Finding communities in complex networks using modularity optimization algorithm

Simone Gasperini*, Giuseppe Filitto*

Abstract
Finding communities is one of the most outstanding issues and of considerable interest in the analysis of complex networks. As a matter of facts many networks of scientific, social and biological interest are found to divide naturally into communities.
Among the different approaches adopted for this task, one highly effective method is the optimization of a quantity called modularity. Here we present the implementation and the performance analysis of Newman's modularity algorithm for communities detection and we discuss about features and limits of this approach.
We show the results of the algorithm both on toy-models, specifically implemented for this task, and real-world networks, taken from the Stanford Large Network Dataset Collection (SNAP).

Keywords
Complex Networks - Communities - Clustering - Modularity - Resolution Limit

* *Physics and Astronomy Department, UNIBO - University of Bologna*

Contents

Introduction

Many systems of scientific, social and biological interest can be represented as networks. Very common examples include the World-Wide-Web, social networks such as Facebook and Twitter, but also metabolic processes, immune system and so on. These networks consist of nodes (or vertices) and edges (or links) which are the connection between them.
Among the features of a network there is the community structure that can be defined as a specific partition of nodes in sets in which there is a greater density of edges within them than the one between them. The detection of such sets could be of practical and theoretical relevance, leading to useful information about the network.
Several algorithmic approaches can be adopted for this task such as spectral partitioning, hierarchical clustering and edge-removal technique based on betweenness. However, some of these only work well for specific types of problems and/or the algorithm is highly demanding from a computational point of view. For instance, the Girvan-Newman algorithm (Ref. [1]), based on iterative removal of edges with highest *betweenness centrality*, runs in $O(m^2 n)$ for a network of n nodes and m edges and in $O(n^3)$ for a sparse network ($m \sim n$).
In this report, we present a pure-Python implementation of a different algorithm by Newman (Ref. [2]), based on *modularity optimization* and exploitable for undirected unweighed networks. The time complexity of this approach turns out to be $O(n^2)$ for a sparse graph which is the kind of networks we are going to analyze.
We also investigate the performance of the modularity-based algorithm (by using specific toy-models) to understand and empirically verify its limitations, theoretically described in the paper by Fortunato and Barthélemy about resolution limit in modularity framework (Ref. [3]).
Finally, we show some results of the application of such algorithm to real-world networks data, downloaded from the Stanford Large Network Dataset Collection (SNAP).
The whole implementation is done using only the basic tools of numerical computation provided by the *Numpy* library.

1. Algorithm

In this section, we briefly discuss the general technique of modularity optimization and then we provide a detailed description of our specific algorithm implementation.

1.1 Modularity optimization

Modularity is a measure of the goodness of nodes partition into communities. From a mathematical point of view, we define modularity as the fraction of edges falling within communities minus the expected fraction in the equivalent network with edges placed at random (Ref. [4]).
Let A_{vw} be the elements of the adjacency matrix:

$$A_{vw} = \begin{cases} 1 & \text{if node v is connected to node w} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

and let's define the degree k of a node as the number of edges connected to it:

$$k_v = \sum_w A_{vw} \tag{2}$$

Suppose node v belongs to community c_v, we can write the modularity as:

$$Q = \frac{1}{2m} \sum_{v,w} \left[A_{vw} - \frac{k_v k_w}{2m} \right] \delta(c_v, c_w) \tag{3}$$

where the δ-function $\delta(i,j) = 1$ if $i = j$ (i and j represent communities) and 0 otherwise; and m is the total number of egdes in the network.
In order to choose which is the best partition, according to the definition of modularity, one should search among all possible divisions the one that results in the maximum modularity. However, this task is very hard especially for large networks. Reasonably good results can be achieved by approximate optimization techniques: the algorithm, implemented and used in this work, performs an optimization based on the iterative merging of different communities whose amalgamation produces the largest increase in the modularity Q defined above.

1.2 Implementation

The algorithm starts considering n different communities (each node belongs to its own community) and then performs iterative merging of different set on nodes. This merging procedure consists in at most $n-1$ steps and it goes on until the modularity reaches a local maximum (no possible further increase).
The most straightforward implementation of this idea involves storing the entire adjacency matrix of the graph as an array of integers and repeatedly merging pairs of rows and columns as the corresponding communities are joined.

