An Immunological Algorithm for Graph Modularity Optimization

S. Cavallaro[2], V. Cutello[1], M. Pavone[1], R.A. Scollo[1], and A.G. Spampinato[1,2]

[1] Department of Mathematics and Computer Science
University of Catania
V.le A. Doria 6, 95125 Catania, Italy.
cutello@unict.it, mpavone@dmi.unict.it

[2] Institute for Biomedical Reasearch and Innovation
Italian National Research Council
Via P. Gaifami 18, 95126 Catania, Italy.
sebastiano.cavallaro@cnr.it

Abstract. Complex networks constitute the backbone of complex systems. They represent a powerful interpretation tool for describing and analyzing many different kinds of systems from biology, economics, engineering and social networks. Uncovering the community structure exhibited by real networks is a crucial step towards a better understanding of complex systems, revealing the internal organization of nodes. However, existing algorithms in the literature up-to-date present several crucial issues, and the question of how good an algorithm is, with respect to others, is still open. Recently, Newman [14] suggested modularity as a natural measure of the goodness of network community decompositions. Here we propose an implementation of an Immunological Algorithm, a population based computational systems inspired by the immune system and its features, to perform community detection on the methods of modularity maximization. The reliability and efficiency of the proposed algorithm has been validate by comparing it with Louvain algorithm one of the fastest and the popular algorithm based on a multiscale modularity optimization scheme.

Keywords: Immunological-inspired algorithms, networks, modularity optimization, community structure, opt-IA

1 Introduction

In modern interdisciplinary science, networks (graphs) are an extremely useful for the representation of a wide number of complex systems. A large variety of natural processes can be conveniently described and studied using graphs, where nodes are the elementary parts of the system and edges between them represent their mutual interactions [10, 3]. Usually complex systems are organized in compartments, where each of them has a role and/or a function that satisfy a certain property of relative cohesiveness. In the context of the theory of complex networks, compartments are represented by partitions of the set of nodes with a high density of internal links (whereas links between compartments have a comparatively lower density), called communities or modules [8, 6]. Finding compartments in a graph-theoretic background, has become a fundamental problem in network science, since it may shed some light on the organization of complex systems and on their function. Different compartments often exhibit significantly different properties, therefore a global analysis of the network would be inappropriate and unfeasible. A detailed analysis of individual communities, instead, leads to more meaningful insights into the roles of individuals. Such an approach can also allow the visualization and the analysis of a large and complex network focusing on a new higher-level structure, where each identified communities can be compressed into a node belonging to the latter. Let us underline that classical algorithms for graph clustering are not suitable to reveal the properties of community structures. They are mostly based on optimal subdivisions of graphs in order to guarantee min-flow cut. Finding properties of community structures, instead, requires a complex analysis of link patterns and relations.

In the last few years, a very large amount of new computational techniques (often called network clustering), have been developed and are commonly used for community detection in graphs as well as to optimize a graph structure so to guarantee certain desired features,[6, 15, 13, 5]. Furthermore, many approaches have been proposed for finding such partitions and some different metrics for community structure evaluation have been introduced [12]. Wether or not to search for a hierarchical partition, where the communities are recursively subdivided into sub-communities, as well as the definition of the size of the communities (specified by the user or derived by the algorithm) along with other parameters, are the substantial differences between the proposed approaches.

In this work, we propose an implementation of an Immunological Algorithm, a population based computational systems inspired by the immune system and its features, to perform community detection on the methods of modularity maximization. To evaluate its reliability and efficiency, the proposed algorithm has been compared with Louvain's algorithm [4], one of the fastest and popular community detection methods in networks [2] which uses a multiscale modularity optimization scheme in order to maximize a modularity score for each community.

