Where the Local Search Affects Best in an Immune Algorithm

Rocco A. Scollo, Vincenzo Cutello, and Mario Pavone(✉)

Department of Mathematics and Computer Science, University of Catania, V.le A. Doria 6, 95125 Catania, Italy
rocco.scollo@phd.unict.it, {cutello,mario.pavone}@unict.it

Abstract. Hybrid algorithms are powerful search algorithms obtained by the combination of metaheuristics with other optimization techniques, although the most common hybridization is to apply a local solver method within evolutionary computation algorithms. In many published works in the literature, such local solver is run in different ways, sometimes acting on the perturbed elements and other on the best ones, and this raises the question of when it is best to run the local solver and on which elements it acts best in order to improve the reliability of the algorithm. Thus, three different ways of running local search in an immune algorithm have been investigated, and well-known community detection was considered as test-problem. The three methods analyzed have been assessed with respect their effect on the performances in term of quality solution found and information gained.

Keywords: Hybrid algorithms · Hybrid metaheuristics · Hybrid immune algorithms · Hybrid-IA · Community detection · Modularity optimization · Network science

1 Introduction

Evolutionary computation represents today a consolidated and established class of algorithmic methodologies able to tackle hard and complex optimization problems mainly thanks to their ability to be easily applied on new and unknown problems, and, in general, on all those problems whose knowledge about their features and structures are very limited. Among the evolutionary computation methodologies, the immune-inspired algorithms represent a powerful algorithmic class, which takes inspiration by the principles and dynamics of the biological immune system (IS). What makes the IS very interesting and source of inspiration from a computational perspective is its ability in learning, detecting, and recognizing foreign and dangerous entities [10].

However, although many methodologies inspired by biology and nature have been developed, applied effectively in many combinatorial optimization problems, it clearly emerges from the literature that just on these kinds of problems their hybridization, that is their combination with concepts and/or components

M. Baldoni and S. Bandini (Eds.): AIxIA 2020, LNAI 12414, pp. 99–114, 2021.
https://doi.org/10.1007/978-3-030-77091-4_7

of other optimization techniques (e.g., Local Search algorithms), turns out to be much more efficient and successful, thus proving to be very powerful search algorithms [1]. The basic idea of this combination is to exploit the strengths of one to overcome the weaknesses of the other: random-search performs an excellent exploration of the search space (thanks to its stochastic nature), whilst deterministic approach, for instance, is useful for refine and improve the current solutions found. There are many different ways to generate hybrid methods, but the most common and popular is combine evolutionary algorithms and local improver methods (such as Local Search, Hill-Climbing, etc.), which are applied one after another, using the output of the former as input for the latter. Furthermore, it is also common in this case that the revised and improved individual, by the local improver method, replaces the original one in the population.

In this paper we want to investigate on when is better to perform the Local Search (LS), and if, in the overall, replace the original solution with the revised one by LS is the best choice. In light of this, a *Hybrid Immune Algorithm*, called Hybrid-IA, has been taken into account in an attempt to answer these questions. Hybrid-IA has been considered as it was successful applied in several and various combinatorial optimization problems. [2–5]. Thus, the effect of the local search on the performances of Hybrid-IA has been investigated considering three different positions in the evolutionary cycle where to run the local improver method: (1) acting and refining the best solutions found so far (to be run just after selecting the best elements for the next generation); (2) acting on the perturbed elements and replacing them (to be run after the hypermutation operator, see Algorithm 1); and, finally, (3) acting always on the perturbed elements but producing a new population, whose individuals will compete to the selection of the new population for the next generation.

In order to analyse which among of the three methods best affects the performance of Hybrid-IA, the well-known *Community Detection* problem has been considered, which is considered one of the most important problems in Network Sciences and Graph Analysis. The study of community structures inspires intense research activities to visualize and understand the dynamics of a network at different scales [8,11,12]. The goal of this task is uncovering the inherent community structure of a network, which means to discover those groups of nodes sharing common properties. The *modularity* is certainly the evaluation metric most used to assess the quality of the uncovered communities in a network [19], based on the idea that a random graph is not expected to have a community structure, therefore the possible existence of communities can be revealed by the difference of density between vertices of the graph and vertices of a random graph with the same size and degree distribution. Given an undirected graph $G = (V, E)$, with V the set of vertices ($|V| = N$), and E the set of edges ($|E| = M$), the modularity of a community is defined as:

$$Q = \frac{1}{2M}\left[\sum_{i=1}^{N}\sum_{j=1}^{N}\Big(A_{ij} - \frac{d_i d_j}{2M}\Big)\delta(i,j)\right], \tag{1}$$

where A_{ij} is the adjacency matrix of G, d_i and d_j are the degrees of nodes i and j respectively, and $\delta(i,j) = 1$ if i, j belong to the same community, 0 otherwise.

Several artificial networks have been produced using the well-known *LFR* algorithm [16,17] in order to assess what proposed method affect best in term of solution quality found, and information gained. The use of artificial networks allows to inspect these three methods under different complexity scenarios. It is important to highlight that this investigation begins from consolidated and asserted outcomes in [2] (precisely method A), which prove that Hybrid-IA is competitive and comparable with the state-of-the-art.

Analysing all experimental results emerges that performing the local improver just after the hypermutation operator, that means to act directly on the elements produced by the random search, is the one to produce best efficiency and reliability on Hybrid-IA since it allows to perform a better and wide exploration of the search space and this is useful to jump out from local optima.