However, if the adjacency matrix is sparse (how it is in most practical applications) this process can be carried out more efficiently using data structures for sparse matrices and thus saving memory space. Moreover, rather than maintaining the adjacency matrix and calculating at each iteration ΔQ_{ij} for all the pairs i, j of communities, we can instead maintain and update a similarly sparse matrix of ΔQ_{ij} values, saving a good deal of execution time.
Another data structure to be stored and updated at each iteration is an ordinary array a_i containing the fraction of edges that are attached to all nodes in community i, defined as:

$$a_i = \frac{1}{2m} \sum_v k_v \delta(c_v, i) \tag{4}$$

Our algorithm can now be described in the following steps:

1. Starting off with each node being the sole member of a community of one, calculate the initial values of the matrix ΔQ_{ij} as:

$$\Delta Q_{ij} = \begin{cases} \frac{1}{m} - \frac{k_i k_j}{2m^2} & \text{if } i, j \text{ are connected} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

   and set the initial values of the array a_i to:

$$a_i = \frac{k_i}{2m} \tag{6}$$

2. Select the largest ΔQ_{ij}, merge the community i and j and delete the community i.

3. Remove the ith row and column of the matrix ΔQ_{ij} and update its jth row and column according to the following rule:

$$\begin{cases} \Delta Q'_{jk} = \Delta Q_{ik} + \Delta Q_{jk} & \text{if } k \text{ is connected to both } i \text{ and } j \\ \Delta Q'_{jk} = \Delta Q_{ik} - 2a_j a_k & \text{if } k \text{ is connected only to } i \\ \Delta Q'_{jk} = \Delta Q_{jk} - 2a_i a_k & \text{if } k \text{ is connected only to } j \end{cases} \tag{7}$$

   for each community k with at least one connection to community i or j.

4. Update the values of the array a_i according to the following rule:

$$a'_j = a_j + a_i \qquad a_i = 0 \tag{8}$$

5. Repeat steps 2, 3, 4 until the largest value of ΔQ_{ij} selected in step 2 is 0 or negative (no possible further increase of the modularity Q).

2. Performance Analysis

In the first part of this section we describe the toy-model, developed to build custom structured networks, then after showing some algorithm basic features, we verify the limitations of the modularity optimization approach, running multiple clustering simulations on different kinds of synthetic networks (increasing the blocks heterogeneity, varying their inter/intra-connectivity and and investigating the resolution limit).

2.1 Toy-model

We implemented a toy-model graph in order to build custom networks with specific properties (for instance a defined community structure).
The base object is an undirected unweighted network, provided with several different generators. It is possible to generate *ErdosRenyi* graphs (starting from a fixed number of nodes and a probability of random connection) and *RandomNetwork* graph (starting from a fixed number of nodes and a fixed number of random edges). Moreover, we also implemented a more sophisticated model called *ErdosRenyiBlocks* which let us to build a complex random network composed by different blocks randomly connected. The parameters of this object are the following:

- *blocks_sizes*: array of integers representing the sizes of each block (with length equal to the number of blocks);
- *prob_matrix*: symmetric matrix of floats representing the probability of edge creation within and between the blocks (with dimensions equal to the number of blocks).

Finally, we also implemented a model for generating a complex network composed by different sub-networks connected by a fixed number of random egdes. It is called *CompositeNetwork* and it takes the following paramentes:

- *unetworks*: list of undirected unweighted sub-networks to be attached;
- *edges_matrix*: symmetric matrix of integers representing the number of randomly placed edges to connect the different sub-networks.

In the following, we will study more deeply the performances of the community detection algorithm, varying all the parameters in toy-model object generation.
To provide a first visual example of a single simulation, we show here a network composed by 4 random blocks of similar sizes with different edge densities within them. The community structure is depicted both in a graphical representation of the ”ordered” adjacency matrix (Figure 1) and in a colored visualization of the graph (Figure 2).

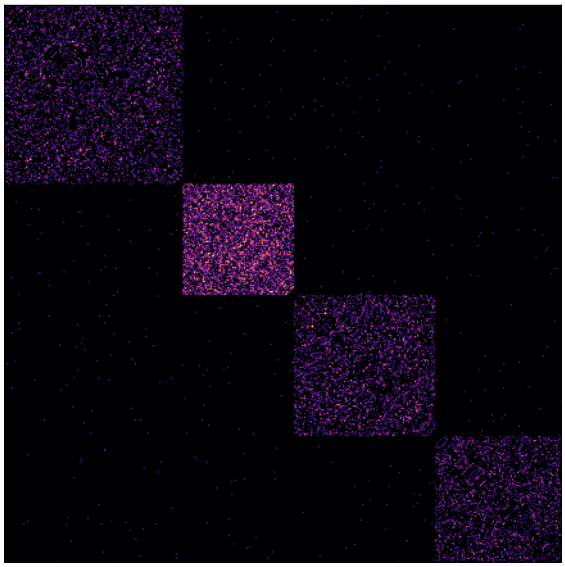


Figure 1. ”Ordered” blocks adjacency matrix after the application of the communities detection algorithm.

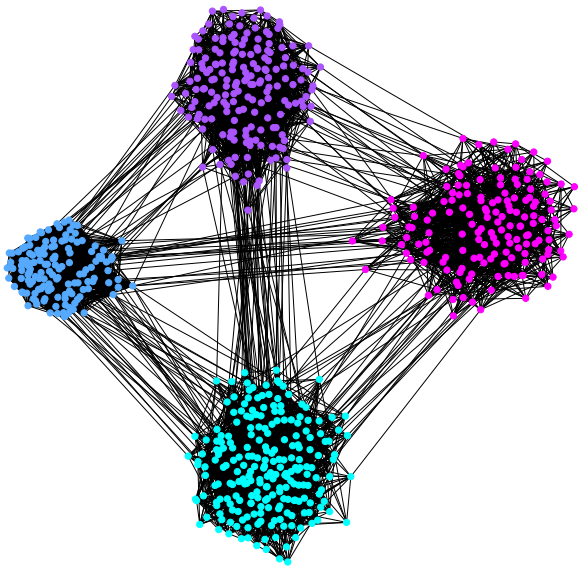


Figure 2. Colored graph visualization of the network. Each color represents a different community (*NetworkX* drawing method).