2 Community detection and modularity maximization

Modularity is a benefit function that measures the quality of a particular partitioning of a graph into communities, and it was proposed by Newman [14]. Originally defined for undirected graphs, the definition of modularity has been subsequently extended to directed and weighted graphs [11, 9, 1]. The aim of community detection in graphs is to identify, by using only the information encoded in the graph topology, the modules and their hierarchical organization. Modularity maximization is the most popular and one of most widely used methods for community partition. It detects communities by searching over possible partitions of a graph, over which modularity is maximized. Given a subgraph, a.k.a. a cluster, the modularity function is defined as the difference between the actual density of edges inside the cluster and the expected density of such edges if the graph was random conditioned on its degree distribution [14]. This expected edge density depends on the chosen null model, a random copy of the original graph that maintains the structural properties but not those on the structure of the communities. The idea behind modularity is that a random graph does not have a clustering structure. The edge density of a cluster should be greater than the expected density of a subgraph whose nodes are randomly connected.

Given a graph $G = (V, E)$ with $|E| = m$, and given a partition of G with n_c clusters, the benefit function of modularity can be written as:

$$Q = \sum_{c=1}^{n_c} \left[\frac{l_c}{m} - \left(\frac{d_c}{2m} \right)^2 \right] \tag{1}$$

where, for each cluster, i.e. subgraphs c,

- l_c is the total number of edges and
- d_c is the sum of the degrees of its vertices,
- $\left(\frac{l_c}{m}\right)$ represents the fraction of edges inside a certain cluster and
- $\left(\frac{d_c}{2m}\right)^2$ the fraction of the expected edges if the graph was random (null model).

Although an important limit of resolution of the modularity measure has been underlined by Fortunato and Barthelemy [7], modularity seems to be a useful measure of the community structures. In fact, algorithms that search for graph partitions that offer optimal modularity are already proposed, generally claim to be able to successfully find communities in very large and complex networks [12, 11].

Unfortunately, in modularity optimization methods, overlapping between the communities is not allowed, i.e. each vertex of the graph can be inserted in a single community. Furthermore, it is possible to discover sub-communities by applying these algorithms iteratively, but it is not possible to discover partially overlapping communities. The modularity defined in a heuristic way, considers a good division one which places most of the edges of a network within groups and only some of them between groups. High values of modularity indicate good partitions. In particular, we desire a quality function Q which, given a network and a candidate division of that network into groups, assigns a score to each partition of a graph,so to rank partitions and evaluate when a partition is better than another (in a graph the partition corresponding to its maximum value should be the best, or at least a very good one). Maximization of modularity is therefore sought at all costs. Obviously, a brute force search to optimize Q is impossible, due to the enormous number of ways in which it is possible to partition a graph. Moreover, it has been proven that the optimization of modularity is an NP-complete problem [4], so it is highly unlikely to perform the optimization task and find an optimal solution in polynomial with with respect to the graph dimension. Therefore, we need to turn to approximate methods of optimization, that can find fairly good approximations of maximum modularity in a reasonable time, through the use of appropriate algorithms particularly suitable when the solution space of a given problem is very large, and an exhaustive brute force search for the optimal solution is unfeasible.

3 The Immunological Algorithm

Immunological-inspired computation is nowadays a wide family of successful algorithms in searching and optimization that takes inspiration from the immune system (IS). What makes the IS interesting and source of inspiration is its defence dynamics and features that allow it to be able to protect living organisms against invaders and diseases. It is also a robust and efficient recognition system, able to detect and recognize the invaders, and distinguish between own cells, and foreign ones (*self/nonself discrimination*). Other really interesting and inspiring features that allow to design efficient solving methodologies are its high ability to learn, memory usage; self-regulation; associative retrieval; threshold mechanism, and the ability to perform parallel, and distributed cognitive tasks. In light of all the above, immunological heuristics are mainly focused on three general approaches: (1) clonal selection [**?**,**?**] (2) negative selection [**?**]; and (3) immune networks [**?**]. Such algorithms have been successfully employed in a variety of different application areas.

In this research work, we have developed an immunological algorithm inspired by the clonal selection theory, which belongs to a special class of the immunological heuristics family called *Clonal Selection Algorithms* (CSA) [**?**,**?**,**?**,**?**]. The developed algorithm is based on two main concepts/entities:

– antigen (Ag), i.e. the optimization problem to be solved, and
– B cell receptor, i.e. a point in the search space (solutions) for the problem Ag.

