Modularity-based Community Detection Algorithms: a Comparative Analysis

Sorin Dragan
s.dragan@students.uu.nl
Utrecht University

15 June 2020

Abstract

Community structure is considered one of the most important tasks in the study of networks. This paper compares 4 different community detection algorithms based on modularity maximization (Clauset-Newman-Moore, Louvain, RenEEL, and a genetic algorithm). The tests are run both on simple graphs like Zachary's Karate Club graph and against the LFR benchmark on graphs. The analysis includes reporting of normalized mutual information, coverage values, performance values and the run time of the algorithms. As a result of the analysis, the newly introduced algorithm by Guo et al. (RenEEL) and Louvain had the best overall performance. RenEEL obtains a higher normalized mutual information on more random graphs (higher μ value), but is slower than Louvain. This implies a trade-off between a higher normalized mutual information and a lower execution time when choosing an algorithm. The genetic algorithm proposed by Tasgin et al. turnes out to be very slow and has a comparable performance with Clauset-Newman-Moore, but an inferior performance than Louvain and RenEEL.

1 Introduction

Community detection is a popular task in the interdisciplinary field of Network Science. Networks (or graphs) consist of a set of nodes and the existing connections (or edges) between the nodes. A subset of these nodes that has a high density of edges inside the subset and a much lower density of edges outside the subset is called a *community*.[7] Community detection aims to identify communities in a graph when its topology is given.[4]

Identifying communities is an important problem for multiple reasons. Knowing the communities of a graph offers information about the vertices inside each community. For example, if a vertex is central in the community it could be important for the stability of the community, while if a vertex stays on the border of a community this might imply a function of mediation between its community and the neighbouring communities. The community structure also tells a lot about the hierarchical organization of a network. Another reason that supports the importance of this task is the fact that real world networks also have communities. Organizations, social networks, protein interaction networks are all examples of real networks that have community structure.[4]

There are different classes of algorithms for community detection. Some of these classes include divisive algorithms, modularity-based algorithms, spectral algorithms, algorithms based on statistical inference etc..[4] In this paper the focus lays on modularity-based algorithms amongst which the following 4: Clauset-Newman-Moore[3], Louvain[2], Reduced network extremal ensemble learning scheme algorithm (RenEEL)[6], and a genetic algorithm for community detection introduced by Tasgin et al.[11] will be described and compared in detail.

Modularity is a quality function introduced by Girvan & Newman that helps to distinguish between a good or a poor partitioning in communities of a graph.[9] As described by Fortunato, modularity reveals the possible existence of clusters by comparing the edges density of a sub-graph with the edges density given by a random graph with the same characteristics (a random graph is expected not to have communities).[4] The formula for modularity can take the following form:

$$Q = \frac{1}{2m} \sum_{i,j} (A_{ij} - P_{ij}) \cdot \delta(i,j) \tag{1}$$

where m is the number of edges, i and j are any two vertices, A is the graph represented by an adjacency matrix, P stores the expected number of edges between every pair of nodes in the random model, and δ is the following function:

$$\delta(i,j) = \begin{cases} 1, & C(i) = C(j) \\ 0, & C(i) \neq C(j) \end{cases} \tag{2}$$

where $C(q)$ is the community of which q is a part.

The main flaw of modularity is called the *resolution limit*. If the partition with maximum modularity contains communities with a total degree of at most $\sqrt{m}$ (m is the number of edges), it cannot be known, based on modularity, whether there is definitely a single unique community or a combination of smaller interlinked communities. This translates in the conclusion that modularity optimization might be unable to detect small communities relative to the size of the graph.[4]

This paper aims to compare 4 community detection algorithms based on modularity maximization by reporting the execution time, coverage, performance, and normalized mutual information of each algorithm in tests on the well known Zachary's Karate Club graph [5], Caveman graphs, and multiple LFR benchmark graphs [8] with sizes between 250 and 3000 nodes. The algorithms are described in section 2. The experimental setting is further detailed in section 3. All algorithms are expected to yield a good performance on small graphs (Karate Club and small Caveman). A performance gap in favour of *RenEEL* is expected in tests on LFR Benchmark Graphs. In terms of speed, all algorithms except for the genetic algorithm are expected not to surpass 10 seconds in run time for small μ values. The run time gap between *RenEEL* and the greedy algoritms (*Louvain* and *CNM*) is anticipated to enlarge proportionally with the size of the graph. This is due to the ensemble method used by *RenEEL*.