2.2 Clustering time

To empirically investigate the time complexity of our algorithm, we applied it several times on different sparse networks always composed by 2 blocks and adding at each iteration a fixed amount of nodes to each block (with fixed intra- and inter-connectivity). The result of these simulations is showed in Figure 3 (clustering time as a function of the number of nodes n).
The time trend for the sparse networks shows, as expected, a quadratic behaviour increasing the number of nodes (time complexity: $O(n^2)$).

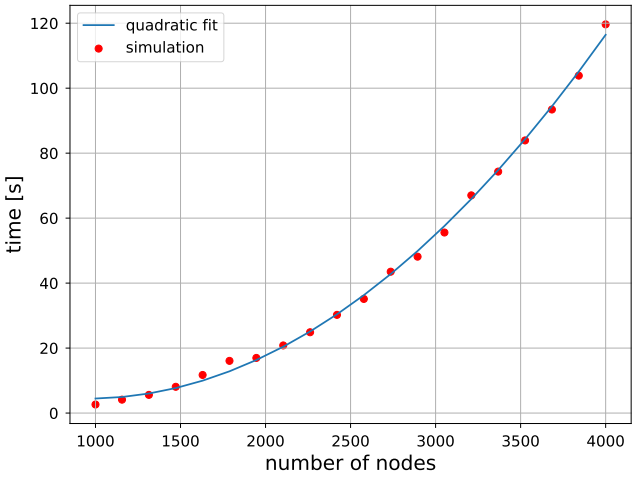


Figure 3. Clustering time increasing the number of nodes. The blue line represents the best quadratic fit.

2.3 Modularity for disjoint blocks

In this second simulation, we consider the simple case of networks composed by an increasing number c (ranging from 2 to 60) of disjoint blocks.

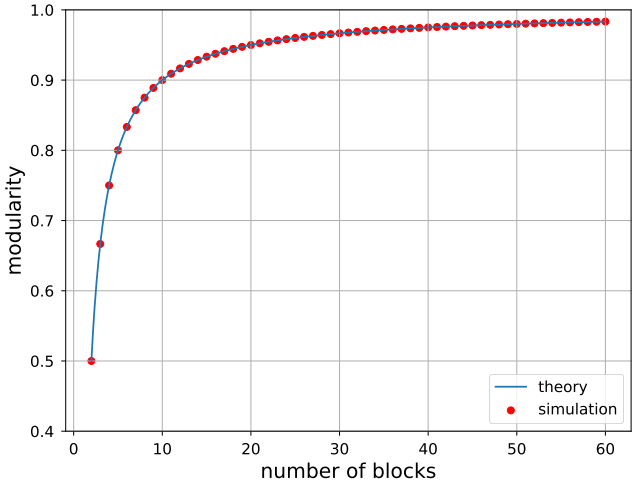


Figure 4. Modularity of the network progressively increasing its number of disjoint blocks. It tends to 1 as the number of blocks goes to infinity.

Each block is a connected network with a random number of nodes n_i and a fixed number of edges $m_i = m$ (placed at random).
In this trivial case, the algorithm is able to properly identify all the communities independently on their number and sizes and perfectly assign each node to the correct community.
Notice that this disjoint blocks model represents the case of maximum modularity (no edges at all between the blocks). However, as theoretically proved in Ref. [3], the modularity turns out to be 1 only in the limit of an infinite number of communities c.
In general, the modularity Q for this kind of model is given by the following function:

$$Q(c) = 1 - \frac{1}{c} \tag{9}$$

As shown in Figure 4, there is a perfect agreement between this theoretical function and the modularity computed on the detected partition after the clustering.

2.4 Blocks heterogeneity

We executed some simulations to better understand how the communities detection algorithm performs on networks composed by heterogeneous blocks.
These networks are generated with a fixed number of randomly interconnected blocks but their heterogeneity progressively increases.
In particular, we sampled the nodes number of each block from a wider and wider uniform distribution (centered in 100), starting from an homogeneous network (about 100 nodes per block) up to a very heterogeneous one (from less that 25 to almost 200 nodes per block). The distributions of the blocks sizes for each simulation are shown in Figure 5 to give an idea about how network blocks are more and more heterogeneous for each simulation.
In principle, if the network has a sufficiently clustered structure (it depends on the ratio between the number of edges intra- and inter-clusters), our clustering algorithm should be always able to detect these communities correctly, independently on their size.
To empirically verify this, we did a comparison between the networks modularity of the detected partition (after the clustering) and the so called natural or "true" partition, known a priori and used in the generation phase.
Performing all the simulations, we can observe from Figure 6 that the modularity computed on the detected partition always matches the modularity of the natural partition.
Moreover, we also observe that the modularity starts from an almost stable maximum value for the homogeneous network (it depends on how many blocks there are and how the edges are distributed), but, after the first few simulations, begins to decrease as the heterogeneity increases.