2 The Hybrid Immune Algorithm

Immune Algorithms (IA) are among the most used population-based metaheuristics, successfully applied in search and optimization tasks. They take inspiration from the dynamics of the immune system in performing its job of protecting living organisms. One of the features of the immune system that makes it a very good source of inspiration is its ability to detect, distinguish, learn, and remember all foreign entities discovered [10]. The presented Hybrid Immune Algorithm, called Hybrid-IA, belongs to the special class *Clonal Selection Algorithms* (CSA) [5,20], whose efficiency is due to the three main immune operators: (i) cloning, (ii) hypermutation, and (iii) aging. The overall scheme of Hybrid-IA, proposed in [2,5], is shown in Algorithm 1. It is based on two main concepts: antigen (Ag), which represents the problem to tackle, and B cell, or antibody (Ab) that represents a candidate solution ($\boldsymbol{x}$), i.e., a point in the solution space. At each time step t, the algorithm maintains a population of d candidate solutions to the problem tackled. The population is randomly initialized at the time step $t = 0$. Then, just after the initialization step, the fitness function, specific to the problem, is evaluated for each randomly generated element ($\boldsymbol{x} \in P^{(t)}$) by using the function ComputeFitness($P^{(t)}$). The algorithm ends its evolutionary cycle when the halting criterion is reached. For this work, it was fixed to a maximum number of generations ($MaxGen$).

The first immune operator to take place is the *Cloning Operator* ($6th$ line in Algorithm 1). This operator simply copies *dup* times each B cell producing an intermediate population $P^{(clo)}$ of size $d \times dup$. We used a static cloning in order to avoid premature convergences. Indeed, if a number of clones proportional to the fitness value is produced instead, we could have a population of B cells very similar to each other, and we would, consequently, be unable to perform a proper exploration of the search space getting easily trapped in local optima. Once created the copies of any B cell, to each of those is assigned an age, which determines how long it can live in the population, from the assigned age until it reaches the maximum age allowed (τ_B). Specifically, a random age chosen in the range $[0 : \frac{2}{3}\tau_B]$ is assigned to each clone [20]; in this way, each clone is guaranteed to stay in the population for at least a fixed number of generations ($\frac{1}{3}\tau_B$ in the worst case). The age assignment and the aging operator play a crucial role on Hybrid-IA performances, and any evolutionary algorithm in general, because they are able to keep a right amount of diversity among the solutions, thus avoiding premature convergences [7,21].

Algorithm 1. Pseudo-code of Hybrid-IA.

```
1:  procedure Hybrid-IA($d$, $dup$, $\rho$, $\tau_B$)
2:      $t \leftarrow 0$
3:      $P^{(t)} \leftarrow$ InitializePopulation($d$)
4:      ComputeFitness($P^{(t)}$)
5:      while $\neg$StopCriterion do
6:          $P^{(clo)} \leftarrow$ Cloning($P^{(t)}, dup$)
7:          $P^{(hyp)} \leftarrow$ Hypermutation($P^{(clo)}, \rho$)
8:          ComputeFitness($P^{(hyp)}$)
9:          $(P_a^{(t)}, P_a^{(hyp)}) \leftarrow$ Aging($P^{(t)}, P^{(hyp)}, \tau_B$)
10:         $P^{(select)} \leftarrow (\mu + \lambda)-$Selection($P_a^{(t)}, P_a^{(hyp)}$)
11:         $P^{(t+1)} \leftarrow$ LocalSearch($P^{(select)}$)
12:         ComputeFitness($P^{(t+1)}$)
13:         $t \leftarrow t + 1$;
14:     end while
15: end procedure
```

The *Hypermutation Operator* has the main goal of exploring the neighbourhoods of solutions by evaluating how good each clone is (7*th* line in Algorithm 1). The mutation rate is determined through an *inversely proportional* law to the fitness function value of the B cell considered, that is, the better the fitness value of the element is, the smaller the mutation rate will be. In particular, let $\boldsymbol{x}$ be a cloned B cell, the number of mutations is determined by $M = \lfloor(\alpha \times \ell) + 1\rfloor$, with ℓ the length of the B cell (i.e. $\ell = |V|$), and α representing the *mutation rate* obtained as $\alpha = e^{-\rho \hat{f}(\boldsymbol{x})}$, where ρ determines the shape of the mutation rate, and $\hat{f}(\boldsymbol{x})$ is the fitness function normalized in the range $[0, 1]$. Naturally, the mutation operator that acts on a single element of the cloned B cell is problem-dependent.

The static *Aging Operator* in Hybrid-IA acts on each mutated B cells by removing older ones from the two populations $P^{(t)}$ and $P^{(hyp)}$ (9*th* line in Algorithm 1). Basically, let τ_B be the maximum number of generations allowed for every B cell to stay in its population; then, once the age of a B cell exceeds τ_B (i.e., age $= \tau_B + 1$), it will be removed independently from its fitness value. However, an exception is done in Hybrid-IA for the best current solution, which is kept into the population even if its age is older than τ_B. Such a variant of the aging operator is called *elitist aging operator*. In the overall, the main goal of this operator is to allow the algorithm to escape and jump out from local optima, assuring a proper turnover between the B cells in the population, and producing, consequently, high diversity among them.

After the aging operator, the best d survivors from both populations $P_a^{(t)}$ and $P_a^{(hyp)}$ are selected, in order to generate the temporary population $P^{(select)}$, on the local search will be performed ($10th$ line in Algorithm 1). Such a selection is performed by the $(\mu + \lambda)$-*Selection Operator*, where $\mu = d$ and $\lambda = (d \times dup)$. The operator identifies the d best elements among the set of offspring and the old parent B cells, ensuring consequently monotonicity in the evolution dynamics.