2 Algorithms

Four modularity-based community detection algorithms have been tested. Two of them are the popular Clauset-Newman-Moore[3] and Louvain[2] algorithms which already had implementations in the *NetworkX* Python library. Due to this fact, the execution time of the previously mentioned algorithms rarely surpass 5 seconds even for LFR graphs with 3000 nodes and small μ. Another algorithm that is considered is the new "Reduced network extremal ensemble learning (RenEEL) scheme for community detection complex networks".[6] The authors of the paper provide a fast *C* implementation of the algorithm which I integrated with *Python*. The last algorithm included in the tests is a genetic algorithm introduced by Tasgin et al.[11] that had no publicly available implementation. My own implementation of the algorithm is available at `https://github.com/sorindragan/community-detection`. For speed purposes, the implementation of this algorithm uses *Cython* as a programming language. However, the execution time of the genetic algorithm is relatively high in comparison with the other 3 even on small graphs of 40 nodes. This is due to the large number of iterations that the algorithm requires in order to find a decent solution. A parallel implementation of the algorithm should be considered in order to increase the execution speed.

2.1 Clauset-Newman-Moore

This is a greedy algorithm proposed by Clauset, Newman and Moore (CNM).[3] The algorithm starts from an unconnected set of nodes and iteratively adds edges in order to achieve the highest increase in modularity at each iteration. The speed of the algorithm comes from the usage of efficient data structures (like max-heaps). The complexity of the algorithm is $O(Nlog^2N)$.[7]

2.2 Louvain

This is a multi-step algorithm proposed at *UCLouvain* by Blondel et al..[2] Initially, in this algorithm, each individual vertex is its own community. In the initial step, each individual vertex is merged to neighbouring communities by assuring the largest increase in modularity. In the second step, the same intuition applies, but at a different level. Instead of merging individual vertices, whole communities are merged based on modularity increase. This is done until the increase in modularity becomes negligible. The benefits of this algorithm are its simplicity and a generally good reported complexity of $O(m)$, where m is the number of edges.

2.3 RenEEL

This is a new extended approach introduced in 2019 by Guo et al.[6] The algorithm makes use of the machine learning paradigm of ensemble learning. It iteratively updates an ensemble of graph partitions based on modularity maximization. In each iteration, the vertices that are part of the same community in every ensemble partition are grouped and used to create a reduced network that will be further used to update the ensemble. According to the paper, the complexity of the algorithm is $O(k_{max}k'n^2ln(n))$. Here, k_{max} is the initial number of ensemble partitions of the graph and k' is the number of ensemble partitions in the reduced network. Both k_{max} and k' are parameters of the algorithm.[6] In the performed tests, k_{max} is set to 50 and k' is equal to 20. The code for the algorithm can be found at `https://github.com/kbassler/RenEEL-Modularity`.

2.4 Genetic Algorithm

This is a genetic algorithm based on modularity optimization described by Tasgin et al.[11] The idea is to apply the basic steps of a genetic algorithm: selection, crossover and mutation on a population of individuals that encode the partition of the graph. The initial population is formed by starting with a community for each node and randomly combining neighbouring vertices into the same community. The selection step chooses the top candidates based on the fitness function. In this case, the fitness function is represented by the modularity of the partition. The crossover operation is a one-way crossover in which all the values (communities) equal to a randomly selected vertex (position in the array encoding the partition) are transferred to a destination individual. Mutation takes 2 random vertices from an individual and assign them to the same community (one of the communities that initially contained one of the randomly selected vertices). The steps are repeated for g iterations.

