Big Data Analysis and Data Science Master
Machine Learning Project

Roberto Gallea

October 24, 2020

Abstract

This report summarizes a study aimed to the evaluation of two optimization algorithms for the purpose of network communities detection. In particular, Simulated Annealing (SA) and Genetic Algorithm (GA) optimization have been implemented to detect two communities in a realistic network.
Algorithm implementation details using R language code are also shown.
The performance of the two methods was assessed using the modularity metric, a well-established method for measuring networks partition quality.

1 Introduction and Overview

Community Detection is one of the fundamental problems in network analysis, where the goal is to find groups of nodes that are, in some sense, more similar to each other than to the other nodes. Such similarity can be measured in several ways depending on the definition of "community". It may depend on the properties of the network (weighted, directed, etc.) or on the processes running on it (e.g. spreads of information).
One general structural way of defining a community is in terms of the density of edges between the nodes of each of the communities was introduced by Newman and Girvan [5]. Such measure is called *modularity*. One way of detecting communities across the network is to assign community mebmership do nodes such that modularity is maximized.

2 Methods

Consider a particular division of a network into K communities. Let us define a KxK symmetric matrix e whose element ei_j is the fraction of all edges in the network that link vertices in community i to vertices in community j. The trace of this matrix $Tr\ \ \mathbf{e} = \sum_i e_{ii}$ gives the fraction of edges in the network that connect vertices in the same community, and clearly a good division into communities should have a high value of this trace. The trace on its own, however, is not a good indicator of the quality of the division since, for example, placing all vertices in a single community would give the maximal value of $Tr\mathbf{e} = 1$ while giving no information about community structure at all. So the row (or column) sums $a_i = \sum_i e_{ij}$ is further defined, which represent the fraction of edges that connect to vertices in community i. In a network in which edges fall between vertices without regard for the communities they belong to, we would have $e_i = a_i * a_j$. Thus modularity measure is defined by:

$$Q = \sum_i \left(e_{ii} - a_i^2\right) = Tr\ \ \mathbf{e} - \left\|\mathbf{e}^2\right\| \tag{1}$$

Any method that effectively maximizes such measure could be a good candidate for community detection. In this report two algorithms are used: Simulated Annealing (SA) and Genetic Algorithm (GA).

2.1 Simulated Annealing

Simulated Annealing [4, 3] is a classical algorithm for addressing optimization of *non-convex* functions. This method simulates the annealing principle, i.e. a solid is heated at a high temperature and then cooled slowly

so that the system at any time is approximately in thermodynamic equilibrium. At equilibrium, there may be many configurations with each one corresponding to a specific energy level. The chance of accepting a change from the current configuration to a new configuration is related to the difference in energy between the two states. The simulated annealing strategy is widely applied to optimization problems.
The algorithmic key concept is to allow the current solution to be perturbed randomly in its domain, then a target function is evaluated, if a better value is found, the new position is kept, if the value is worse, there is still a probability to accept it anyway. Such probability is function of a time-decreasing heat function. This gives the optimization process the chance to escape from local minima/maxima. During simulation, such probability gets smaller, reaching total stability at the end of the simulation.

2.2 Genetic Algorithms

Genetic Algorithms [1] are a class of heuristic search methods inspired by the evolutionary principle. It reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation. This leads to a progressive fitter population. Relating the fitness measure to the object function of optimization leads to an optimum or sub-optimum solution. Basically, GA consists of five phases:

- *Initial population*: an initial population of size K is created, generally using random values. Each individual *chromosome* represents a point in the feasible solution space and is represented by a list of feature parameters, also called *genes*.
- *Fitness function*: this function determines the metric used to evaluate how fit a chromosome is. It provides a fitness score, which is related to the probability that each individual has to be selected for reproduction (highest fitness), or, conversely, to be removed from the population (lowest fitness).
- *Selection*: basing on the aforementioned probability, pairs of individuals, the *parents*, are selected for passing their genes to the next generation.
- *Crossover*: this is the crucial phase of the algorithm. For each pair of parents to be mated, a crossover point is chosen at random. Genes after crossover point are swapped between the parents to create the chromosome of the new generation individuals.
- *Mutation*: in the new offspring formed, some genes are given a random low probability to mutate. This implies that some genes of the chromosome can be flipped.

