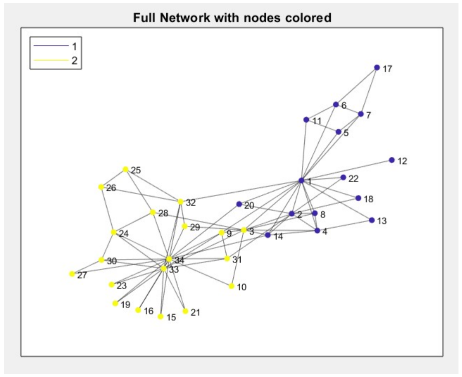

FIG. 4. Zachary's Club network with nodes colored in function of the community they belong to. Here is a representation corresponding to the 2 groups peak in figure 2.

Now for the other peak we have some good results (see figure 4). The division coincides with the real case of Zachary's Karate Club in all members except member 3 that is wrong placed. This was a promising result at the time Newman and Girvan's article was written but as we said previously the algorithm has a high demand on operational time when we go to bigger networks. We will keep that in mind for the next section. However, it wasn't long before Newman published a new article designing a newly algorithm based on modularity optimisation (basically chasing bigger modularity values on each iteration, leaving the idea of erasing edges and embracing an agglomerative approach, see [8]).

III. APPLYING THE MODIFIED ALGORITHM TO A NETWORK OF WORDS BETWEEN ARTICLES

The previous algorithm works for any network if we have its adjacency matrix (weighted or unweighted, directed networks have not being tested). As already mentioned the time demanded for this algorithm grows rapidly with the number of nodes and edges (for a sparse network complexity is $O(n^3)$). Having that in mind we have modified the algorithm in order to obtain meaningful results for larger networks in a reasonable computation time. To do that we have added in *ShortPathBetw(s,A)* function, at the beginning, an "if" condition that acts when the network has 100 or more than 100 nodes and that picks up randomly from the network a group of nodes of size $10^{[ceil(o(n)/2)]}$ [3] where $o(n)$ is the order of n. Then the connections of the rest of the nodes are set to 0 and it is applied the short-path betwenness algorithm. The idea behind this modification is that for a network this size calculating the betweenness for a group of edges on each iteration instead of doing it for the whole network can work as well in some cases. Also to save more computational time instead of removing just one edge after calculating the betweenness we modified again the code, this time in *DendrogramAndQComp(A)* function, to remove $round\left(length(\vec{B}_{max})/10^{[round(o(length(\vec{B}_{max}))/2)]}\right)$ edges on each step [4], where $\vec{B}_{max}$ is an array with the nonzero valued maximum edges on each column of matrix B (see equation 1). With that we loose some of the algorithm's power of community detection but we save a lot of computational time.

Before presenting our results we are going to explain how did we manage to obtain a network of words from a sample of abstracts (i.e. to obtain its adjacency matrix).

A. Obtaining the adjacency matrix

We have started with a sample of 20000 articles (and some books but mostly articles) from 2008 to 2015 acquired via the Scopus database writing "Black holes" in the searcher (see [5]). The idea was to identify communities within black holes articles using connections between their abstracts (i.e. coincidence of words between abstracts). Now let's say up front that our matrix creation function [5] has a complexity of order $O(n^2l^2)$ where l is the maximum number of words of the articles. This forces us to select a smaller amount of articles with less words.

The name of our MATLAB function is *SAAMC* (Scientific Abstracts Adjacency Matrix Creation) and has as an input a .csv document with all the data (in our case: [4]).

The first filter is made while splitting every word of the abstracts. We erase all abstracts with more than 400 words as those abstracts are extremely large. Then, to focus on meaningful connections we throw away basic words such as: "the", "we", "they", "each", etc. (in practice we set them to a string equal to $''0''$). We obtain for each abstract an array made out of meaningful words, in theory, related to black holes. Of course, there might be repeated words, using *unique()* MATLAB function we get rid of them (see SAAMC function in [3] for more details). Next, we pick up the maximum number of words N in our modified sample of abstracts to throw away useless $''0''$ values on each article.

At this point and with the number of words saved in an array, we created an histogram to have a good criterion of what articles we should keep and what articles we should throw away.

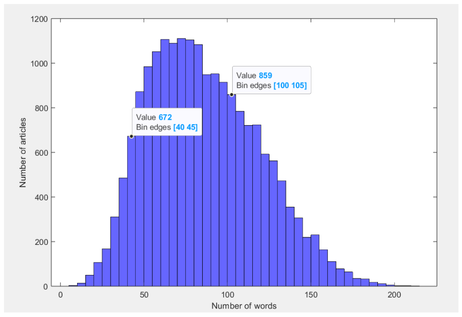

FIG. 5. Histogram of number of articles vs number of words. The articles with a number of words between 45 and 105 are the ones we keep.