The major features and points of strength of this kind of heuristics are the operators: (*i*) cloning, (*ii*) inversely proportional hypermutation and (*iii*) aging. The first operator generates a new population of B cells centered on the highest affinity/fitness values; the second one explores the neighbourhood of each point of the search space, perturbing each solution with a probability which is inversely proportional to its fitness function; and the last one eliminates old solutions from the current population with the goal of introducing diversity and avoiding local minima during the evolutionary search process.

For simplicity, hereafter, we call the algorithm as Opt-IA. At each time step t Opt-IA maintains a population of B cells $P^{(t)}$ of size d (i.e., d candidate solutions). Each element of the population represents a possible subdivision of the vertices of the graph $G(V, E)$ in the community. More in details, if N is the cardinality of the set of vertices V, a B cell c will be a sequence (array) of n integers, between 1 and N, where $c[i] = j$ represents the fact that the vertex i is placed in the cluster j.

These elements are randomly initialized at the time step $t = 0$, using the uniform probability distribution. Once the population is initialized, the next step is to evaluate the fitness function for each B cell $\mathbf{x} \in P^{(t)}$ using the function *Compute_Fitness($P^{(t)}$)* which computes for each B cell c the value given by equation 1.

A description of Opt-IA is presented in the pseudocode shown in Algorithm 1. The rest of the section describes the main steps performed by the immunological algorithm (algorithm 1).

Algorithm 1 Opt-IA (d, dup, ρ, τ)

```
1: $t \leftarrow 0$
2: $P^{(t)} \leftarrow$ Initialize_Population($d$);
3: Compute_Fitness($P^{(t)}$)
4: repeat
5:     $P^{(clo)} \leftarrow$ Cloning($P^{(t)}, dup$);
6:     $P^{(hyp)} \leftarrow$ Hypermutation($P^{(clo)}, \rho$);
7:     Compute_Fitness($P^{(hyp)}$)
8:     $P_p^{(t)} \leftarrow$ Precompetition($P^{(t)}$);
9:     $P_a^{(t)} \leftarrow$ Aging_Random($P_p^{(t)}, \tau$);
10:    $P^{(t+1)} \leftarrow$ Selection($P_a^{(t)}, P^{(hyp)}$)
11:    $t \leftarrow t + 1$
12: until (termination criterion is satisfied)
```

Initialization phase: Each element of the population, denoted by $P^{(t)}$ in the algorithm 1, represents a possible subdivision of the vertices of the graph $G(V, E)$ in the community. In the initialization phase ($t = 0$), the elements of the population are randomly generated, assigning to the vertices a number included in the interval $[1, N]$, where $N = |V|$. The assigned number is the cluster number to which the vertex is assigned.

Cloning operator: The cloning operator has the simple purpose to duplicate *dup* times the elements of the population, creating an intermediate population $P^{(clo)}$ of dimensions $d \times dup$. To avoid a premature convergence of the algorithm, we made *dup* independent from the fitness of the element, If we had chosen to increase the number of clones for high fitness elements, we would have obtained quickly a very homogeneous population, causing in turn a poor exploration of the search space.

Hypermutation operators: The *hypermutation operator* acts on each element of the population $P^{(clo)}$ performing ρ mutations, where as for *dup*, ρ is a constant determined by the user. Again, we wanted to avoid a premature convergence of the algorithm, and so the mutation rate (ρ) does not depend upon the fitness value. The cloning operator, coupled with the hypermutation operator, performs a local search around the cloned solutions. The introduction of blind mutation produces individuals with higher affinity (i.e. higher fitness function values), which will be then selected forming the improved mature progenies. We considered different types of mutation operators which can act on a single vertex of the sequence (*local operators*) or on a group of nodes (*global operators*). Among them, we distinguish:

- **TotalRandom:** randomly selects a vertex of the solution and randomly assigns it to a cluster among the N possible.
- **Equiprobality:** randomly selects a vertex of the solution and assigns it to a cluster among those existing at time t. Each cluster has the same probability of being selected.
- **Existing:** randomly selects a vertex of the solution and assigns it to a cluster among those existing at time t. The probability with which a cluster is selected is proportionate to its size: the larger the cluster, the greater the probability that the vertex will be assigned to that cluster.
- **Fuse:** randomly selects a solution cluster and assigns all its nodes to a randomly selected cluster among those existing at time t.
- **Destroy:** randomly selects a cluster of the solution and assigns a variable percentage of its nodes to clusters chosen at random from the N possible.