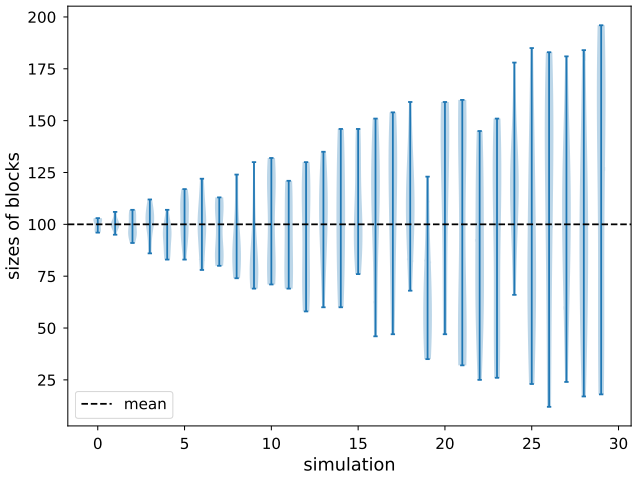


Figure 5. Density estimation of blocks sizes for each simulation.

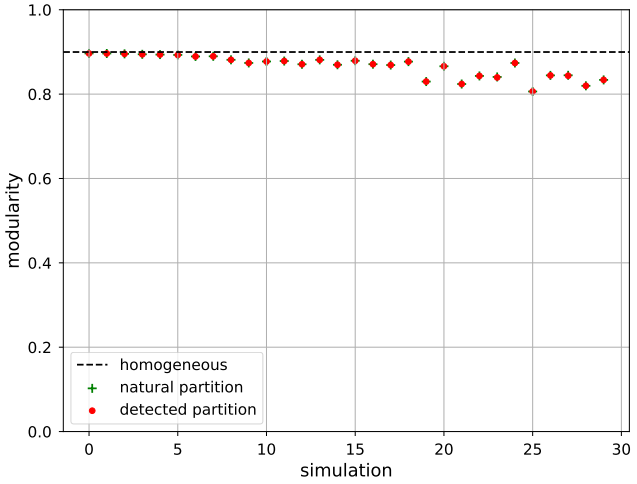


Figure 6. Modularity of natural partition (in green) and detected partition (in red). The black dashed line represents the expected modularity for a perfectly homogeneous network.

2.5 Blocks connectivity

In the previous section we showed that the modularity and the communities detection algorithm strictly depends on how the egdes are distributed in the network with respect to the null random model.
To study more precisely this dependence, we performed two different kinds of simulations:

- increasing the number of intra-blocks edges with different fixed values of the inter-blocks connection probability ($p = 0.002, 0.006, 0.010$);
- increasing the number of inter-blocks edges with different fixed values of the intra-blocks connection probability ($p = 0.10, 0.15, 0.20$).

In the first case, we generated a series of networks composed by 5 blocks of 100 nodes and the number of intra-blocks edges (per block) varied between 150 and 4000 (each block is a random network always built to have a single connected component).
In Figure 7 we show the resulting modularity of both detected and natural partition as a function of intra-blocks edges number.
As expected, the three different curves are monotonically increasing because each block is more and more dense at each simulation. It also interesting to notice that they finally all converge to a maximum value of 0.8 (see Eq. (9) for $c = 5$) as the intra-blocks density becomes much larger than the one between.
For a small number of edges within the blocks, instead, not only the modularity is lower but the algorithm struggles to find the communities and/or to assign each node to the correct community: for this reason, the modularity of the detected partition is smaller that the modularity of the natural one.

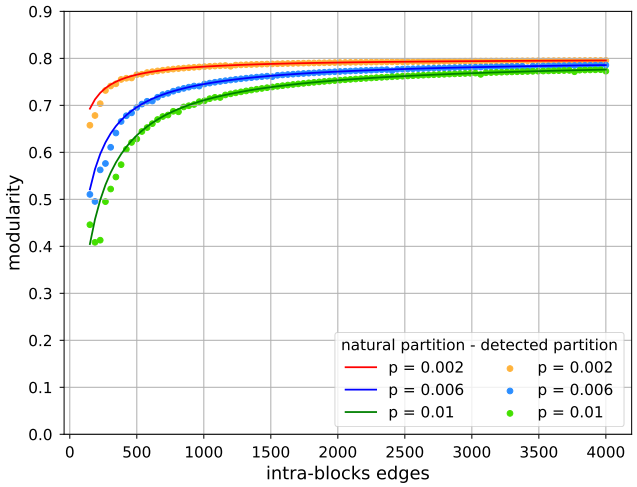


Figure 7. Modularity as a function of the intra-blocks edges number per block. The three colored curves refer to different values of the connection probability p between the blocks.