As one can see in figure 5 the number of words and the number of articles follow a Gaussian distribution centered more or less in 75 words. Thinking further, the articles we should save are the ones with sufficient words to explain the work they talk about but not with a big amount of words because the abstract is a summary.

With that criteria we select the articles within 45 and 105 words in their abstracts and obtained a set of 12234

[3] *ceil(X)* is a MATLAB inbuilt function that rounds each element of X to the nearest integer greater than or equal to that element.

[4] *round(X)* is a MATLAB inbuilt function that rounds each element of X to the nearest integer.

[5] It is a daughter function of our main function SAAMC but the time is mostly spent on it rather than the rest of SAAMC.

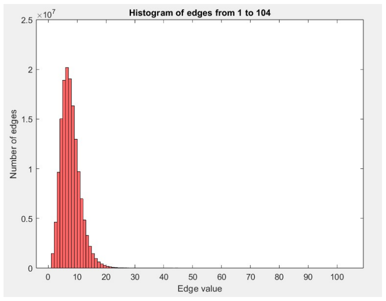

FIG. 6. Histogram of number of edges versus their value. This corresponds to the network with 12234 nodes.

articles. Having the selected abstracts we used *AdjacencyMatrixCreation(AbstrS)* (see [3]) which creates a weighted adjacency matrix (in int8 format to save computational space) counting the number of coincidences between two abstracts i.e. each edge (entry of the matrix) represents how strongly related two articles are.

However, we still need to filter the articles because now we have a network of 12234 nodes with almost 100 million of edges, clearly an amount unprocessable by our algorithm in a human's lifetime. So we need to turn once again to histograms.

The first histogram we need is the one of the number of edges versus their value. As one can see in figure 6 there is a peak between 1 connection and 15 connections and almost all of the connections lie in that range. This is because of the basic words that in our first filter we haven't been able to detect and erase. This is not a problem and to see some structure we need to go to a higher edge value.

For that we represent only the histogram of values between 50 and 104, that is the maximum value [6], and see what we get. The result is showed in figure 7 where one is able to see some structure. There's a pronounced peak between 80 and 81 and some minor peaks at other values. This will be enough for us to get a valid network to which apply our algorithm.

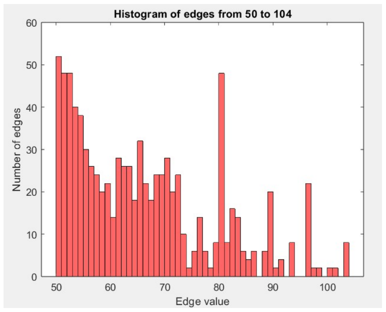

FIG. 7. Histogram of number of edges versus their value between 50 and 104.

[6] Probably because there are some articles repeated and generated by the system of downloading data in Scopus web page.

Now the idea is to keep the articles with connections greater than 50 but without throwing away their connections smaller than 50, as they are important for the network's structure [7]. The nodes that are left (see *BHAdjMatrix.m* in [3] for more details) constitute a network of 643 nodes so we have removed from the original network 11591 nodes. With the previous done, we create another histogram to see from where to keep the connections.

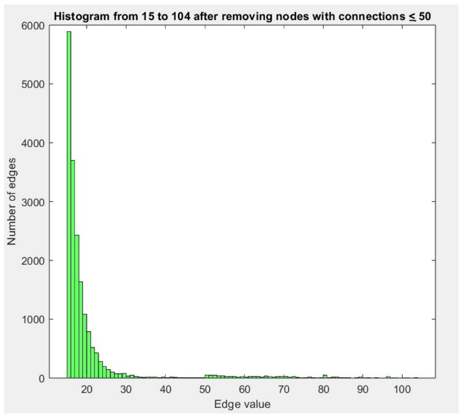

FIG. 8. Histogram of number of edges versus their value from 15 to 104 after removal of nodes whose maximum connection is smaller or equal to 50.

We see in figure 8 that we have the peak corresponding to basic words connections but, as we have removed a great amount of nodes with connections equal or smaller than 50, the number of edges is significantly smaller than in figure 6. At 50 we see the number of edges increases a bit. This correspond to the connections of the 643 nodes we selected in the previous step. The idea now is to erase those edges that don't carry any important information. The procedure followed at this point was erasing edges smaller than a certain value and plot the network to see if we got something interesting. In that way we arrived to the network showed in figure 9 that represents the network of 643 nodes with edges < 18 removed.

