Community detection in complex networks using Genetic algorithms

Simon Lehnerer, Physics MSc student at ETH Zurich

August 9, 2017

ABSTRACT

Detecting the community structure is of great interest when analysing the topology of a network, however not a trivial problem. In this article a genetic algorithm is proposed which finds the community structure of a network based on the maximization of a quality function called modularity. Tests using several sample networks show that it reliably finds the community structure. However it does not resolve sufficiently small communities as intuitively expected due to an effect known as resolution limit.

1 INTRODUCTION

Broadly speaking, the *community structure* of a network is the division of the network into groups of nodes called *communities*, which are internally densely connected and loosely connected to nodes from other communities. There is no unique formalized definition of the term *community* (*structure*), but a common way to define it is with the help of a quality function called *modularity* [3]. The modularity of a community partition is a measure, that is the higher the "better" a partition indicates the community structure of the network. Thus the community structure is given by the community partition with maximal modularity.

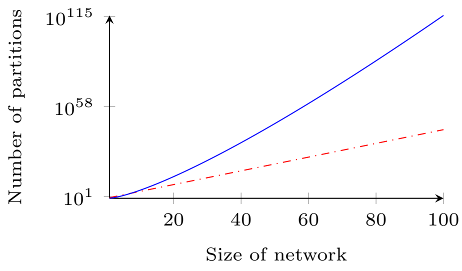

Figure 1: Number of possible partitions (blue) compared with exponential curve (red). For a network consisting of twelve nodes there already exist more than one million possible community partitions.

There is no method known that can directly determine this community partition. What is more, the number of possible community partitions of a network with n nodes is given by the Bell number B_n, which has factorial growth (see fig. 1). A "brute-force" approach to find the community structure by comparing all possible partitions becomes impractical for larger networks. Furthermore, the modularity function features a high degeneracy which makes it difficult to find its global maximum [1]. For that reason we need an algorithm that can find the community partition with maximal modularity both efficiently and effectively. For this purpose a genetic algorithm (see fig. 2) is proposed in the following.

2 THE ALGORITHM

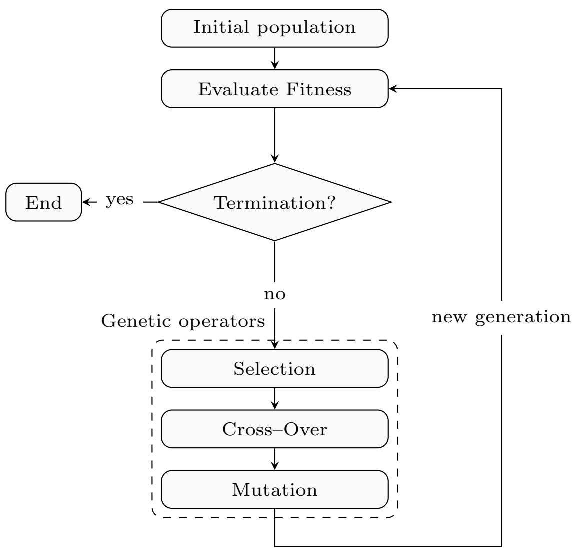

Figure 2: Schematic of a genetic algorithm
Genetic algorithms, inspired by processes of evolution, are stochastic optimization methods used to find an optimal solution in a large solution space. They use a *population* of *individuals* where each *chromosome* of an individual represents a possible solution. A *fitness* value reflects the quality of the solution of an individual and is used to stochastically perform *selection*, *crossover* and *mutation* operations to form a new generation of individuals (also called *children*). The process is repeated until a reasonable solution is found.

The algorithm described here is aimed at finding the community structure of an undirected, unweighted network. A chromosome represents a possible community partition of the network. Technically it is a list where the i-th entry contains the community label c_i of node i (see fig. 3). Thus two nodes i and j are in the same community iff $c_i = c_j$. The chromosomes of the initial population are generated randomly.