The main idea behind the *Local Search* operator is to refine and improve in deterministic way the solutions produced by the stochastic mutation operator. In this study the affect and impact of the position where to run the local search within Hybrid-IA is inspected (see Algorithm 1). Specifically, three approaches have been taken into account:

- **Method *A*:** applying the local search operator just after the selection operator, acting, consequently, on the individuals already selected to produce the new population for the next generation. In this way, the local search is always applied to the best solutions, intensifying the exploration in their relative neighbourhood.
- **Method *B*:** applying LS to the population generated by the hypermutation operator, where each revised individual replaces the hypermutated one, maintaining the same population. In this way, it is applied to a wider set of solutions generated from the current ones through mutation allowing a better exploration of the search space. Of course, the computational complexity is higher than in the previous case because it is applied to a population of $d \times dup$.
- **Method *C*:** applying LS to the hypermutated individuals, as in the previous method, but producing a new temporary population, which will compete with the other populations to the selection for the next generation. In this way, the algorithm keeps memory of the discoveries made via random search, which generates diversity in the population, and, at the same time, it carries out a careful exploration of their neighbourhood via local search. Computational complexity is the same as the previous method.

2.1 Hybrid-IA for the Community Detection

Once described the structure and features of Hybrid-IA in general, in this section all details on the operators and local search developed specifically for the community detection problem are reported. Any B cell in the population is represented as subdivision of the vertices of the graph $G = (V, E)$ in communities. A solution $\boldsymbol{x}$ is a sequence of $N = |V|$ elements in the range $[1, N]$ such that $x_i = j$ indicates that the node i belongs to the cluster j. In the initialization phase ($t = 0$), each element of the population is randomly generated assigning each vertex i to a group j, with $j \in [1, N]$. The aim of the designed hypermutation operator is to explore the search space in order to create new communities by moving a nodes variable percentage from existing communities. For each B cell, it chooses randomly two communities c_i and c_j $(c_i \neq c_j)$ among all existing ones, and, with a probability given by α, all vertices in c_i are moved to c_j. The α *mutation rate* is, then, defined as the probability to move a node from one community to another one.

The designed Local Search, introduced in [2], allows to speed up the convergence of the algorithm and intensify the search in the neighbourhood of each solution. The idea is to deterministically improve a solution using the *Move Vertex* operator [14]. This operator moves a node from its community to another one within its neighbours, taking into account the gain of modularity, that can be defined as the variation in modularity produced when a node is moved from a community to another. The modularity Q, defined in Eq. (1), can be rewritten as:

$$Q(c) = \sum_{i=1}^{k} \left[\frac{\ell_i}{M} - \left(\frac{d_i}{2M} \right)^2 \right], \tag{2}$$

where k is the number of communities identified; $c = \{c_1, \ldots, c_i, \ldots c_k\}$ is a partition of V; l_i and d_i are, respectively, the number of links inside the community i, and the sum of the degrees of vertices belonging to the i community. The gain of modularity of a vertex $u \in c_i$ is the modularity variation produced by moving u from c_i to c_j, that is:

$$\Delta Q_u(c_i, c_j) = \frac{l_{c_j}(u) - l_{c_i}(u)}{M} + d_V(u) \left[\frac{d_{c_i} - d_V(u) - d_{c_j}}{2M^2} \right], \tag{3}$$

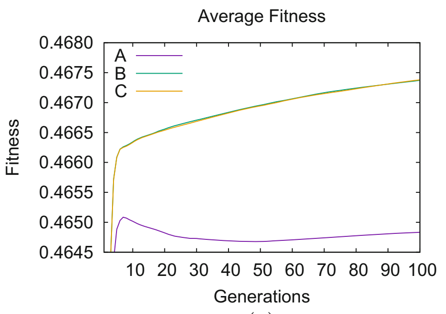
where $l_{c_i}(u)$ and $l_{c_j}(u)$ are the number of links from u to nodes in c_i and c_j respectively, and $d_V(u)$ is the degree of u when considering all the vertices V. If the gain is greater than 0, then moving node u from c_i to c_j produces an improvement in modularity. Consequently, the goal of the move vertex operator is to find a node u to move to an adjacent community in order to maximize the gain of modularity. The local search, for each solution, works sorting the communities in increasing order with respect the ratio of internal links and degree of the community; in this way, it tries to repair the solutions starting from poorly formed communities, which are produced by the hypermutation operator (random search).

3 Experimental Results

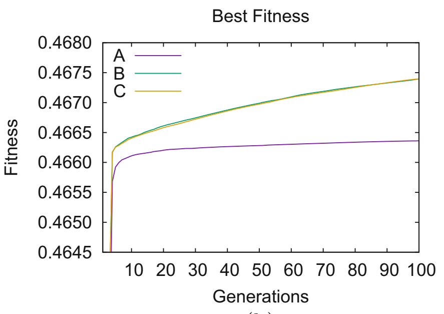
In this section all experiments performed are presented in order to inspect what is the best position where to run the local search within the evolutionary cycle of Hybrid-IA. For this study, the community detection has been considered as the test problem, and, specifically, several artificial networks have been taken into account as benchmark instances. These networks were generated by *LFR* algorithm, proposed in [16,17], and have been used because they allow us to perform our study on different complexity scenarios thanks their diverse features. Note that the validity of this benchmark is given by faithfully reproducing the keys features of real graphs communities. In particular, networks with number of nodes 300, 500, 1000, and 5000 have been generated, with average degree 15, 20, and 25, and maximum degree 50. Further, for all $|V|$ values, the exponent of the degrees distribution was set to $\tau_1 = 2$, whilst the distribution of community sizes to $\tau_2 = 1$. Minimum and maximum of the communities' size for such artificial networks were considered, respectively, $min_c = 10$ and $max_c = 50$. The mixing parameter μ_t, which identifies the relationship between the node's external and internal degree with respect to its community, was instead set to 0.5: greater is the value of μ_t, greater is number of edges that a node shares with nodes outside of its communities. For each network parameters configuration 5 random instances have been generated. For all experiments performed on all tested networks the following parameters setting have been used for Hybrid-IA: B cells population size $d = 100$; number of generated clones $dup = 2$; ρ and τ_B, respectively, to 1.0 and 5. All these parameters have been identified both from the knowledge learned by previous works [2,5,20], and from preliminary experiments carried out. Maximum number of generations has been considered as stopping criterion and was set to $MaxGen = 100$. 50 independent runs were also performed. In order to assess which of the three method is the most efficient and reliable, in addition to the convergence behaviour analysis and solution (modularity) quality obtained by each method, also the Information Gain as been considered as evaluation metric. This entropic function measures the quantity of information the system discovers during the learning phase (see [3,15]).