Precompetition: This function randomly chooses two individuals from the population and, if they have the same cluster number, it removes the lower fitness element with a 50% probability. This strategy makes it possible to maintain a more heterogeneous population during the evolutionary cycle, maintaining solutions with a different number of communities, so as to better explore the research space.

Random aging operator: To help the algorithm escape local maxima, we introduced a random aging operator. In details, at each iteration the elements of the population are removed with a given probability τ. The aging operator reduces premature convergences and keeps high diversity into the population.

Selection operator: at this point the new population $P^{(t+1)}$ is created for the next generation by using $(\mu+\lambda)$-*Selection operator*, which selects the best d survivors of the aging step from the populations $P^{(t)}$ and $P^{(hyp)}$.. Such operator, with $\mu = d$ and $\lambda = (d \times dup)$, reduces the offspring B cell population of size $\lambda \geq \mu$ – created by cloning and hypermutation operators – to a new parent population of size $\mu = d$. The selection operator identifies the d best elements from the offspring set and the old parent B cells, thus guaranteeing monotonicity in the evolution dynamics.

Termination: Finally, the algorithm terminates its execution when the termination criterion is satisfied. In this research work, a maximum number of the fitness function evaluations FFE_{Max} has been considered for all experiments performed.

4 Results

In this section, we present the results obtained by OPT-IA, showing the competitiveness of the proposed approach with respect to the state-of the-art. In particular, to properly evaluate the performance of our proposed Clonal algorithm, we compared it with the deterministic Louvain algorithm, whic is able to obtain good results in reasonable times. The analysis was conducted on a series of networks, most commonly used as a benchmark in community detection in graphs, whose set includes *dolphins*, *karate* and *ukfaculty*. Before running our algorithm, we need to set both the population size and the number of clones, i.e. the two fundamental parameters of the algorithm, *d* and *dup*. To determine the best values to assign to them, we studied the fitness trend as they varied. The points visible in the figure 1 represent the average values on *10 runs* relative to the graph *almost_lattice*, chosen for its particularly complex landscape.

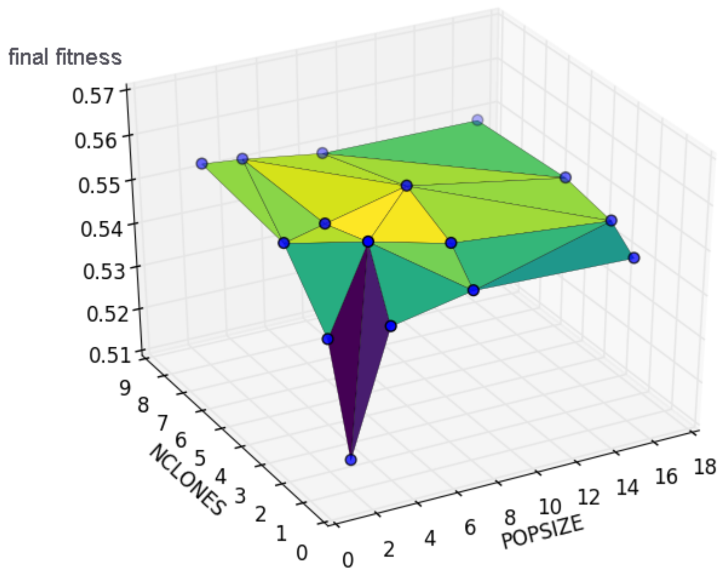

Fig. 1. Dependence of the algorithm on the parameters for *mut_rate = 3*, on *almost_lattice* graph