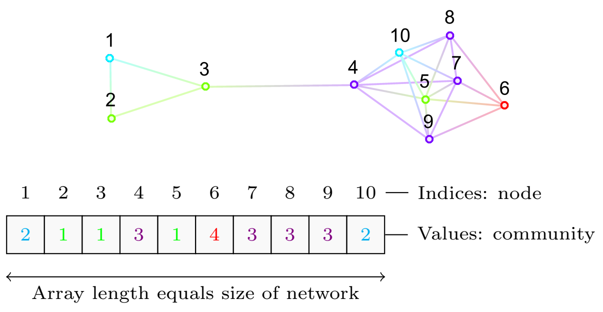
1
2
3
4
5
6
7
8
9
10
1 2 3 4 5 6 7 8 9 10 — Indices: node
2 1 1 3 1 4 3 3 3 2 — Values: community
Array length equals size of network

Figure 3: Example of a chromosome for a network of size 10. It represents a possible community partition of the network (the communities are color-coded so that two nodes are in the same community iff their color is the same).

The fitness is calculated from the modularity of the community partition, which is a global measure and given by

$$Q = \frac{1}{2m} \sum_{i,j} \left(A_{ij} - \frac{d_i d_j}{2m} \right) \delta(c_i, c_j) \quad \in [-0.5, 1]$$

Here m is the number of total links, A_{ij} is the adjacency matrix (which is 1 if there is a connection between nodes i and j, otherwise 0), d_k is the degree of node k, $c_k = l$ if node k is in community l, and $\delta(c_i, c_j) = 1$ if $c_i = c_j$, otherwise 0. The formula compares the number of internal links in a community with the expected number of links of a randomized graph and therefore implies a definition of the term *community*: A community is a group of nodes that is more tightly connected than expected [3].

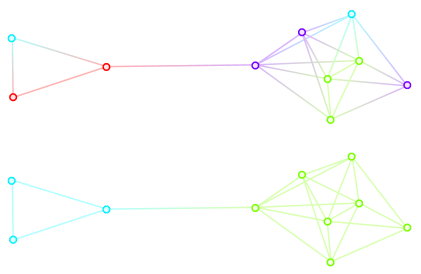

Figure 4: The figure shows two possible community partitions of a network. While the upper partition has a low modularity of about -0.08, the partition below it has a modularity of about 0.22 which is maximal among all possible partitions and thus shows the community structure of the network.

The algorithm provides two methods of selection. The first, in the following called *Elitism*, only keeps the fittest individual. All children are unaltered copies of this individual. The second method, in the following called *Mating*, retains a certain fraction of the fittest individuals (*parents*), and performs cross-over operations to create new children (see figure 5 for a detailed description). Both methods are indeed a variant of *elitism*, where only the fittest individuals are preserved. Important to note is that the community partition is a relational property, meaning that the labeling of the communities is not unambiguous. Two individuals could actually refer to the same community but use different community labels. This is considered by the algorithm by "translating" the community label when doing cross-over operations.

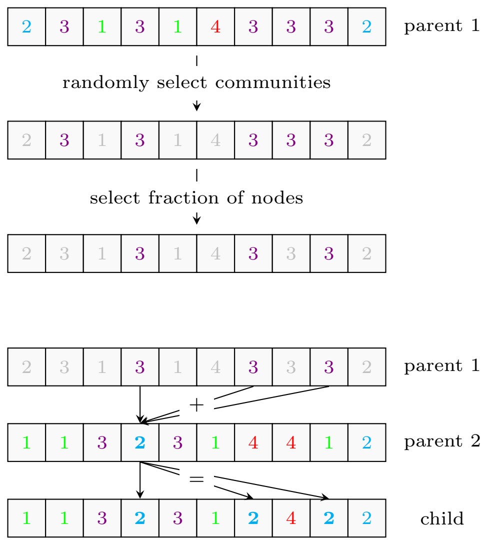

Figure 5: Cross-over process
At first a number of communities of *parent 1* are selected randomly. A certain fraction of the selected nodes are chosen to be transferred to *parent 2*. In order to preserve the community structure the community label of *parent 1* will be translated. For each community to be transferred it uses the corresponding community label of the **first node** in *parent 2* as community label. The resulting chromosome gives the chromosome of the child.

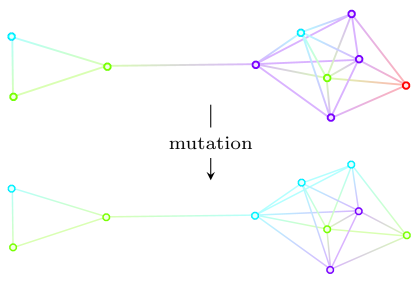

Figure 6a: First mutation process
A random sample of nodes is assigned to random communities. E.g. the community of the red node in the above network is changed from red to green.