The algorithm terminates when the populations converges (i.e. new offspring are not significantly different from the previous generation), or when a maximum number of iteration is reached.

3 Algorithms Implementation

In this context, R was used as implementation language. SA algorithm was implemented *ad-hoc*, while for GA the library available from CRAN (Comprehensive R Archive Network) was used. The full source code is available on the github repository at the link https://github.com/unipa-bigdata/gallea-network-community-detection.

3.1 SA.R

SA.R is the SA algorithm function, it accepts six parameters:

- *network* - The network to use for community detection, represented by its adjacency matrix.
- T - Pair of initial and final temperatures, computed externally (see 3.2).
- *sol* - Initial solution vector.
- *nsim* - Number of simulation steps, default to 1000.

- *pchange* - expected ratio of perturbed nodes at each simulation steps, default to 0.1, see below.
- *metric* - An objective function used to evaluate the solution, in this context network modularity.
- *callback* - A callback function which is called at each simulation step, for example for showing algorithm progress.

In lines 2-10 the vector of temperature for each steps is computed from the initial and final temperatures using an exponential function.
After initialization (lines 12-17) the main loop of the process occurs (lines 20-52). Note that, due to oscillations, the final solution could not be the best one encountered. For this reason, best solution encountered is stored in a separate variable, which is also the actual final solution provided by the algorithm.
At each step, the solution is perturbed by flipping a random number of network nodes. Such number is obtained by sampling a poisson distribution, with *pchange* percentage of the total number of nodes as expected value.
The final result returned by the function consists of a list containing the following objects:

- *network.final* - the input network augmented with the community each node belongs to.
- *solution* - The best solution found.
- *modularity* - Vector of modularity across each simulation step.
- *best.modularity* - Vector of *best* modularity across each simulation step.
- *max.modularity* - Scalar containing the final modularity value.

3.2 estimateTemperatures.R

This function serves as a way of estimating the initial and final temperature for the SA. It draws N s_i random solution and computes their modularity m_i. Then, the mean distance of each possible solution pair is evaluated. Initial temperature is chosen as

$$T_0 = 100\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij} = 100\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} abs\left(m_i - m_j\right), \tag{2}$$

while final temperature is chosen as

$$T_f = 0.01\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij} = 0.01\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} abs\left(m_i - m_j\right). \tag{3}$$

3.3 Other files

Other code has been developed, for running multiple simulations and generating the plots in this report:

- *singleOptimization.R* - runs a single complete optimization over the network.
- *multiOptimization.R* - runs M complete optimization over the network, then computes some statistics and generate confidence plot for *modularity* and for *best.modularity*, jittered boxplot for *max.modularity* and histogram plot for *max.modularity*. Finally, it saves the network on a text file.
- *gaOptimization.R* - Executes islanded GA algorithm, plot the solution over time and saves the network on a text file.

4 Data and Results

In order to evaluate the SA and GA algorithms, tests were conducted over a mid-sized test network, aimed to determine the best decomposition in 2 communities.

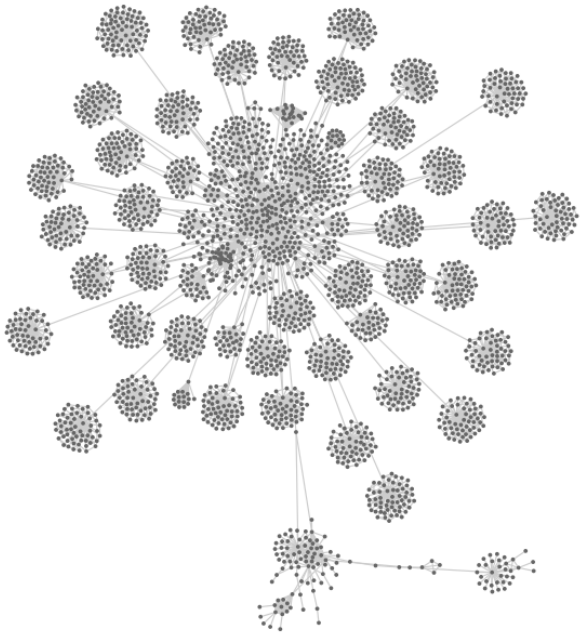


Figure 1: web-edu network topology.