The algorithm has various parameters: α is the rate of randomly combining individuals in the initial population, θ is the crossover rate, ξ is the mutation rate, β is the proportion of elite individuals kept in every generation, and r is a newly introduced parameter which represents the fraction of new random individuals (created in the same manner as the initial population) introduced in each generation. Based on empirical observations gathered from tests on LFR graphs of 250 nodes (with both $\mu = 0.1$ and $\mu = 0.4$), the parameters proposed in the paper were proved to yield poor performances. Thus, after several trials, the parameters presented in

Parameters	$\mu = 0.1$	$\mu = 0.4$
α	0.3	0.1
θ	0.5	0.4
ξ	0.05	0.05
β	0.02	0.4
r	0.5	0.5

Table 1: Parameters used for the genetic algorithm

Table 1 were chosen for the final experiment.

Some other modifications in the implementation introduce a shuffle of the population of individuals before the crossover step and an early stopping mechanism that activates when there are no improvements after 100 iterations. In the experiments, both the population size and the number of iterations are dependent on the number of nodes of the tested graph.

3 Experimental Setup

For this experiment, each algorithm is run on two sets of graphs. Every graph is created using the graph generators form the *NetworkX* library. The first set of graphs contains only two well-known graphs of small size. The two graphs are the Zachary's Karate Club graph first introduced by Girvan et al. [5] and a connected Caveman graph of 7 cliques, each having 3 nodes (this implies that each clique is a separate community). The second set of graphs is composed of several LFR Benchmark graphs of various sizes. These graphs introduced by Lancichinetti et al. consider inconsistencies relevant to both the degree and the community size of real world graphs.[8]

The parameters used for generation are enumerated next: $\tau_1 = 2$ is the power law exponent of the degree distribution, $\tau_2 = 1.1$ is the power law exponent of the community size distribution, $\mu = 0.1$ and $\mu = 0.4$ are fractions of intra-community edges incident for each node. A μ value of 0 would mean that all edges lie inside the communities, while a value of 1 would mean that all edges are situated between nodes from different communities. The choice of $\mu = 0.4$ tries to introduce more randomness into the graphs, so the community detection task becomes more difficult. Other parameters include $k_{min} = 20$ and $k_{max} = 50$ which are the minimum and, respectively, the maximum degree of the graph. The number of nodes in the graph varied according to the sequence: 250, 500, 1000, 1500, 2000, 2500, 3000. Figure 1 shows the relationship between the number of nodes and the number of edges of the LFR graphs with $\mu = 0.1$ and $\mu = 0.4$. The average degree when $\mu = 0.1$ is about 35, while the average degree in the $\mu = 0.4$ case is between 38 and 39.

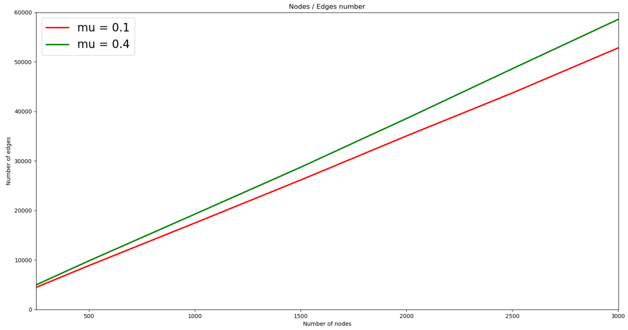

Figure 1: Relation between the number of nodes and number of edges in the LFR graphs used for tests

Testing algorithms on these graphs necessitates a way to estimate how good is the discovered community structure against the target community structure of the graphs. The similarity measure used in this paper is the *normalized mutual information* (NMI) described in Lancichinetti et al.[7]. This measure will return 0 if the partitions are independent and 1 if the partitions are identical. The formula presented for the NMI can be rewritten as:

$$I_n(X, Y) = 2 \cdot \frac{(H(X) - H(X|Y))}{H(X) + H(Y)} \qquad (3)$$

where $H(Z)$ is the entropy of Z and $H(Z|W)$ is the conditional entropy of Z given W.