The mutation process consists of two independent parts and is applied to each child by a certain chance.

The first mutation process randomly alters parts of the chromosome (see fig. 6a). In the second mutation process a random sample of nodes will inherit the community that is most dominant in their respective neighborhood (the nodes they are directly connected with). This improves the speed of convergence since it is likely that neighbors belong to the same community.

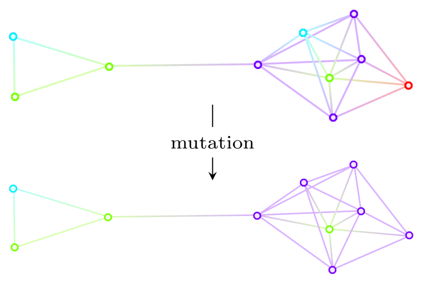

Figure 6b: Second mutation process
A random sample of nodes inherits the most dominant community in its respective neighborhood. E.g. the red node in the above network inherits the purple community since three out of four neighbors are in the purple community.

Finally the algorithm removes communities that consist of only one single node, because a proper community must contain at least two nodes. The whole process is repeated until a certain number of generations have been created. The algorithm then stops and returns the best found solution.

3 EXPERIMENTAL RESULTS

Since the algorithm finds solutions stochastically, its outcome might differ on every execution. To test the effectiveness of my algorithm two networks consisting of two respectively five communities were used (see fig. 7 and 8). For each trial the algorithm was run 100 times with the same parameters and the number of times where the algorithm found the correct community partition was counted.

I used a population size of 20 and, for each child, a 75% probability for the first mutation process and a 50% probability for the second mutation process. The algorithm was set to terminate after 5 generations for the first network and after 50 generations for the second network. For the *Mating*-method the 15% fittest were selected and all nodes from the selected communities were transferred (see fig. 5).

Results of the testing are shown in table 1. The two communities of the first network were correctly detected in all 100 runs within two generations on average. The communities of the second network were detected correctly in 95% of all cases for the *Elitism*-method and 99% for the *Mating*-method within eight respectively seven generations on average. In most of the failed cases the algorithm merged two communities into one single community.

method	fails	avg. gen.	max. gen.
Elitism	0%	2.13 ± 0.34	3
Mating	0%	2.19 ± 0.39	3

(a) Results for first network

method	fails	avg. gen.	max. gen.
Elitism	5%	8.56 ± 5.43	30
Mating	1%	7.08 ± 2.53	27

(b) Results for second network

Table 1: Results of the testing
The columns show the used selection method, the number of runs where the algorithm did not find the correct community partition, and the average and maximum number of generations needed to find the best community partition.

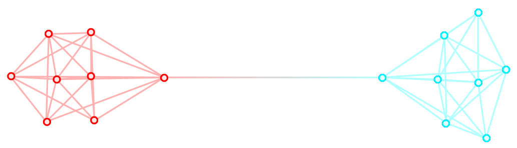

Figure 7: First tested network consisting of 16 nodes, 49 edges and two communities. Number of possible community partitions $\approx 1 \cdot 10^{10}$

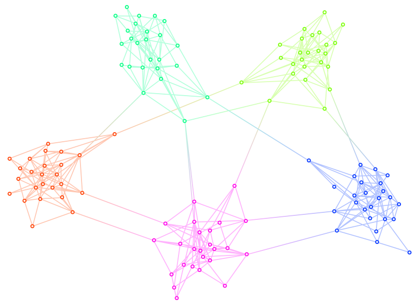

Figure 8: Second tested network consisting of 125 nodes, 390 edges and five communities. Number of possible community partitions $\approx 2.5 \cdot 10^{153}$.

Also, the algorithm was tested using a large network with unknown community structure and its result were compared with *Mathematica*'s built-in function `FindGraphCommunities`. For my algorithm I used the *Mating*-method, the same parameters as for the other two networks and set it to terminate after 100 generations. In order to estimate how far *Mathematica*'s results and the result of my algorithm were apart I calculated the partition distance, i. e. the number of nodes that were classified into different communities (this was transformed into an assignment problem and solved using *Mathematica*'s built-in function `FindIndependentEdgeSet`). My algorithm detected 38 communities (see fig. 9). The community structure differs only slightly from the ones obtained by *Mathematica*'s *Centrality* and *Hierarchical* methods (see table 2).