This network presents an important number of unconnected components but at bottom left of the figure we can see that the large component has some interesting structure and doesn't seem to be random.

We will keep that component and study its structure. In order to isolate and keep the large component we selected a node from it and, via the *ShortPathBetw(s,A)* function, we obtained the distance to all remaining nodes. Then, throwing away the nodes without distance (distance equal to -1), we got the large component that will be our studied weighted network.

[7] Not all of them are important because most of them will be the ones corresponding to basic words connections (notice the peak at figure 8). However, we will get rid of these unimportant edges in the following step.

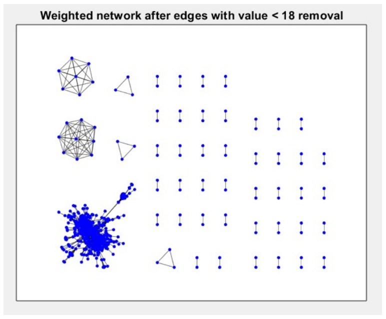

FIG. 9. Network obtained after removal of edges smaller than 18. One can see unconnected components and a large component at bottom left of the picture. The later will be the weighted network we will study.

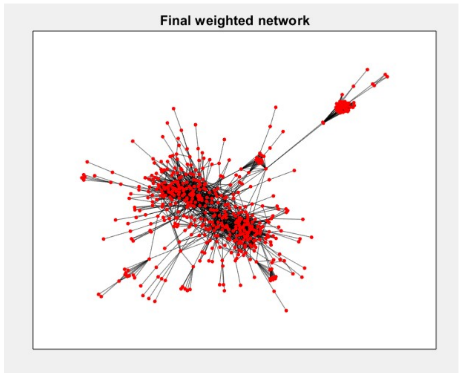

FIG. 10. Final weighted network. This is the isolated large component of network represented in figure 9.

With all the previous (see script *BHAdjMatrix.m* in [3] for more details on the proceeding followed) we have arrived to a weighted adjacency matrix that represents the connections between our group of articles, see figure 10.

B. Results for the sample of abstracts

In this subsection we show the results of applying our community detection program to the obtained adjacency matrix made out of 522 nodes. The exact time spent was 37 minutes and 44 seconds. It is not a large amount of time and we have obtained some interesting results (without modifications mentioned at the begining of section III the computational time would have been enormous).

The algorithm has shown maximum modularity for a partitioning in 321 groups. The maximum modularity reached is 0.2696 which is considerably near to the range Newman and Girvan gave in [2] for a network well divided (their range was from 0.3 to 0.7). In figure 11 we show the several iterations and their corresponding modularity. As one can see modularity slowly increases and then reaching the 113th iteration it suddenly hits its maximum value from a value of 0.059 in modularity. The rapid increasing

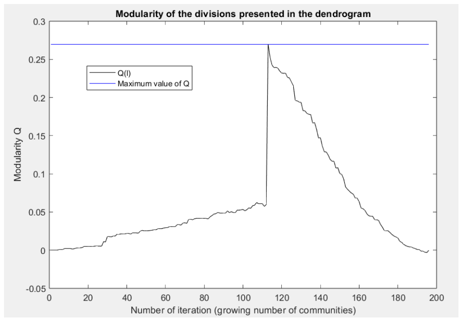

FIG. 11. Modularity for our sample of articles.

is due to the random picking explained in the beginning of section III. At iteration 113 the removal correctly erase a good amount of inter community edges as the previous random picking selects them isolating the network into at least two true communities, probably the red and orange communities showed in figure 12. From that point on, modularity continues to drop until it reaches 0.

The later is the main disadvantage of the added modifications because they are a bit random but still they are worthwhile because they allow us to use our algorithm and get meaningful results in a short amount of time. However, we have to keep that in mind when using this algorithm.

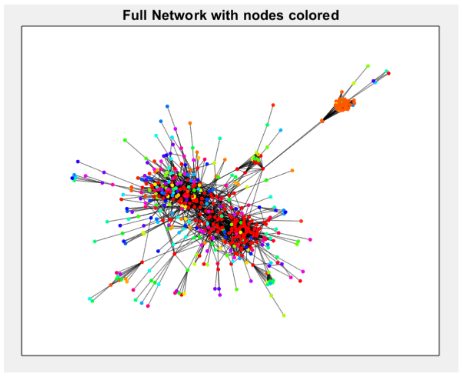

FIG. 12. Plot of the best division into communities of our network, this time we represent the network with the nodes colored depending on the community they belong to.

Also we have obtained a plot of the partitioning corresponding to the maximum of the modularity. The edges are broader if the connection is more important and narrower if the contrary. Also the more edges within the community the bigger is the node that represent it. We show the result in figure 13. As one can see there are two main communities, red and orange, already mentioned above.