Besides the NMI, the analysis also includes other two quality functions. These are *performance* (counts the pairs of connected vertices from the same community and the pairs of not connected vertices, each belonging to a different community; the counts are divided by the total number of edges) and *coverage* (the count of edges inside a community over the total number of edges).[4] The execution time of each algorithm[1] is also reported for each graph together with relevant pictures of the resulted graph partitioning in which each vertex is colored with the color of its community.

[1] The run time includes just the algorithm execution, ignoring other intermediary steps

The tests were run on an Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz Acer machine with 8Gb (7,63G showed by htop) RAM. It was empirically noticed that running the algorithms on LFR generated graphs with over 4500 nodes exceeded the RAM capacity of the computer.

4 Results

This section is split into two subsections. The first one deals with the results obtained on non-LFR small graphs. The second analyses the results obtained by running the algorithms on LFR Benchmark graphs with a size between 250 and 3000 nodes. It should be mentioned as a clarification that the names of the algorithms are used in short or abbreviated form throughout this section (both ”gcm” and ”GenCom” refers to the community detection genetic algorithm).

4.1 Small Graphs

This section begins by presenting the results of the algorithms on the previously mentioned Zachary's Karate Club graph. The partition returned by each algorithm can be visualized in Figure 2. Based on Agarwal & Kempe, the modularity of the optimal community structure for Zachary's Karate Club graph is 0.4197.[1] On this graph, *RenEEL* reaches the optimal modularity of 0.4197. *Louvain* reaches a modularity of 0.3942 in the graph from Figure 2. However, *Louvain* could also be seen reaching the optimal modularity during multiple runs on this graph. *CNM* reaches just 0.3806 while *GCM* yields slightly bellow with a modularity of 0.3717. Even if *GCM* fails in achieving a highest modularity, it is worth saying that on this graph, *GCM* is the closest to the true community structure of the original graph that also had only 2 communities (see Perry, 2018; Figure 1)[10]. Nevertheless, because of the randomness of the genetic algorithm, *GCM* is not always discovering just 2 communities.

Another interesting graph to look at is the Caveman graph consisting of 7 cliques of 3 nodes each presented in Figure 3. It is

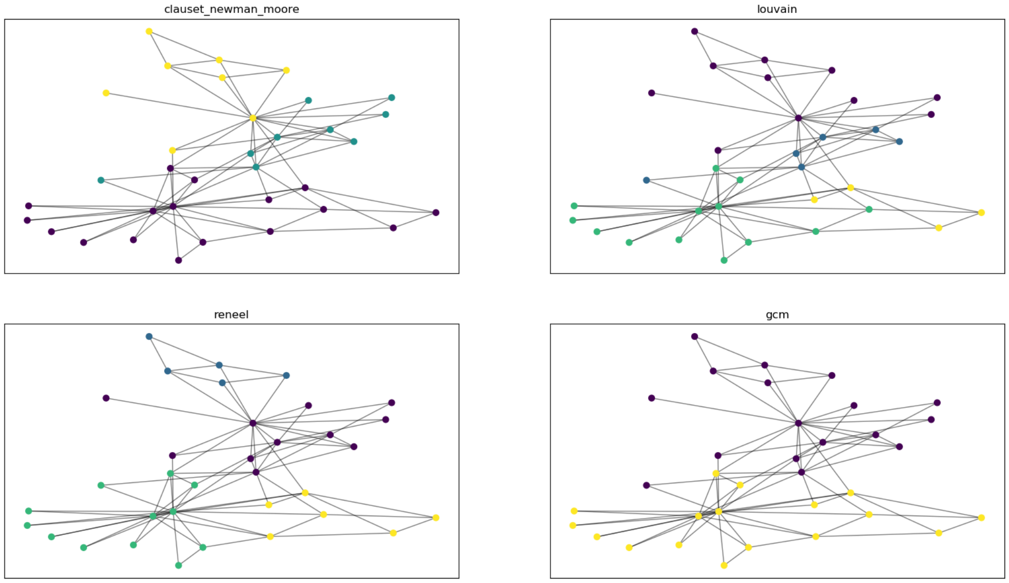

Figure 2: Community structure of Zachary's Karate Club graph (34 nodes; 78 edges) discovered by every tested algorithm