The best combinations of *popsize* and *clone number* are respectively $d = 8$ and $dup = 4$. The comparison between them, carried out before the choice of the most advantageous combinations, is shown in figure 2, corresponding to three different values of the mutation rate ρ. It is clear that both *fuse* and *total_random* do not offer good results. The first one leads to premature convergence of fitness, while the second one, at each cycle, modifies the communities of a single element with few improvements in fitness. On the other hand, *equiprobality*, *existing* and *destroy* are very effective functions. The latter allows to escape from local optima due to the presence of isolated nodes. In light of the tests carried out, the *equiprobality* and *destroy* operators were used with the same probability, while the *fuse* operator was used with low probability in order to have a cluster aggregation tool that allows to compete if minimally with the *destroy* operator.

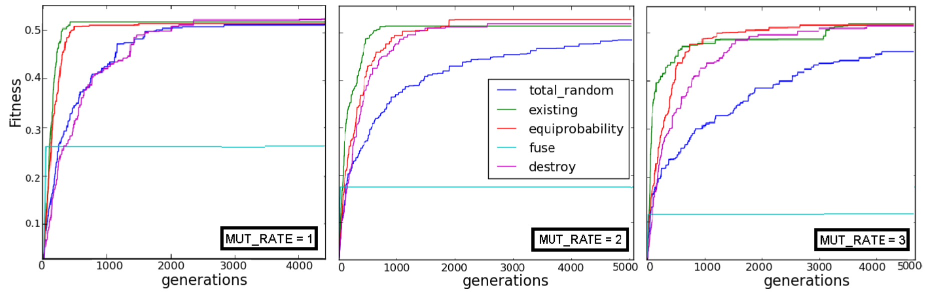

Fig. 2. Comparison of mutations for $MUT\_RATE = 1, 2, 3$ in the *dolphins* graph

The choice of mutation rate was made on the *almost_lattice* instance by varying ρ (from 1 to 5). It was observed that the best results were obtained for $\rho = \{1, 2, 3\}$ (figure 3). Thus, at each t iteration, we chose to mutate the cloned elements a number of times included between $[1, \rho]$.