In the second case, we considered again a series of networks composed by 5 blocks of 100 nodes but we varied the number of inter-blocks edges (per blocks pair) between 1 and 300.
In Figure 8 we show the resulting modularity of both detected and natural partition as a function of inter-blocks edges number.
As expected, the three different curves start from a maximum modularity of 0.8 (see Eq. (9) for $c = 5$) and they are monotonically decreasing because each block is more and more densely connected to all the others.
Moreover, as the number of inter-blocks edges increases, the algorithm is not able to properly find the communities and, as a result, the modularity of the detected partition is significantly lower that the modularity of the natural one. In particular, this is more evident for networks with less dense communities (intra-blocks connection probability $p = 0.10$ and $p = 0.15$).

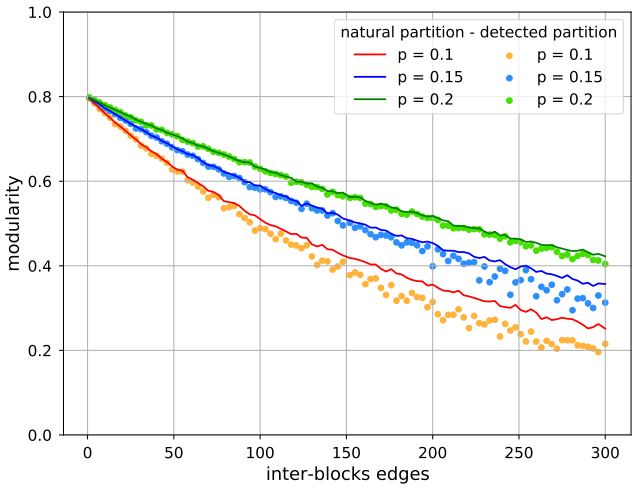


Figure 8. Modularity as a function of the inter-blocks edges number per blocks pair. The three colored curves refer to different values of the connection probability p within the blocks.

The last interesting observation is about the plot in Figure 9. We actually repeated the same simulations as before, varying the inter-blocks connectivity, but we increased the number of edges between each pair of blocks up to 1200. Beyond a specific number of these edges (which depends on the internal density of the blocks) the expected modularity would be 0 because the network becomes completely devoid of communities structure. However, once the clustering is completed, the minimum modularity value computed on the detected partition is about 0.1.

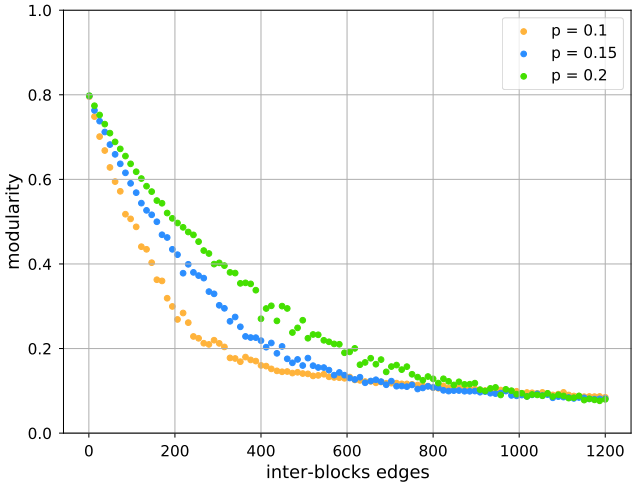


Figure 9. Modularity as a function of the inter-blocks edges number per blocks pair. The three colored curves converge to the same minimum value.

This is consistent with no observation of a community structure in the network (the usual threshold is about 0.3) but it is still higher than 0. Moreover, this minimum value seems to be the same for the three curves, meaning that it is independent on the intra-blocks probability of connection p.
This effect reflects a limit of the modularity optimization approach (which is simply a mathematical function maximization) in the communities detection problem because the algorithm finds some non-zero modularity partition even in cases of completely random and unstructured networks.

2.6 Resolution limit

Let us consider the specific case of a maximum modularity connected network to further investigate the features and the limitations of this optimization technique.
Ideally, to maximize the contribution to modularity of each block, we should reduce the number of edges connecting blocks as much as possible. To keep the network connected, we shall consider the simple ring-like graph: in this case, the number of blocks c is equal to the total number of inter-blocks edges and the maximum modularity is reached when all the blocks has exactly the same number of internal edges $m_i = m = M/c - 1$, where M is their total number.
The modularity Q of this ring configuration model (see Ref. [3] for more details) is given by the following function:

$$Q(c,M) = 1 - \frac{c}{M} - \frac{1}{c} \tag{10}$$

We want now to maximize this function when the number of blocks c is variable. For this purpose, we treat c as a continuous variable, take the derivative of the function, and make it vanishing: for $c = c_{MAX} = \sqrt{M}$ we have the maximum value of the modularity in Eq. (10).
To verify this result empirically, we run the clustering simulation on networks of this type and plotted the modularity of the detected partition as a function of the blocks number, as shown in Figure 10.