Property	Value
Total degree	6475
Average degree	2.136
Minimum degree in	0
Minimum degree out	0
Maximum degree in	66
Maximum degree out	101
Total strength	12948
Average strength	4.272
Minimum strength in	0
Minimum strength out	0
Maximum strength in	6474
Maximum strength out	6475
Total self-loops strength	6474
Asymmetry	0.999
Reciprocity	0.0002
Average entropy in	0.848
Average entropy out	0.274
Assortativity	-0.136
Assortatitvity error	0.001
Average clustering coefficient	0.282
Efficiency	0.007

Table 1: web-edu network properties

4.1 Test network

As test network, the *web-edu* dataset was chosen from the NetworkRepository Archive [6]. Such network, shown in Figure 1, represents the map of the web.edu website, each node reprents a page and each edge an hyperlink between two pages. It is composed by 3031 nodes and 3475 edges, and exhibits the properties shown in Table 1 (summary generated by Radatools).
From the network topology is clear that many communities exist. However, showing that a dual-communities partitioning could be effectively performed, other communities could be found by bisection of each community. The process can be recursively applied until a community cannot be further divided.

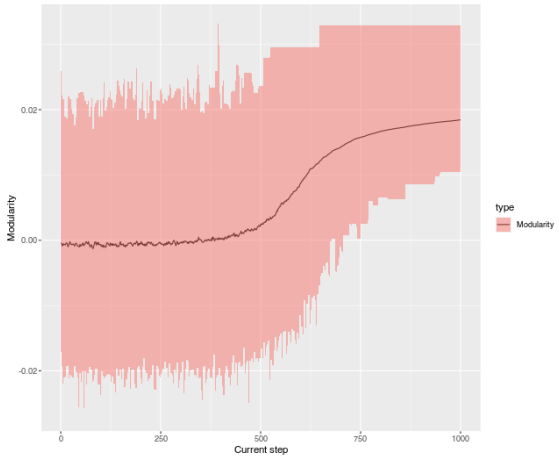
4.2 Experimental results - Simulated Annealing

The SA algorithm was run over 1000 simulations: mean, maximum and minimum values were obtained. Simulation results are depicted in Figures 2a and 2b. From Figure 2a, it can be seen how the algorithm oscillates in the early stages of the simulation, and then approaches convergence, in average, in the second half. However, in Figure 2b can be deducted how the final solution, for each simulation, is not very likely to be neither the same nor of the same quality. Indeed, the confidence interval is very large and keeps constant across the whole process.

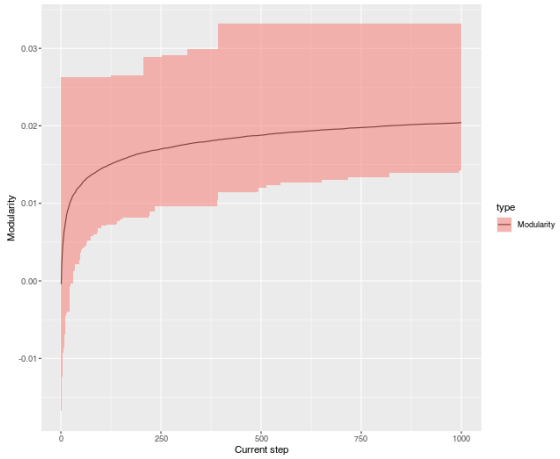
The same problem is put in evidence from the jittered boxplot and the histogram plot in Figure 3a and 3b, which show how the found solution is spread over a wide distribution.