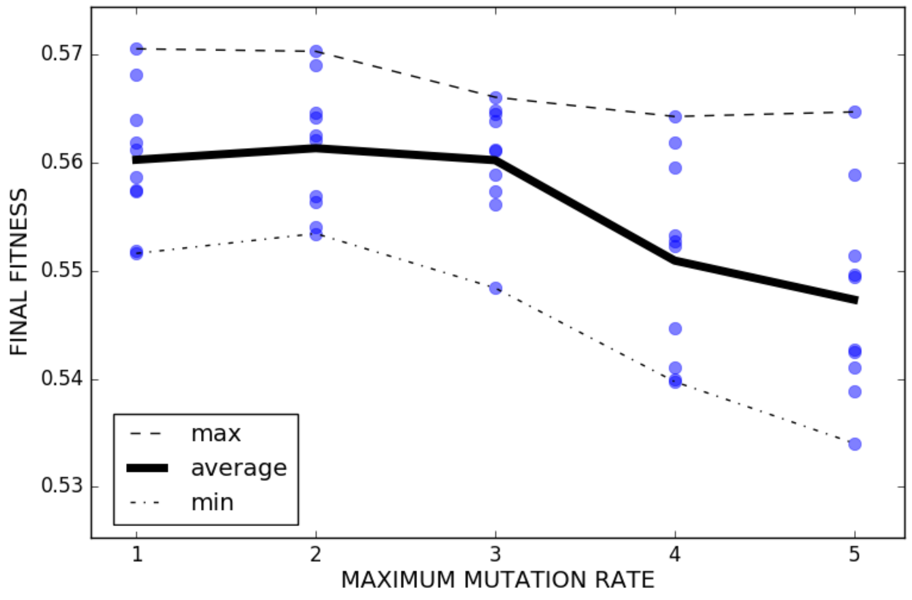

Fig. 3. Value of Fitness when MUT_RATE changes in *almost_lattice* graph

For each instance, the algorithm was executed *10 times*, and that's the reason why the best result obtained (max) is reported, with the relative number of communities, the worst (min) and the average with the standard deviation. A tuning was performed on the parameter τ used by the aging operator. The graph in the figure 4, shows the trend of the best individual for each iteration as the parameter τ varies: the values 1 and 0.001 represent the two extreme cases, that is the one in which all the elements or no element is discarded by the population; the intermediate values instead introduce a turnover among the elements of the population, producing diversity within the same population and avoiding that the algorithm converges in a premature way. The effectiveness of the operator

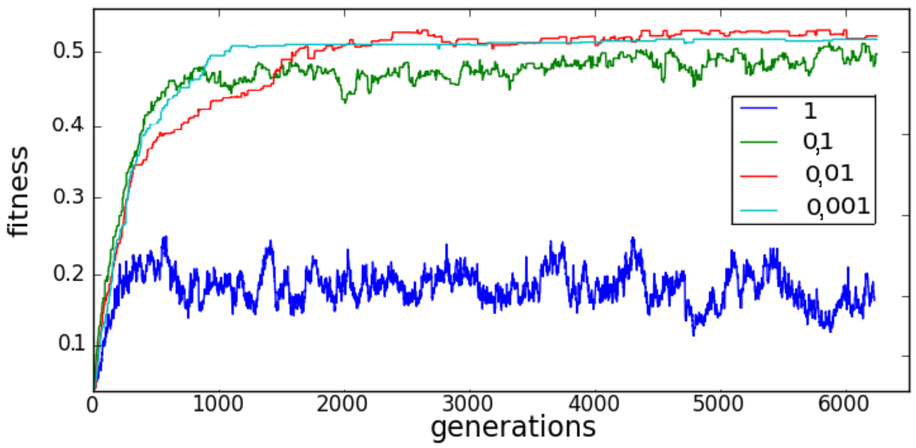

Fig. 4. Best fitness at every generation for various *RANDOM_DIE* action probabilities in the *dolphins* graph

is shown in the chart 5, in which the loss of the element with the best fitness corresponds, a few generations later, to the discovery of solutions with a higher fitness value.

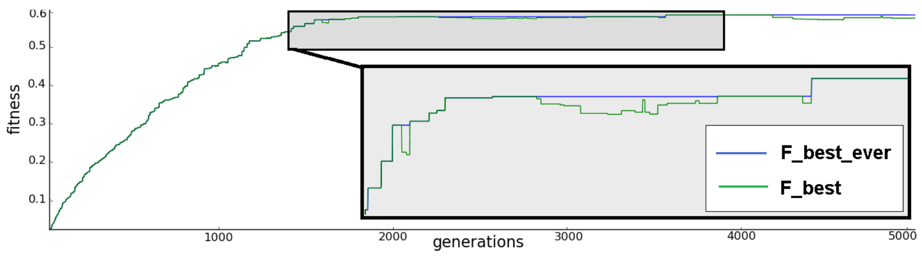

Fig. 5. Comparison between the fitness of the best individual at each cycle and that of the best up to that cycle

Each experiment was performed with population size $d = 8$, duplication parameter $dup = 2$, mutation rate $\rho = \{1, 2, 3\}$, probability of removal from the population $\tau = 0.01$ and maximum number of the fitness function evaluations $FFE_{Max} = 10^5$. The *fraz* column represents the fraction of the times in which the clonal exceeds or equals the deterministic of Louvain. Except for the *miserables* and *huckleberry* networks, for which probably more generations would be needed, in most cases, it succeeds in overcoming it. In fact the functions exploited have been chosen precisely to avoid the entrapment of dynamics in excellent premises, from which the deterministic it fails to escape instead. Furthermore, the number of communities found is not particularly influential in this disparity: there are cases in which the deterministic algorithm does not reach the global optimum despite having identified the correct number of clusters that maximizes modularity. Not even the number of nodes constituting N seems relevant in this regard. At the same time, from a simple graphical examination it is possible to note that the evolutionary (on the right) is a winner in cases where the number of links between one community and another is very large (below), therefore the network is rather complex, while for graphs with more evident clusters the two algorithms lead almost to the same result (above). In the latter case the landscape is in fact quite simple, so that the global optimum is easily reachable even from the deterministic algorithm, while in the former only the evolutionary algoritmh manages to explore the entire research space well.