In terms of speed my implementation of the algorithm can not compete with *Mathematica*'s function. My implementation is almost 10^3 times slower (timewise) and therefore not suited for networks consisting of millions of nodes.

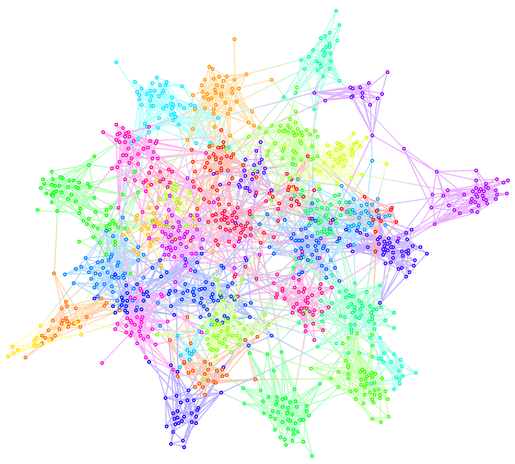

Figure 9: Large network consisting of 1293 nodes and 4145 edges showing the community structure detected by my algorithm. About $6.7 \cdot 10^{2609}$ possible community partitions.

method	communities	modularity	part. dist.
Centrality	38	0.907235	3
Hierarchical	38	0.904835	8
Modularity	35	0.900042	83
Spectral	43	0.817467	143
My algorithm	38	0.907236	-

Table 2: Comparison of the results of *Mathematica*'s built-in function `FindGraphCommunities` with the result of my algoritm. The columns show the method used by *Mathematica*'s built-in function, the number of detected communities, the modularity of the partition and the partition distance.

4 ANALYSIS OF THE ALGORITHM

Due to its stochastic way of finding solutions the algorithm does not always find the best solution, which is supported by the high degeneracy of the modularity function with a lot of local maxima close to the global maximum. However it usually finds a solution that is at least close to the best solution.

Yet there might be a principal problem of the algorithm. Imagine n identical complete graphs K_m with m nodes each, which are arranged on a ring lattice and connected via a single link. Intuitively we would say that each complete graph forms a community since all nodes within a complete graph are densely (in fact maximally) connected and share only two links to nodes from other communities. However Fortunato and Barthélemy proved that for a sufficiently large number of graphs n modularity maximization will lead to a partition that merges pairs of communities into one single community [2]. Therefore modularity maximization can lead to a partition which seems counter-intuitive as it "neglects" small community structures, which is however the consequence of the implicit definition of a community using modularity (see fig. 10). In general, communities smaller than a certain threshold, which depends on the network size, will always be merged, which is known as *resolution limit*. A possible way to solve this problem is to check every community whether it is a merging of smaller communities.

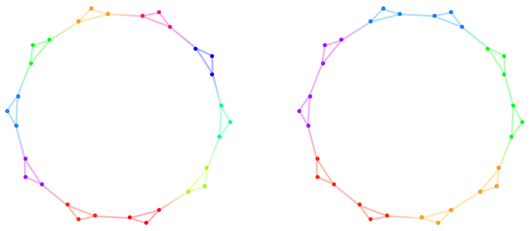

Figure 10: Resolution Limit
Example network consisting of 10 complete graphs with three nodes each. For the left, intuitively expected community partition the modularity is 0.65, whereas in the right partition, where each time two communities are merged, the modularity is 0.675, which is higher than for the expected partition. Modularity maximization will therefore lead to a community partition that might be counter-intuitive.

5 CONCLUSION

The algorithm shows a good performance in detecting the communities of complex networks and is suited for cases where a community partition close to the best community partition is required. However, from the investigation of the fitness function *modularity* the question arises if one could improve the definition of modularity such that it also resolves sufficiently small communities as intuitively expected.

6 LITERATURE

[1] Albert-László Barabási. *Network Science*. Cambridge University Press, 2016. Also available online: `http://barabasi.com/networksciencebook/`.

[2] Santo Fortunato and Marc Barthélemy. Resolution limit in community detection. *PNAS*, volume 104 no. 1:36–41, 2007.

[3] Santo Fortunato and Claudio Castellano. Community structure in graphs. *Springer New York*, Computational Complexity: Theory, Techniques, and Applications:490–512, 2012.