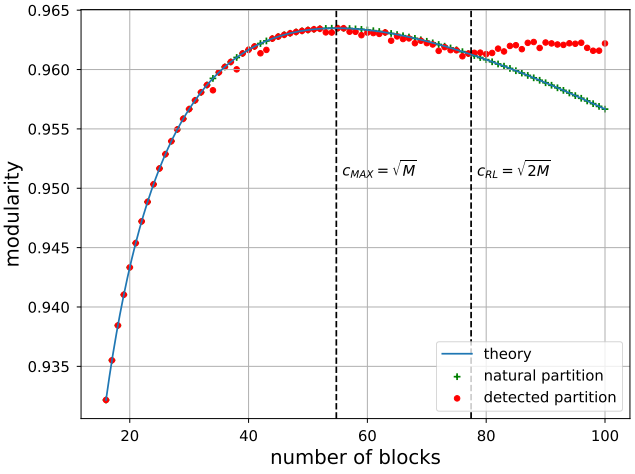


Figure 10. Modularity as a function of the blocks number. The theoretical function is plotted in blue.

Notice that the the maximum modularity is, as expected, in $c = \sqrt{M}$ and that for $c > \sqrt{2M}$ the measured modularity is higher that the one expected.

It's interesting to notice that, for values of $c > \sqrt{2M}$, the algorithm is not able to perform a correct communities detection : in particular, above this resolution limit c_{RL} (see again

Ref. [3]), the detected partition (maximum modularity) does not correspond to the natural partition because the algorithm tends to merge pairs of communities.
This effect further remarks the limits of the modularity-based approach applied to the analysis of community structure in complex networks.

3. Real Networks Results

This section is aimed to present and discuss the results about the analysis of some real-networks taken from the Stanford Large Network Dataset Collection (SNAP).

3.1 Real vs Random

To investigate the role of edges distribution for modularity optimization algorithm, we made a comparison between some real networks and their analogous random networks, generated with the same number of nodes and edges.
The real data belongs to the SNAP Stanford Large Network Dataset Collection (Ref. [5]) which is a database edited by the Stanford University for the analysis of many different kinds of networks.
In Table 1 the results of the analysis are shown. Regarding the real modularity Q_{real}, we always obtain a value that is above the threshold 0.3 as expected from theory (Ref. [4]).
As for the random network modularity Q_{random}, even if it is, as expected, always below the threshold, in all cases is greater than 0. This is due to the same effect described in the previous section and it represents a limitation of the modularity optimization approach.
Analyzing more carefully Table 1, it is also possible to notice that, as the ratio Edges/Nodes increases, Q_{random} increases as well. As for Condensed Matter, this effect is quiet evident: in fact, this network is very sparse and Q_{random} reaches a particularly high value.

Dataset	Nodes	Edges	Q_{real}	Q_{random}
Astro Physics:	18,772	198,110	0.496	0.162
Facebook:	22,470	171,002	0.690	0.193
HEP Collaboration:	12,008	118,521	0.593	0.173
Condensed Matter:	23,133	93,497	0.647	0.297
Github:	37,700	289,003	0.385	0.189

Table 1. Table of real and random networks modularity.

3.2 Community structure analysis

Once the functionality of the algorithm has been ensured and its performances analyzed, our goal was to apply it to real-world networks to find their communities. Among the different real-world data available in SNAP database, we selected some undirected unweighted networks up to about 30,000 nodes and 200,000 edges.
For each dataset, we show the results of communities detection using two different plots: a communities graph and a barplot for their sizes. The former is a *NetworkX* graph in which each node represents a community and the higher is the number of nodes in that community, the larger is the size of the node itself; the latter, instead, is a barplot showing the number of nodes for each community (in logarithmic scale). Let's notice that we only show the communities with a minimum number of nodes set to 15.
As expected, in general we have few central big communities with lots of connections and a very high number of smaller clusters with very few links (in many cases just one).
In Table 2 we summarize the main datasets statistics, in which the modularity and the number of communities are measured after the clustering process.

Dataset	Nodes	Edges	Modularity	Communities
Deezer:	28,281	92,752	0.635	230
Facebook:	22,470	171,002	0.690	195
General Relativity:	5,242	144,960	0.811	424

Table 2. Summary table of the analyzed real networks.

Figure 11. SNAP logo

3.2.1 Deezer

Deezer is an online music streaming service that has 14 million monthly active users on different platforms such as Android, iOS, MacOS, Windows and so on. The network of Deezer users was collected from the public API in March 2020. The nodes represent Deezer users from European countries and edges are mutual follower relationships between them.