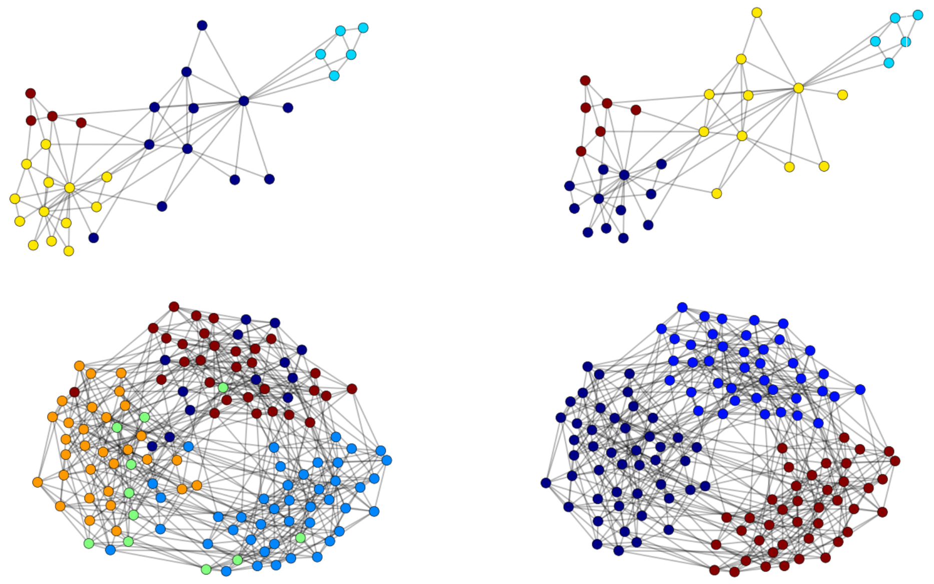

Fig. 6. Deterministic algorithm of Louvain (left) vs Opt-IA (right) on the graphs *karate* (top) and *3mixed* (bottom)

Table 1.

		Opt-IA				
Instance	$\lvert V\rvert$	Min	Max	Avg	Dev.St	Community
dolphins	62	0.5265	0.5285	0.5275	0.0008	5
karate	34	0.4198	0.4198	0.4198	0.0	4
ukfaculty	81	0.4488	0.4488	0.4488	0.0	4
miserables	77	0.5562	0.5600	0.5584	0.002	6
huckleberry	69	0.5308	0.5346	0.5334	0.0018	4
GN_benchmark2	128	0.4336	0.4336	0.4336	0.0	2
GN_benchmark4	128	0.5393	0.5393	0.5393	0.0	4
LFR_benchmark	128	0.1869	0.1980	0.1936	0.0037	5
almost_lattice	64	0.5436	0.5576	0.5576	0.0089	8
3mixed	128	0.4297	0.4297	0.4297	0.0	3

Table 2.

		LOUVAIN		Opt-IA	
Instance	$\lvert V\rvert$	Q	Community	Q	Community
dolphins	62	0.5188	5	**0.5285**	5
karate	34	0.4156	4	**0.4198**	4
ukfaculty	81	**0.4488**	4	**0.4488**	4
miserables	77	0.5583	6	**0.5600**	6
huckleberry	69	**0.5346**	4	**0.5346**	4
GN_benchmark2	128	**0.4336**	2	**0.4336**	2
GN_benchmark4	128	**0.5393**	4	**0.5393**	4
LFR_benchmark	128	0.1560	6	**0.1980**	5
almost_lattice	64	0.5279	8	**0.5576**	8
3mixed	128	0.3682	5	**0.4297**	3