4.3 Experimental results - Generic Algorithm

Binary GA was also run for optimizing the modularity of the test network. A population size of 50 individuals was used, with a probability of mutation of 0.1. Maximum number of iteration was set to 20000 steps. Also, an islanded approach was used [2]. This approach considers multiple isolated populations, which differentiates in their evolution. Also, individuals from islands, have a chance to migrate from one island to another after a number of iterations, bringing their genetic features to the new population. In this context, 4 islands were

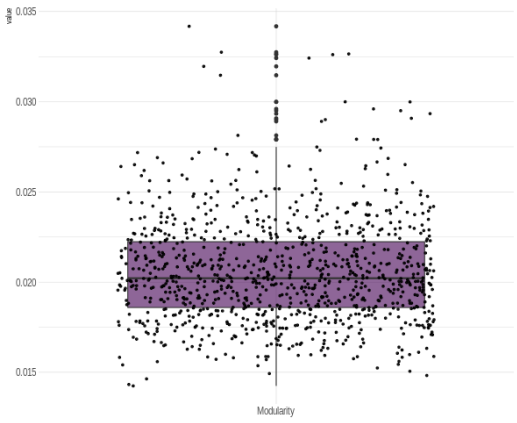


(a) Confidence range: current modularity vs step #

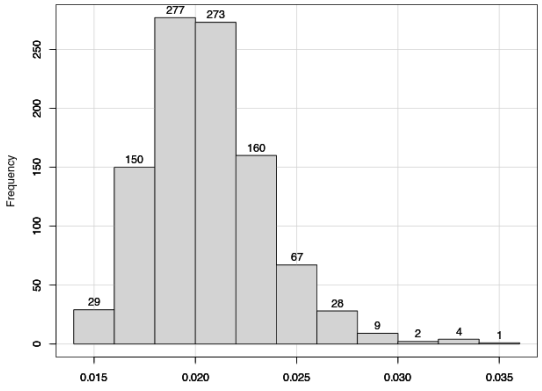


(b) Confidence range: best modularity vs step #

Figure 2: Confidence plot for simulated annealing results: current step modularity confidence range (a), best modularity confindence range (b).



(a) Current step modularity confidence range



(b) Histogram

Figure 3: Spreading of SA solution: final objective values are distributed over a large interval of the objective space, as can be seen from jittered boxplot (a) and histogram (b).

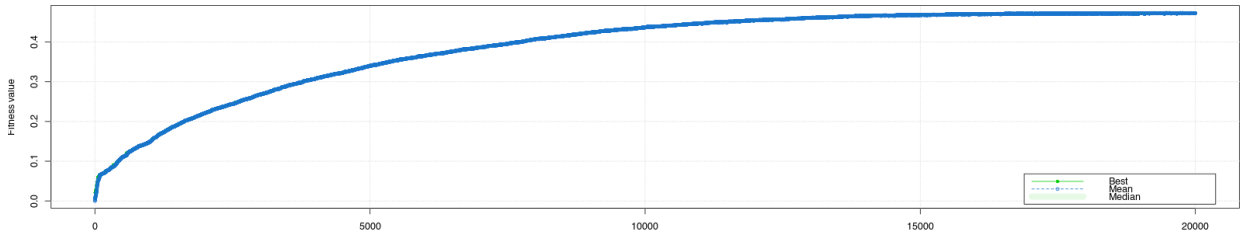
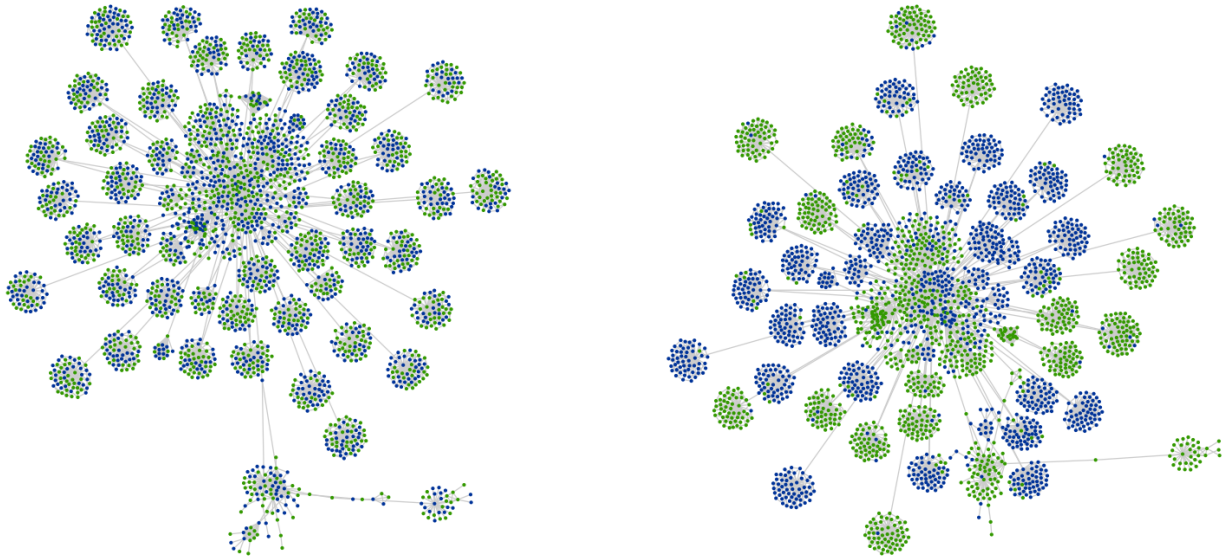