For all network instances the convergence analysis was carried out for the three methods. Due to the space limit only the most significant ones are reported. In Figs. 1 and 2 are shown the convergence plots for the LFR instances with 1000 nodes and average degree k of 15 and 20 respectively. From these plots can be noted that method A reaches high modularity values in the first 10 generations, afterwards improves very slowly. The same trend is also visible in the average fitness of the population, with a peak in the first generations and a slow growth for the rest of the run. Methods B and C, on the other hand, have a growth much more constant and linear, both in terms of the best solution and average of the population. The average fitness curve is very close to the best solution one, indicating then a population composed of solutions with values of modularity very similar to each other and consequently very homogeneous. This is also supported by the information gain curve, in which the peak is reached in the earliest generations, after that it stays in a steady-state for the rest of the execution (Figs. 1c and 2c), while method A needs more generations to converge to the same value reached by the other two methods.

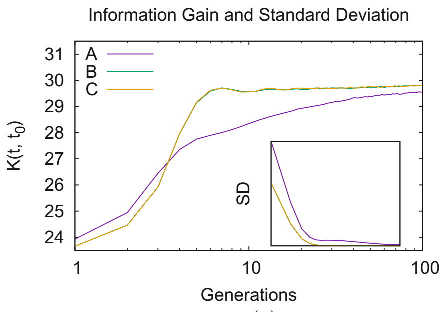
The same situation is obtained in the networks with 5000 nodes and average degree k equal to 20 and 25 (Figs. 3 and 4). Also in these plots, method A has a much slower convergence than the other two methods and maintains a certain degree of diversity within the population, demonstrated by the distance between the two curves: best solution, and average fitness of the population. In this case, in both methods B and C, the two curves have a higher slope, which suggests that with more generations they could achieve better solutions.



(a)



(b)



(c)

Fig. 1. Convergence behavior of the three methods on `LFR(1000,15,0.5)` graph. (a) Average and (b) best fitness value of the population versus generations. (c) Information gain and standard deviation.

The greater diversity introduced by methods B and C, allows to better explore the search space and to find solutions with a higher modularity value. The Table 1 shows the results of the experiments of the three methods carried out on benchmark instances. In particular, in the table are reported the maximum value of modularity (Q), average number of communities (K) and computational time, all averaged over 5 random instances. From these results, can be noted that on the networks with 300 nodes, all three methods reach what is most likely the maximum modularity value, detecting the same number of communities. On the other hands, for the instances with 500 nodes, only for $k = 20$ method A reaches the same modularity value of methods B and C, while for $k = 15$ reaches a slightly lower modularity value, about 1.79×10^{-4}, which leads to a different number of communities detected (17.6 for method A versus 16.8 for both method B and C). The difference in modularity becomes greater as the number of nodes increases. For the instances with 1000 nodes, method A reaches a lower modularity value than the other two methods (about 10^{-3} on average for both instances), as observed in the respectively convergence plots. The other two methods, B and C, reach the best modularity value for $k = 20$ and $k = 15$ respectively, with a minimum difference between each other.

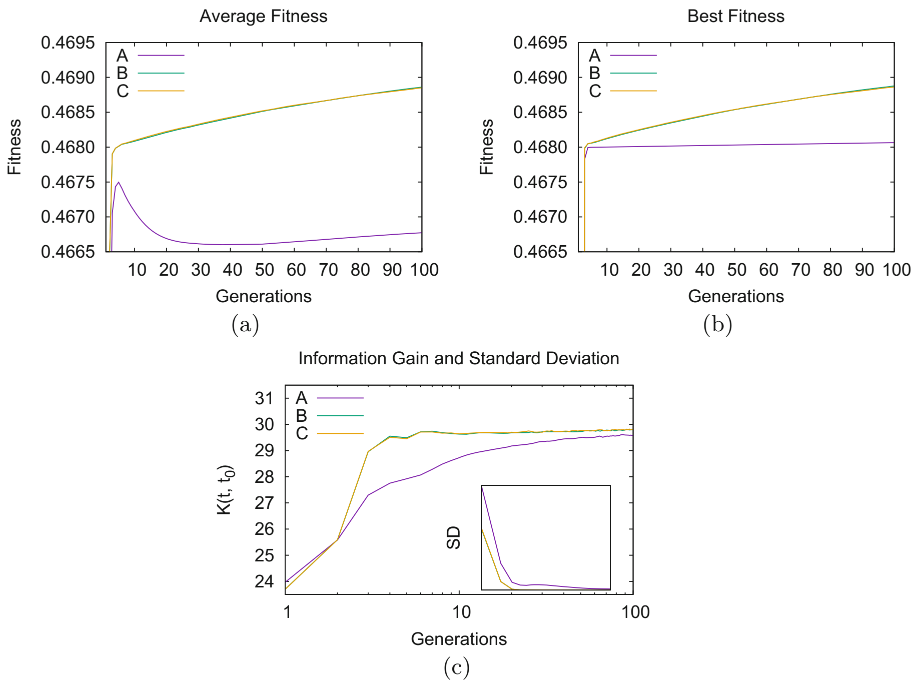


Fig. 2. Convergence behavior of the three methods on `LFR(1000,20,0.5)` graph. (a) Average and (b) best fitness value of the population versus generations. (c) Information gain and standard deviation.