If we go to our data base indeed red and orange communities has very little in common excepting the words “black” and “hole”.

Orange community are articles related to cybersecurity. It encloses 47 articles about **MANETs (Mobile**

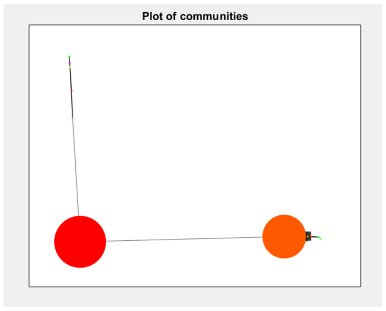

FIG. 13. Plot of the best division into communities of our network of words between articles.

Ad hoc Networks) and “black hole attacks” which is a hacking technique that consists on making a node (e.g. a router) in a connected network to delete all the information it should keep acting as a kind of “black hole”. All the data corresponding to this community is in [10].

On the other hand, the red community, and the bigger one, contains articles that are related to black hole physics. These are 157 articles that talk about e.g. **AGNs (Active Galactic Nucleus), entropy in black holes, acoustic black holes, primordial black holes, Reissner-Nordström and Kerr black holes, black holes and quantum gravity, etc.** We have uploaded to drive all data corresponding to this community in [11].

In relation to the rest of the articles, seeing figure 10 or figure 12 one can detect at least three more groups; at bottom left, bottom right and top right all of them near the big group we previously refereed to as the red community. We have searched for the articles in these groups and almost all of them are black hole physics related or physics related. The one of bottom left is related to **colliders and black hole physics in colliders**. The bottom right one is related to **x-ray polarimetry and search for high energy sources** and as for the third group, the top right one, it is related to **acoustic black holes and vibration on plates (acoustic black holes experiments)**.

The remaining articles are colored different in figure 12 and that means our algorithm has classified them in different communities. This is because of the modifications added but as already mentioned before we have to be aware of that when using the algorithm.

IV. CONCLUSIONS

In this work we have been able to implement the results of short-path betweenness presented on Newman and Girvan’s article on 2004 [1] about community detection reproducing the calculations for Zachary’s Karate Club social network. We saw they are equal and so the implementation works well.

Then, we were able to obtain from a data base of 20000 articles ([5] or in .csv format [4]) a weighted network of 522 articles. Throwing away those articles that have a really short amount of words in their abstracts and those that have an exaggerated amount we built a weighted network of 12234 articles counting the coincidence of words between their abstracts. Later we arrived to the 522 node network using histograms of the edges’ values and keeping only those high connected articles.

After that we have applied the Short-path Betweenness community detection algorithm with some modifications and obtained a division of the network in two main communities. But still it wasn’t able to detect more communities as the ones we talked about in subsection III B.

Although the algorithm in [1] conforms a great implementation of important concepts in the complex networks field (such as *betweenness* or *modularity*), in practice it is not worth it, unless some modifications are added, due to its computational complexity: $O(m^2n)$. This is reflected on the fact that no more than a year after Newman already published an article with a faster algorithm based on a completely new way of detecting communities (even though it still uses modularity as a guide of the goodness of a network’s agglomeration, see [8]). Also, as mentioned before, nowadays there exist better algorithms for community detection such as the ones mentioned in [7].

On the other hand, with our results at hand we can conclude that with our modifications the Short-Path Betweenness algorithm can be used for networks with big communities and/or communities interconnected via few connections but one can’t rely 100% on it if communities are small and have a considerable amount of interconnections.

[1] M. E. J. Newman and M. Girvan, Finding and evaluating community structure in networks, Physical Review E - Statistical, Nonlinear, and Soft Matter Physics **69**, 10.1103/PHYSREVE.69.026113 (2004).

[2] M. E. J. Newman, Mixing patterns in networks, Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics **67**, 13 (2002).

[3] C. G. Sánchez, Github repository complexnetworksproject (2024).

[4] C. G. Sánchez, Excel data (2024).

[5] Scopus database black holes articles from 2015 to 2023.

[6] M. Newman, *Networks: An introduction* (2010).

[7] R. Lambiotte, A gentle introduction to network science: Dr renaud lambiotte, university of oxford (2019).

[8] M. E. Newman, Fast algorithm for detecting community structure in networks, Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics **69**, 5 (2004).

[9] W. W. Zachary, An information flow model for conflict and fission in small groups,

https://doi.org/10.1086/jar.33.4.3629752 **33**, 452 (1977).

[10] C. G. Sánchez, Orange community data (2024).

[11] C. G. Sánchez, Red community data (2024).