Figure 4: GA best solution plot.



(a) Communities using SA

(b) Communities using GA

Figure 5: Partitioned network using SA (a) and GA (b). Community membership is represented in colors.

used with a migration rate of 0.2 every 50 steps. Figure 4 depicts the best solution across the whole process, also showing how the fitness increases slowly but constantly.

4.4 Comparison

From a comparison of the two approaches, the quality of the solution found is numerically much more better using GA. However, such approach is very slow w.r.t. SA. Conversely, SA is very fast but fails in finding a good solution, at least for mid-sized network, like the one used. SA completes after $\sim$10 seconds, while GA reaches convergence after $\sim$2 hours.
As a final comparison, the partitioned network is shown in Figure 5 where is also visually evident how communities found by GA are more definite.

5 Summary and Conclusions

The report presented two approaches aimed to network communities detection. The first one is based on Simulated Annealing Optimization, the latter on Genetic Algorithm Optimization. The methods were applied to a mid-sized realistic network and results were presented. GA performed much better than SA, although it is very slow. An hybrid approach could be considered were SA is used to find an initial population for GA.

References

[1] James Barker. "Simulation of Genetic Systems by Automatic Digital Computers". In: *Australian Journal of Biological Sciences* 11 (Jan. 1958), pp. 603–612. DOI: 10.1071/BI9580603.

[2] Theodore C. Belding. "The Distributed Genetic Algorithm Revisited". In: *Proceedings of the 6th International Conference on Genetic Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, pp. 114–121. ISBN: 1558603700.

[3] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. "Optimization by simulated annealing". In: *SCIENCE* 220.4598 (1983), pp. 671–680.

[4] Nicholas Metropolis et al. "Equation of State Calculations by Fast Computing Machines". In: *The Journal of Chemical Physics* 21.6 (1953), pp. 1087–1092. DOI: 10.1063/1.1699114. eprint: https://doi.org/10.1063/1.1699114. URL: https://doi.org/10.1063/1.1699114.

[5] M. E. J. Newman and M. Girvan. "Finding and evaluating community structure in networks". In: *Phys. Rev. E* 69.2 (Feb. 2004), p. 026113. DOI: 10.1103/PhysRevE.69.026113. URL: http://link.aps.org/doi/10.1103/PhysRevE.69.026113.

[6] Ryan A. Rossi and Nesreen K. Ahmed. "The Network Data Repository with Interactive Graph Analytics and Visualization". In: *AAAI*. 2015. URL: http://networkrepository.com.