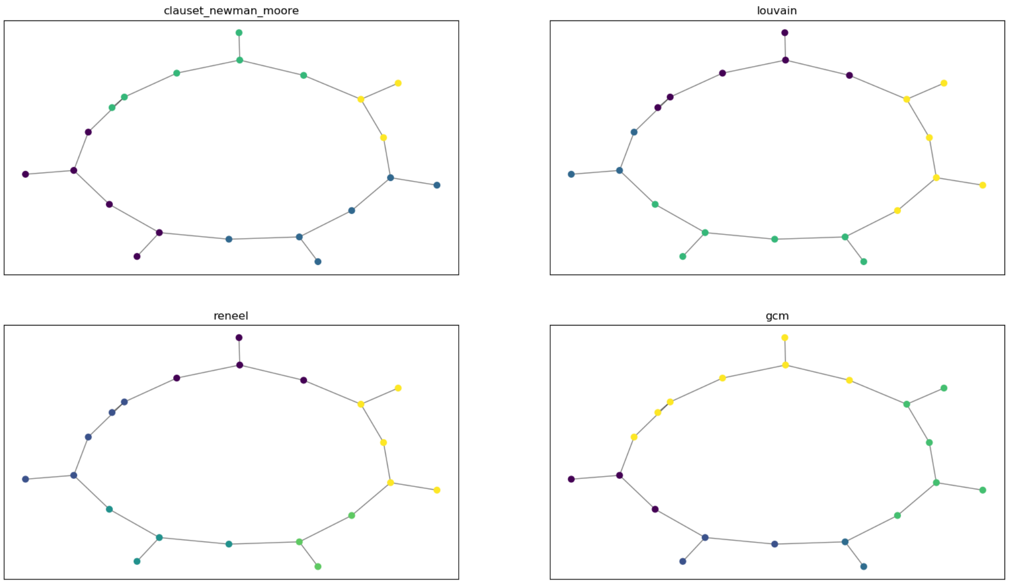

Figure 3: Community structure of Caveman graph with 21 nodes (7 cliques of 3 nodes) discovered by every tested algorithm

interesting to notice that *CNM* and *Louvain* always discovers 4 communities (modularity = 0.5442) and *RenEEL* almost always discovers 5 communities (modularity = 0.5555). *GCM*, however, partitions the graph in 4 to 6 communities durring different runs. In the graph presented in Figure 3 *GCM* also found only 4 communities (modularity = 0.5192). In this example, each algorithm came with reasonable community structure. This situation is hard to interpret due to the resolution limit problem, thus modularity maximization might not be the best choice in this situation. According to *coverage* of the graph, *CNM* and *Louvain* reach the highest value of 0.8095, while according to *performance*, the highest value of 0.8857 was reached by *RenEEL*. It is worth mentioning that the results on a Caveman graph of 4 cliques of 6 nodes each, all algorithms reach the same community structure.

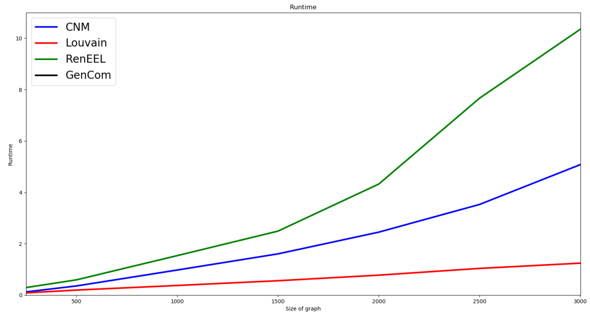

Figure 4: Run time of the algorithms in relation to the number of nodes in the LFR graphs with $\mu = 0.1$; *GenCom* is too slow to be represented at this scale

4.2 LFR Graphs

Tables 2 and 3 illustrate the results of the tests in terms of normalized mutual information. The parameters for each algorithm are the ones described in section 2. The performance[2] of *CNM*