The same results can be observed for the network with 5000 nodes, where method A is behind the other two methods in terms of modularity, although with a lower gap with respect the instances with 1000 nodes (about 4×10^{-4}), while methods B and C achieve the best modularity value for $k = 25$ and $k = 20$ respectively. Moreover, unlike smaller instances (300, and 500 nodes), on the networks with 1000 and 5000 nodes the number of communities found by methods B and C is different. Finally, from the computational time point of view, methods B and C, as expected, take about 90% more time than method A, but, nevertheless, they allow a better exploration of the search space, and then obtaining solutions with higher modularity values.

In order to consolidate the outcomes obtained so far and make them more reliable an extended further analysis has been performed at the varying of the mixing parameter (μ_t), on all three methods, whose outcomes are reported in Tables 2 and 3. As described above, the mixing parameter μ_t identifies the relationships between the communities, that is the ratio between the node's degree internal to the community, and the external one. In this way, it is possible to carry out a comparison analysis in different scenarios, each of which was designed as realistic as possible ($\mu_t = \{0.1, 0.2, \ldots, 0.8\}$). Table 2 reports the experimental results obtained by the three methods on networks with 300 and 500 vertices.

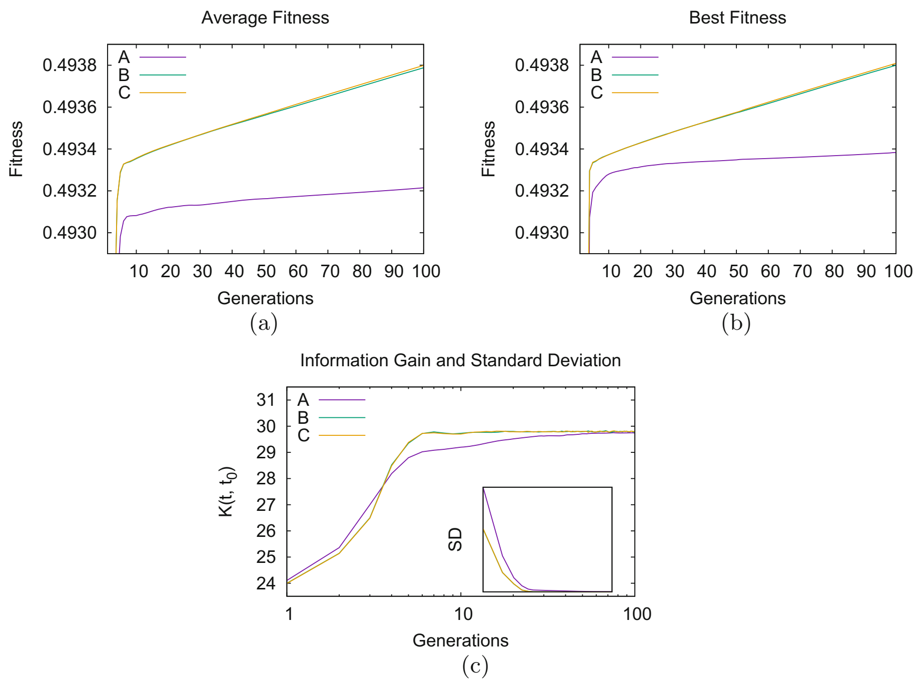


Fig. 3. Convergence behavior of the three methods on `LFR(5000,20,0.5)` graph. (a) Average and (b) best fitness value of the population versus generations. (c) Information gain and standard deviation.

Table 1. Results of the three methods on *LFR Benchmarks*. The results are averaged on 5 random instances and calculated over 100 independent runs.

Instance	*A*			*B*			*C*		
	K	Q	Time	K	Q	Time	K	Q	Time
$(300, 15, 0.5)$	11.6	**0.392061**	8.3	11.6	**0.392061**	14.5	11.6	**0.392061**	15.5
$(300, 20, 0.5)$	11.2	**0.386560**	9.4	11.2	**0.386560**	16.9	11.2	**0.386560**	17.9
$(500, 15, 0.5)$	17.6	0.436989	13.7	16.8	**0.437168**	24.6	16.8	**0.437168**	26.1
$(500, 20, 0.5)$	17.0	**0.430526**	16.3	17.0	**0.430526**	29.7	17.0	**0.430526**	31.3
$(1000, 15, 0.5)$	34.4	0.467073	28.0	30.0	0.468122	51.2	30.4	**0.468205**	53.9
$(1000, 20, 0.5)$	37.0	0.468532	33.8	33.0	**0.469451**	62.6	32.2	0.469442	65.2
$(5000, 20, 0.5)$	196.4	0.493532	182.4	190.2	0.493985	346.0	189.4	**0.493994**	353.5
$(5000, 25, 0.5)$	193.0	0.493379	228.8	185.4	**0.493741**	438.1	184.6	0.493740	427.1

Focusing on the first one, that is the network with 300 nodes, it is possible to note how the three methods are equivalent on all those instances where the external links are below, or around, the threshold of 50% ($\mu_t \leq 0.5$). By increasing this threshold, instead, methods B and C significantly improve method A, both in