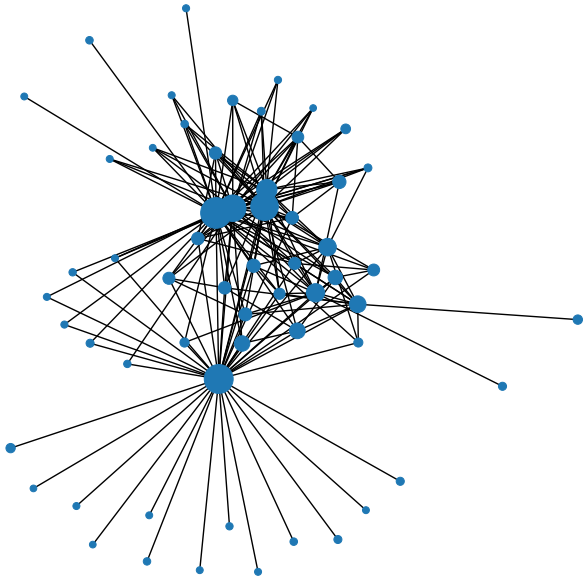


Figure 12. Deezer communities graph. Each node represents a community and the size of each node is proportional to the number of nodes in that community.

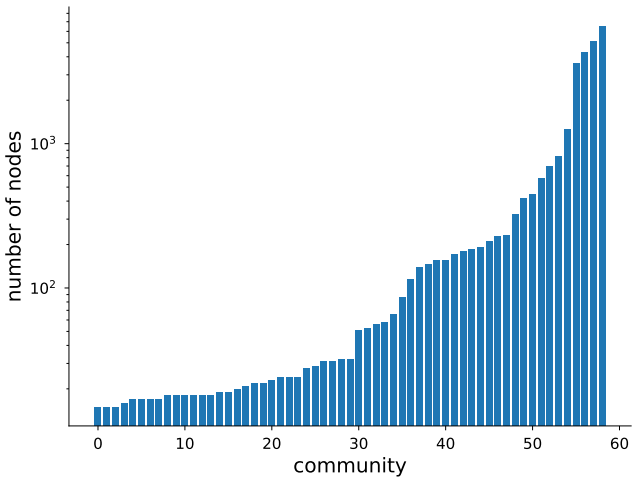


Figure 13. Deezer communities barplot. On the y-axis the size of the communities found is plotted in logarithmic scale.

3.2.2 Facebook

This network consists of a page-page graph of verified Facebook sites. Nodes represent official Facebook pages while the links are mutual likes between sites. This graph is restricted to pages from 4 categories defined by Facebook: politicians, governmental organizations, television shows and companies.

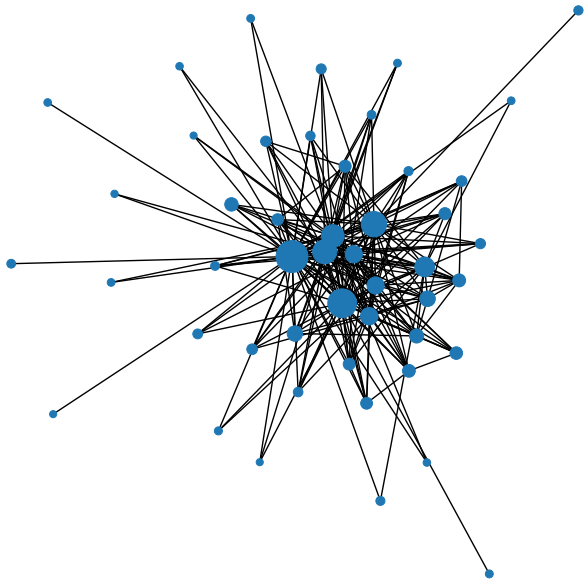


Figure 14. Facebook communities graph. Each node represents a community and the size of each node is proportional to the number of nodes in that community.

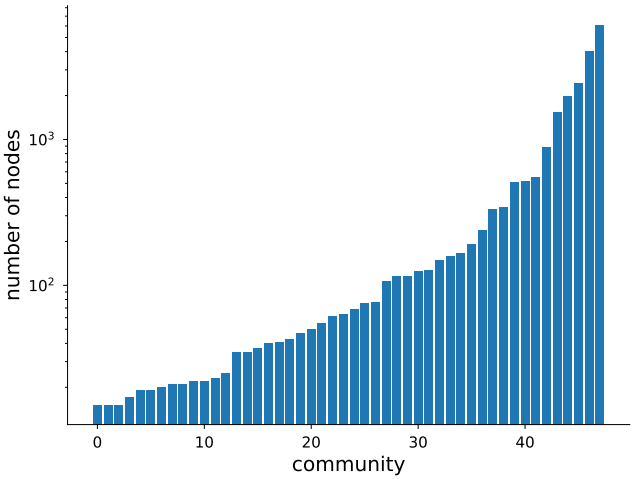


Figure 15. Facebook communities barplot. On the y-axis the size of the communities found is plotted in logarithmic scale.

3.2.3 General Relativity

General Relativity (and Quantum Cosmology) collaboration network covers scientific collaborations between authors papers submitted to General Relativity and Quantum Cosmology category. If two authors co-authored a paper the nodes corresponding to those authors are connected. The dataset covers the period of time between January 1993 and April 2003.