[2] Here performance does not refer to the quality metric, but how good were the results in terms of NMI

decreases linearly with the size of the graph. *Louvain* and *RenEEL* have almost identical results for $\mu = 0.1$. The only difference lays in the 3^{rd} decimal point on graphs that have at least 2500 nodes. However, this comes at a cost of speed. Figure 4 shows the run time of 3 tested algorithms. For the largest graph of 3000 nodes, the run time of *Louvain* is under 2 seconds, while *RenEEL* takes 11 seconds to process the graph. It is evident that because the ensemble method, *RenEEL* is always slower that *Louvain*. In a different run of *RenEEL* with different parameters (a lower value of $k_{max} = 20$ and $k' = 10$), the NMI obtained on a large graph of 4000 nodes was equal to the one obtained by *Louvain*. This time, *RenEEL* processes the graph in 4.01 seconds, but still more than double the process time of *Louvain*. As a result, there is a trade-off between a NMI improvement in the 3^{rd} decimal and a lower run time when setting the parameters of *RenEEL*. The performance of *GCM* is independent of the graph size and comparable with the performance of *CNM* on large graphs. The *GCM* algorithm is also very slow, having a run time of 251 seconds on the 250 node graph and 3752 on the 3000 node graph. This suggests an execution time roughly the same as the number of nodes in the processed graph when measured in seconds.

Size	**CNM**	**L**	**RenEEL**	**GCM**
250	0.99	1.0	1.0	0.81
500	0.94	1.0	1.0	0.79
1000	0.84	1.0	1.0	0.80
1500	0.86	1.0	1.0	0.82
2000	0.83	1.0	1.0	0.81
2500	0.82	0.995	0.995	0.81
3000	0.79	0.994	0.996	0.80

Table 2: NMI of the community structures found by the 4 algorithms on the LFR graphs with $\mu = 0.1$

Size	**CNM**	**L**	**RenEEL**	**GCM**
250	0.43	0.839	0.868	0.33
500	0.44	0.886	0.927	0.39
1000	0.49	0.952	0.963	0.52
1500	0.45	0.949	0.937	0.49
2000	0.42	0.931	0.931	0.56
2500	0.42	0.914	0.916	0.54
3000	0.39	0.921	0.925	0.55

Table 3: NMI of the community structures found by the 4 algorithms on the LFR graphs with $\mu = 0.4$

In the tests on LFR graphs with $\mu = 0.4$, *RenEEL* has visibly higher overall performance than *Louvain* (the only exception is $n = 1500$). This shows that the ensemble method appears to be useful when there is more randomness in the graph, so the community structure is harder to detect. However, the trade-off between speed and performance still exists, but with a larger performance gap in this case. On the same experimental setting, *CNM* cannot reach a NMI value over 0.5. *GCM* has a very poor performance on small graphs (most probably because of the early stopping mechanism), but exceeds *CNM* on larger graphs. The run time of *GCM* also increases considerably. Figure 5 captures this change.

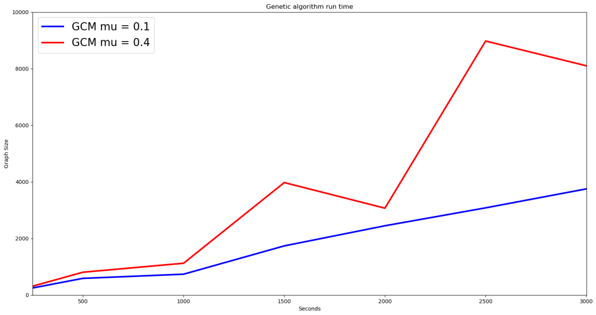

Figure 5: Run time of the genetic algorithm in relation to the number of nodes in the LFR graphs with $\mu = 0.1$ and $\mu = 0.4$

Figures 6 and 7 show the performance and coverage quality metrics described in section 3. Both of these metrics are stable. Performance has a high value across all graph sizes (in the case when $\mu = 0.4$) for 3 out of 4 algorithms. *CNM* yields the highest coverage, but the lowest performance. This might be because the partition discovered by *CNM* has a lot of inter-community edges which do not affect the coverage, but sig-