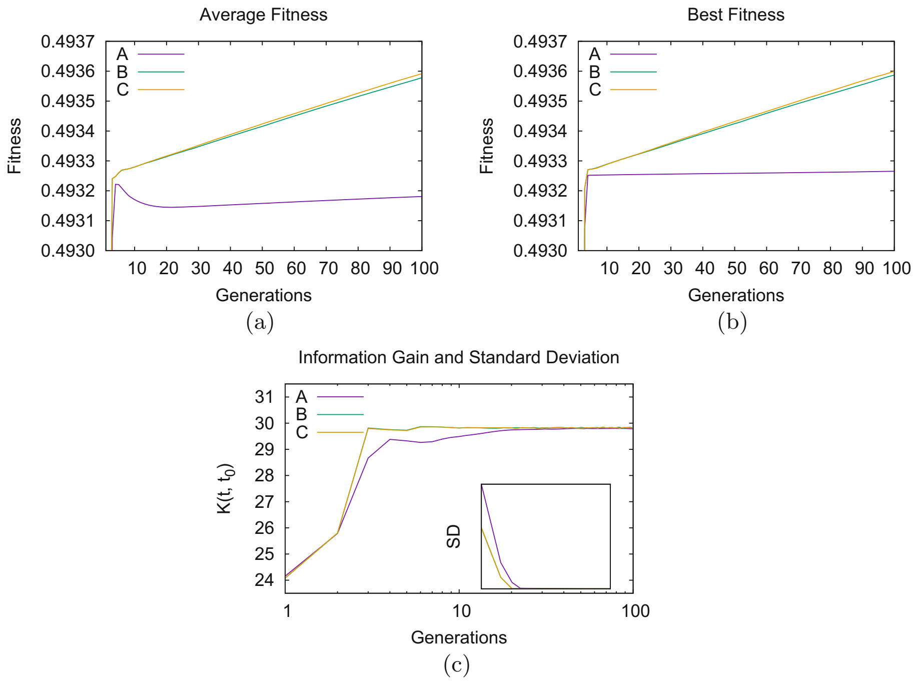


Fig. 4. Convergence behavior of the three methods on `LFR(5000,25,0.5)` graph. (a) Average and (b) best fitness value of the population versus generations. (c) Information gain and standard deviation.

terms of the modularity values and number of communities discovered. A similar behavior can be observed also on the network with 500 nodes, although the threshold, where the three methods are equivalent, decreases to $\mu_t \leq 0.4$ when the average degree k of the nodes is 15. What is more interesting to note is that the method C considerably outperforms not only the method A, which is to be expected based on the previous results, but also the method B, especially when the average degree is $k = 20$ and the external links grow ($\mu_t \geq 0.6$). This is due to the fact that as the nodes average degree and, primarily, the number of external links increase, the problem becomes harder and, consequently, to have two different populations competing with each other (the ones produced by the random search and by the refinement one) produce more heterogeneity and therefore higher diversity in the population, which helps the algorithm to carry out a better exploration in the search space, avoiding thus being trapped in local optima.

From Table 3, where are showed the experimental results on the networks with 1000 and 5000 nodes, appears even more obvious how the method A achieves worst performances than the other two, in all instances considered. On the other hand, analyzing the results obtained with the two methods B and C, it is possible to note that the improvements of one over the other are minimal, except in some

Table 2. Results of the three methods at the mixing parameter varying (μ_t), on networks with 300 and 500 nodes.