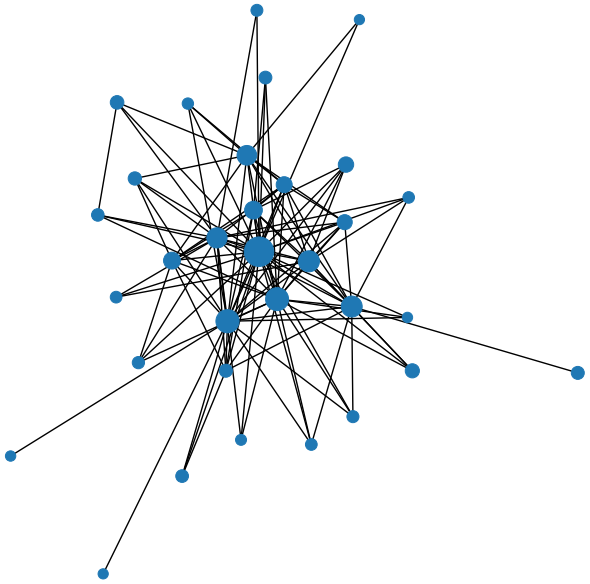


Figure 16. General Relativity communities graph. Each node represents a community and the size of each node is proportional to the number of nodes in that community.

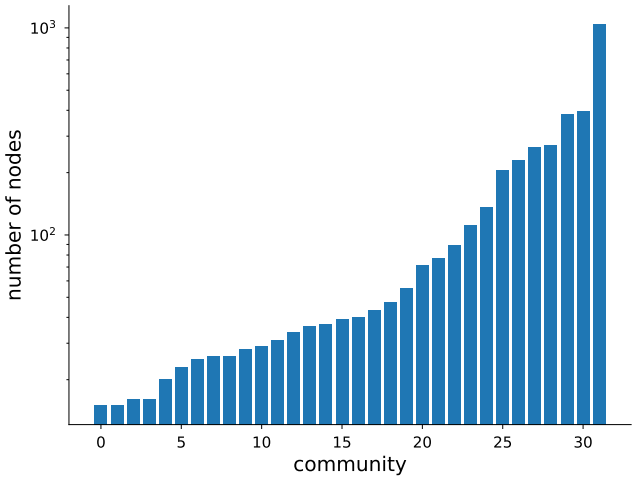


Figure 17. General Relativity communities barplot. On the y-axis the size of the communities found is plotted in logarithmic scale.

4. Conclusions

In this report we described and analyzed an algorithm for finding communities in complex networks using the modularity optimization approach. First of all, we analyzed the performance of the algorithm and we showed that our implementation runs essentially in $O(n^2)$ for sparse networks of practical interest. This is a remarkable result since other algorithms (for example the ones based on the edges-removal approach of Ref. [1]) are in general slower.

We tested, by running several simulations, the modularity-based algorithm investigating the role of different network features, such as blocks heterogeneity and blocks connectivity. Moreover, we also investigated the limitations of the algorithm: in particular, we showed the resolution limit for communities detection by running our simulations.

Then we applied the algorithm to some real-world networks taken from the Stanford Large Network Dataset Collection (SNAP), making the comparison with their analogous random network.

Finally, the main results of communities detection analysis on real networks are provided.

A. Appendix: Code features

The code used for this work is written in Python and fully available at: https://github.com/SimoneGasperini/network_clustering.

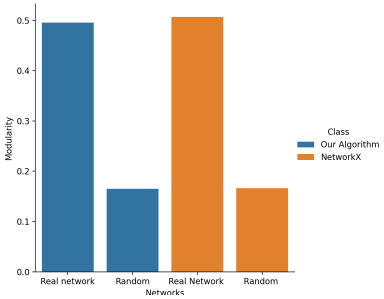
We adopted an object-oriented approach and the only installation requirements to access the basic networks analysis functionalities are the Python libraries *Numpy* and *Matplotlib*. To be able to use all the visualization tools, it is further required the installation of the open-source complex networks software *NetworkX*.

The main class *UndirectedNetwork* has been implemented to support undirected unweighted networks and it basically contains the core of the program. Among the other features, it provides a method able to directly create a network object starting from a text file containing the dataset in edge list format. Moreover, the most important feature of the class is that each network object is internally stored as a Python dictionary in which each node (represented as an integer number ID) is mapped to the corresponding list of nodes connected to it. This data structure of the network object is very efficient both to save memory and to speed up all the required computations, for instance in the case of the clustering algorithm to detect communities. Of course, this is true especially for sparse networks ($m \sim n$): indeed, in this case storing the entire adjacency matrix is much more expensive in terms of memory storage.

B. Appendix: NetworkX validation

To validate our results, we made a comparison using NetworkX as a reference and, as expected, the computed modularities are essentially the same.

As an example, we present the case of Astro Physics dataset and its analogous random network.



References

[1] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. 2002.

[2] A. Clauset, M. E. J Newman, and C. Moore. Finding community structure in very large networks. 2004.

[3] S. Fortunato and M. Barthélemy. Resolution limit in community detection. 2007.

[4] M. E. Newman. Modularity and community structure in networks. 2006.

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