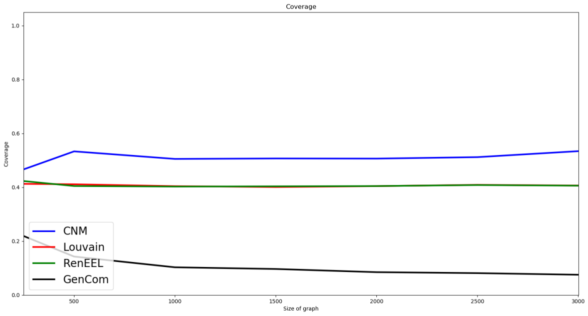

Figure 6: Coverage of the community structures returned by the algorithms in relation to the size of the LFR graphs with $\mu = 0.4$

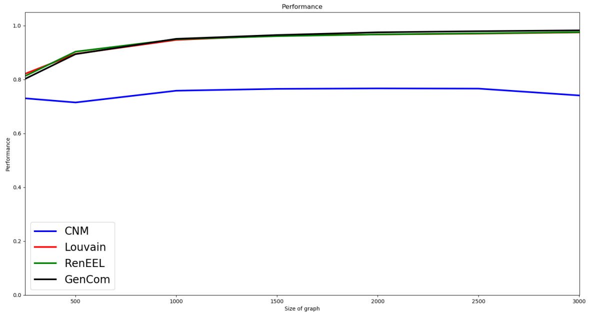

Figure 7: Performance quality metric of the community structures returned by the algorithms in relation to the size of the LFR graphs with $\mu = 0.4$

nificantly lowers the value of performance. Other exception is the genetic algorithm where the performance value slowly increases with the size of the graph while the coverage value decreases inverse proportionally with the size of the graph (both changes happen gradually). This might be due the increase of nodes. As the graph is not a fully connected graph, the increase in the number of nodes also results in an increases in the amount of pairs of nodes from different communities that do not have an edge between them. These pairs will be counted and will contribute to the calculation of the performance quality metric. On the other side, these pairs do not contribute in the calculation of coverage. Under the same circumstances, the number of edges that are strictly inside a community will decrease making the value of coverage to drop under 0.2 and even lower.

5 Conclusion

This paper presented a comparative analysis of 4 different community detection algorithms based on modularity maximization. The tests were carried out mainly on LFR Benchmark graphs of sizes up to 3000 vertices. Separately, some tests were also run on the Zachary's Karate Club graph and several Caveman graph configurations of which only one interesting one was presented.

Overall, *RenEEL* seems to reach the highest normalized mutual information independent of the graph size surpassing *Louvain* for generated graphs with $\mu = 0.4$. Both of them visibly outperform *CNM* and *GCM*. Nevertheless, *RenEEL* is slower than *Louvain* and *CNM* because of the k_{max} and k' parameters which could be set lower at the cost of performance. Both *RenEEL* and *Louvain* show a good balance of performance and speed on the LFR graphs with $\mu = 0.1$ with *RenEEL* coming on top when the underlying community structure of the graphs is harder to detect ($\mu = 0.4$). The *GenCom* algorithm is an interesting theoretical approach to community detection, but it turns out to be unfeasible and impossible to scale on relatively large graphs. The algorithm is very slow and the overall final graph partition is still worst than the one discovered by *Louvain* and *RenEEL*.

It must be mentioned that the results are heavily influenced by the parameters set for the graph generators and algorithms in discussion. The present analysis could be extended on graphs with different values of τ, minimum and maximum degrees. Another factor that influenced the results was the vastly different implementations of the algorithms as two algorithms were used from the *NetworkX* library, one had a *C* implementation, and another one was implemented in a *Cython*. A re-implementation together with a close analysis of parameter selection is required for a better unbiased comparison.

References

[1] Gaurav Agarwal and David Kempe. “Modularity-Maximizing Network Communities via Mathematical Programming”. In: *The European Physical Journal B* 66.3 (Dec. 2008), pp. 409–418. ISSN: 1434-6028, 1434-6036. DOI: 10.1140/epjb/e2008-00425-1. arXiv: 0710.2533. URL: http://arxiv.org/abs/0710.2533 (visited on 06/01/2020).