Instance	*A*			*B*			*C*		
	K	Q	Time	K	Q	Time	K	Q	Time
$\|V\| = 300$									
$(300, 15, 0.1)$	10.2	**0.766602**	3.8	10.2	**0.766602**	6.5	10.2	**0.766602**	7.4
$(300, 15, 0.2)$	12.0	**0.673573**	4.5	12.0	**0.673573**	7.9	12.0	**0.673573**	8.7
$(300, 15, 0.3)$	13.0	**0.583192**	5.3	13.0	**0.583192**	9.3	13.0	**0.583192**	10.1
$(300, 15, 0.4)$	11.2	**0.484404**	5.9	11.2	**0.484404**	10.5	11.2	**0.484404**	11.3
$(300, 15, 0.5)$	11.6	**0.392061**	8.3	11.6	**0.392061**	14.5	11.6	**0.392061**	15.5
$(300, 15, 0.6)$	11.0	0.305655	6.9	10.6	**0.306509**	12.4	10.6	**0.306509**	13.2
$(300, 15, 0.7)$	8.0	0.241728	7.1	6.6	0.247123	12.8	6.8	**0.247811**	13.5
$(300, 15, 0.8)$	7.8	0.232779	7.0	6.8	**0.240292**	12.7	6.8	0.239547	13.4
$(300, 20, 0.1)$	8.6	**0.754457**	4.5	8.6	**0.754457**	7.8	8.6	**0.754457**	8.7
$(300, 20, 0.2)$	11.8	**0.670396**	5.2	11.8	**0.670396**	9.3	11.8	**0.670396**	10.1
$(300, 20, 0.3)$	12.0	**0.577310**	6.1	12.0	**0.577310**	11.0	12.0	**0.577310**	11.8
$(300, 20, 0.4)$	12.6	**0.496497**	7.0	12.6	**0.496497**	12.7	12.6	**0.496497**	13.5
$(300, 20, 0.5)$	11.2	**0.386560**	9.4	11.2	**0.386560**	16.9	11.2	**0.386560**	17.9
$(300, 20, 0.6)$	11.0	0.288503	8.4	10.8	**0.288713**	15.4	10.8	**0.288713**	16.2
$(300, 20, 0.7)$	8.6	0.223786	8.5	7.8	0.227081	15.5	7.6	**0.227327**	16.2
$(300, 20, 0.8)$	7.6	0.202636	8.7	6.2	**0.207970**	15.8	6.4	0.207610	16.5
$\|V\| = 500$									
$(500, 15, 0.1)$	18.6	**0.820874**	6.1	18.6	**0.820874**	10.4	18.6	**0.820874**	11.6
$(500, 15, 0.2)$	20.4	**0.724672**	7.4	20.4	**0.724672**	13.0	20.4	**0.724672**	14.2
$(500, 15, 0.3)$	18.6	0.626449	8.7	18.4	**0.626457**	15.5	18.4	**0.626457**	16.7
$(500, 15, 0.4)$	17.0	**0.529800**	9.9	17.0	**0.529800**	18.0	17.0	**0.529800**	19.2
$(500, 15, 0.5)$	17.6	0.436989	13.7	16.8	**0.437168**	24.6	16.8	**0.437168**	26.1
$(500, 15, 0.6)$	16.6	0.336580	12.0	14.6	**0.337333**	22.0	14.8	0.337325	23.2
$(500, 15, 0.7)$	11.0	0.248666	12.1	8.8	0.256057	22.2	8.2	**0.256795**	23.3
$(500, 15, 0.8)$	10.2	0.237794	11.9	8.0	0.245876	21.8	7.6	**0.246542**	22.8
$(500, 20, 0.1)$	19.2	**0.817709**	7.1	19.2	**0.817709**	12.6	19.2	**0.817709**	13.8
$(500, 20, 0.2)$	17.6	**0.721416**	8.8	17.6	**0.721416**	15.9	17.6	**0.721416**	17.1
$(500, 20, 0.3)$	19.2	**0.629277**	10.4	19.2	**0.629277**	19.0	19.2	**0.629277**	20.2
$(500, 20, 0.4)$	18.6	0.533117	11.9	18.4	**0.533150**	22.1	18.4	**0.533150**	23.3
$(500, 20, 0.5)$	17.0	**0.430526**	16.3	17.0	**0.430526**	29.7	17.0	**0.430526**	31.3
$(500, 20, 0.6)$	17.4	0.338484	14.2	17.0	**0.338712**	26.5	17.0	**0.338712**	27.7
$(500, 20, 0.7)$	13.2	0.241751	15.0	11.2	0.245852	27.6	10.4	**0.246251**	28.6
$(500, 20, 0.8)$	9.0	0.211891	14.3	7.4	0.218448	26.3	7.0	**0.218741**	27.4

Table 3. Results of the three methods at the mixing parameter varying (μ_t), on networks with 1000 and 5000 nodes.

Instance	A			B			C		
	K	Q	Time	K	Q	Time	K	Q	Time
$\lvert V\rvert = 1000$									
$(1000, 15, 0.1)$	38.8	0.860777	12.3	38.0	**0.860833**	21.2	38.2	0.860826	23.4
$(1000, 15, 0.2)$	37.0	0.762139	15.1	35.8	0.762252	27.0	35.8	**0.762255**	29.2
$(1000, 15, 0.3)$	36.6	0.664539	17.6	35.2	0.664881	32.0	34.0	**0.664966**	34.2
$(1000, 15, 0.4)$	34.8	0.565415	20.6	33.0	**0.565756**	37.8	33.2	0.565726	40.0
$(1000, 15, 0.5)$	34.4	0.467073	28.0	30.0	0.468122	51.2	30.4	**0.468205**	53.9
$(1000, 15, 0.6)$	33.2	0.368714	25.1	28.2	0.370871	46.7	27.6	**0.370888**	48.9
$(1000, 15, 0.7)$	24.4	0.270108	25.3	16.8	**0.279350**	47.0	16.4	0.279087	49.1
$(1000, 15, 0.8)$	17.0	0.241944	23.9	9.0	0.249147	44.2	9.0	**0.250582**	46.2
$(1000, 20, 0.1)$	39.2	0.860630	14.2	37.8	**0.860694**	25.0	38.0	0.860692	27.3
$(1000, 20, 0.2)$	38.0	0.762173	17.9	36.2	**0.762229**	32.5	36.6	0.762228	34.8
$(1000, 20, 0.3)$	38.2	0.665545	21.2	36.0	**0.665768**	39.1	36.4	0.665732	41.5
$(1000, 20, 0.4)$	40.4	0.566834	24.3	35.6	**0.567383**	45.3	35.8	0.567336	47.6
$(1000, 20, 0.5)$	37.0	0.468532	33.8	33.0	**0.469451**	62.6	32.2	0.469442	65.2
$(1000, 20, 0.6)$	36.4	0.368643	30.1	29.4	**0.370331**	56.7	29.4	0.370210	59.0
$(1000, 20, 0.7)$	32.0	0.271142	31.6	26.2	**0.275290**	59.6	26.4	0.275260	62.0
$(1000, 20, 0.8)$	11.8	0.215022	29.4	8.2	**0.222399**	54.7	8.8	0.221720	56.8
$\lvert V\rvert = 5000$									
$(5000, 20, 0.1)$	199.8	0.892274	71.4	193.8	0.892360	129.3	193.0	**0.892371**	139.4
$(5000, 20, 0.2)$	200.4	0.792707	90.6	194.4	**0.792857**	167.7	194.4	0.792848	177.7
$(5000, 20, 0.3)$	191.0	0.692909	108.4	186.4	0.693102	203.3	185.2	**0.693111**	213.5
$(5000, 20, 0.4)$	190.4	0.593117	127.1	186.6	0.593357	240.5	184.0	**0.593378**	251.0
$(5000, 20, 0.5)$	196.4	0.493532	182.4	190.2	0.493985	346.0	189.4	**0.493994**	353.5
$(5000, 20, 0.6)$	196.6	0.393921	160.3	187.8	0.394428	306.1	186.0	**0.394455**	316.5
$(5000, 20, 0.7)$	203.8	0.292948	176.2	192.8	0.293851	337.3	191.0	**0.294002**	347.9
$(5000, 20, 0.8)$	87.2	0.209471	158.8	74.8	**0.212451**	299.5	73.6	0.212407	307.7
$(5000, 25, 0.1)$	173.8	0.892187	85.3	170.2	**0.892213**	156.9	170.2	0.892212	167.8
$(5000, 25, 0.2)$	184.6	0.792506	109.8	177.2	0.792600	205.6	176.6	**0.792608**	216.5
$(5000, 25, 0.3)$	190.2	0.692853	133.5	180.6	0.693021	252.5	179.8	**0.693028**	263.8
$(5000, 25, 0.4)$	203.6	0.593075	157.5	192.6	**0.593369**	300.6	191.0	0.593358	311.9
$(5000, 25, 0.5)$	193.0	0.493379	228.8	185.4	**0.493741**	438.1	184.6	0.493740	427.1
$(5000, 25, 0.6)$	195.6	0.393853	199.4	186.8	0.394274	383.5	184.2	**0.394325**	395.3
$(5000, 25, 0.7)$	198.4	0.293983	219.1	190.0	0.294451	421.9	186.6	**0.294483**	434.2
$(5000, 25, 0.8)$	136.8	0.187554	213.6	128.2	0.189877	409.5	121.0	**0.189884**	419.8