5 Conclusions

We have introduced a clonal algorithm, denoted Opt-IA , for the optimization of modularity function. The discovery of community structures is essentially based on the maximization of the quality function Q, proposed by Newman [11]. The maximization of sought modularity has been achieved through the proposed heuristic algorithm which, with its performance, gives a significantly good guarantee on the solution quality. To drive the system to escape from local optima, we use, at each cycle, two special operators, *equiprobality* and *destroy*, that separate the community structures ensuring great improvements in fitness. When the *equiprobality* and *destroy* procedures eventually finish, we apply a *fuse* operator to have a cluster aggregation tool in order to refine the community structure. The proposed method outperforms, in most cases, the deterministic Louvain algorithm in terms of higher modularity values found and in less CPU time for computer-generated graphs most commonly used as a benchmark in community detection in networks. The number of communities found and the number of nodes constituting the communities are not particularly influential in this disparity. Furthermore, the evolutionary algorithm manages to well explore the entire search space when the network is rather complex and the number of links between one community and another is very large.

References

1. Bickel, P. J., and Chen, A.: A nonparametric view of network models and newman girvan and other modularities. Proceedings of the National Academy of Sciences 106, 50 (2009), 21068–21073. doi:10.1103/PhysRevE.74.036104
2. Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E.: Fast unfolding of communities in large networks. Journal of statistical mechanics: theory and experiment 2008, 10 (2008), P10008. doi:10.1088/1742-5468/2008/10/P10008
3. Boccaletti, S., Latora, V., Moreno, Y., Chavez, M., and Hwang, D.-U.: Complex networks: Structure and dynamics. Physics reports 424, 4-5 (2006), 175–308. doi: 10.1016/j.physrep.2005.10.009
4. Brandes, U., Delling, D., Gaertler, M., Gorke, R., Hoefer, M., Nikoloski, Z., and Wagner, D.: On modularity clustering. IEEE transactions on knowledge and data engineering 20, 2 (2007), 172–188. doi:10.1109/TKDE.2007.190689
5. Coscia, M., Giannotti, F., and Pedreschi, D.: A classification for community discovery methods in complex networks. Statistical Analysis and Data Mining: The ASA Data Science Journal 4, 5 (2011), 512–546. doi:10.1002/sam.10133
6. Fortunato, S.: Community detection in graphs. Physics reports 486, 3-5 (2010), 75–174. doi:10.1016/j.physrep.2009.11.002
7. Fortunato, S., and Barthelemy, M.: Resolution limit in community detection. Proceedings of the National Academy of Sciences 104, 1 (2007), 36–41. doi: 10.1073/pnas.0605965104
8. Girvan, M., and Newman, M. E.: Community structure in social and biological networks. Proceedings of the national academy of sciences 99, 12 (2002), 7821–7826. doi:10.1073/pnas.122653799
9. Mucha, P. J., Onnela, J., and Porter, M.: Communities in networks. Notices of the American Mathematical Society 56 (2009), 1082–1097.
10. Newman, M. E.: The structure and function of complex networks. SIAM review 45, 2 (2003), 167–256. doi:10.1137/S003614450342480
11. Newman, M. E.: Fast algorithm for detecting community structure in networks. Physical review E 69, 6 (2004), 066133. doi:10.1103/PhysRevE.69.066133
12. Newman, M. E.: Finding community structure in networks using the eigenvectors of matrices. Physical review E 74, 3 (2006), 036104. doi:10.1103/PhysRevE.74.036104
13. Newman, M. E.: Communities, modules and large-scale structure in networks. Nature physics 8, 1 (2012), 25. doi:10.1038/nphys2162
14. Newman, M. E., and Girvan, M.: Finding and evaluating community structure in networks. Physical review E 69, 2 (2004), 026113. doi:10.1103/PhysRevE.69.026113
15. Porter, M. A., Onnela, J.-P., and Mucha, P. J.: Communities in networks. Notices of the AMS 56, 9 (2009), 1082–1097.