[2] Vincent D Blondel et al. “Fast unfolding of communities in large networks”. In: *Journal of Statistical Mechanics: Theory and Experiment* 2008.10 (Oct. 9, 2008), P10008. ISSN: 1742-5468. DOI: 10.1088/1742-5468/2008/10/P10008. URL: https://iopscience.iop.org/article/10.1088/1742-5468/2008/10/P10008 (visited on 05/29/2020).

[3] Aaron Clauset, M. E. J. Newman, and Cristopher Moore. “Finding community structure in very large networks”. In: *Physical Review E* 70.6 (Dec. 6, 2004), p. 066111. ISSN: 1539-3755, 1550-2376. DOI: 10.1103/PhysRevE.70.066111. URL: https://link.aps.org/doi/10.1103/PhysRevE.70.066111 (visited on 05/29/2020).

[4] Santo Fortunato. “Community detection in graphs”. In: *Physics Reports* 486.3 (Feb. 2010), pp. 75–174. ISSN: 03701573. DOI: 10.1016/j.physrep.2009.11.002. arXiv: 0906.0612. URL: http://arxiv.org/abs/0906.0612 (visited on 05/04/2020).

[5] M. Girvan and M. E. J. Newman. “Community structure in social and biological networks”. In: *Proceedings of the National Academy of Sciences* 99.12 (June 11, 2002), pp. 7821–7826. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.122653799. URL: http://www.pnas.org/cgi/doi/10.1073/pnas.122653799 (visited on 05/30/2020).

[6] Jiahao Guo, Pramesh Singh, and Kevin E. Bassler. “Reduced network extremal ensemble learning (RenEEL) scheme for community detection in complex networks”. In: *Scientific Reports* 9.1 (Dec. 2019), p. 14234. ISSN: 2045-2322. DOI: 10.1038/s41598-019-50739-3. URL: http://www.nature.com/articles/s41598-019-50739-3 (visited on 05/27/2020).

[7] Andrea Lancichinetti and Santo Fortunato. “Community detection algorithms: a comparative analysis”. In: *Physical Review E* 80.5 (Nov. 30, 2009), p. 056117. ISSN: 1539-3755, 1550-2376. DOI: 10.1103/PhysRevE.80.056117. arXiv: 0908.1062. URL: http://arxiv.org/abs/0908.1062 (visited on 04/29/2020).

[8] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. “Benchmark graphs for testing community detection algorithms”. In: *Physical Review E* 78.4 (Oct. 24, 2008), p. 046110. ISSN: 1539-3755, 1550-2376. DOI: 10.1103/PhysRevE.78.046110. arXiv: 0805.4770. URL: http://arxiv.org/abs/0805.4770 (visited on 05/27/2020).

[9] M. E. J. Newman and M. Girvan. “Finding and evaluating community structure in networks”. In: *Physical Review E* 69.2 (Feb. 26, 2004), p. 026113. ISSN: 1539-3755, 1550-2376. DOI: 10.1103/PhysRevE.69.026113. arXiv: cond-mat/0308217. URL: http://arxiv.org/abs/cond-mat/0308217 (visited on 05/30/2020).

[10] Marcus B. Perry. “On the detection of transitive clusters in undirected networks”. In: *Journal of Applied Statistics* 46.2 (Jan. 25, 2019), pp. 364–384. ISSN: 0266-4763, 1360-0532. DOI: 10.1080/02664763.2018.1491535. URL: https://www.tandfonline.com/doi/full/10.1080/02664763.2018.1491535 (visited on 06/01/2020).

[11] Mursel Tasgin, Amac Herdagdelen, and Haluk Bingol. "Community Detection in Complex Networks Using Genetic Algorithms". In: *arXiv:0711.0491 [physics]* (Nov. 3, 2007). arXiv: `0711.0491`. URL: `http://arxiv.org/abs/0711.0491` (visited on 05/27/2020).