few instances, in which the difference in the results is more consistent, but in any case, there is no one method that outperforms the other.

Finally, at the conclusion of the analysis conducted, also on these experiments emerges that the methods B and C seem to be more suitable than method A

Table 4. Functional sensitivity analysis in community detection.

Instance	*NMI*			*ARI*			NVI		
	A	*B*	*C*	*A*	*B*	*C*	*A*	*B*	*C*
$\|V\| = 1000$									
(1000, 15, 0.1)	**0.995076**	0.993003	0.993406	**0.987372**	0.981960	0.982767	**0.009774**	0.013864	0.013076
(1000, 15, 0.2)	**0.989911**	0.986152	0.986299	**0.972978**	0.961779	0.962704	**0.019958**	0.027290	0.026998
(1000, 15, 0.3)	**0.986682**	0.982557	0.978907	**0.959832**	0.947997	0.938800	**0.026279**	0.034215	0.041232
(1000, 15, 0.4)	**0.984039**	0.977067	0.978024	**0.951622**	0.928518	0.931673	**0.031354**	0.044819	0.042942
(1000, 15, 0.5)	**0.980926**	0.964332	0.966406	**0.929535**	0.880768	0.887543	**0.037157**	0.068856	0.064955
(1000, 15, 0.6)	**0.962855**	0.941986	0.938977	**0.878173**	0.810258	0.807698	**0.071483**	0.109514	0.114978
(1000, 15, 0.7)	0.566374	**0.570550**	0.561501	0.255813	**0.267680**	0.267179	0.602756	**0.598517**	0.607492
(1000, 15, 0.8)	**0.153777**	0.141303	0.133809	0.018578	**0.020726**	0.018665	**0.916663**	0.923923	0.928225
(1000, 20, 0.1)	**0.997058**	0.994186	0.994506	**0.992727**	0.986538	0.986870	**0.005859**	0.011530	0.010894
(1000, 20, 0.2)	**0.996428**	0.991117	0.992519	**0.991809**	0.977045	0.981142	**0.007108**	0.017577	0.014826
(1000, 20, 0.3)	**0.992238**	0.986079	0.987491	**0.975973**	0.960172	0.964767	**0.015342**	0.027357	0.024658
(1000, 20, 0.4)	**0.991002**	0.975938	0.976641	**0.969473**	0.924002	0.927008	**0.017786**	0.046974	0.045641
(1000, 20, 0.5)	**0.981394**	0.966023	0.963240	**0.938705**	0.887424	0.878358	**0.036511**	0.065719	0.070864
(1000, 20, 0.6)	**0.983971**	0.955068	0.956729	**0.938273**	0.844614	0.854032	**0.031441**	0.085971	0.082923
(1000, 20, 0.7)	**0.900786**	0.883085	0.889212	**0.697262**	0.647438	0.661896	**0.179480**	0.208653	0.198344
(1000, 20, 0.8)	**0.188565**	0.169013	0.178392	0.032102	0.030706	**0.032586**	**0.895656**	0.907590	0.901889
$\|V\| = 5000$									
(5000, 20, 0.1)	**0.998699**	0.995858	0.995852	**0.993303**	0.979368	0.981182	**0.002598**	0.008249	0.008261
(5000, 20, 0.2)	**0.997220**	0.994302	0.994357	**0.985554**	0.970421	0.970917	**0.005540**	0.011330	0.011222
(5000, 20, 0.3)	**0.995626**	0.993126	0.992399	**0.978385**	0.963591	0.959471	**0.008707**	0.013653	0.015086
(5000, 20, 0.4)	**0.994701**	0.992059	0.990630	**0.971258**	0.953632	0.945201	**0.010541**	0.015756	0.018565
(5000, 20, 0.5)	**0.994329**	0.989973	0.989782	**0.965862**	0.935363	0.936114	**0.011278**	0.019852	0.020225
(5000, 20, 0.6)	**0.995598**	0.989923	0.989166	**0.967716**	0.932997	0.931082	**0.008751**	0.019952	0.021435
(5000, 20, 0.7)	**0.994403**	0.988913	0.990555	**0.962057**	0.914500	0.931269	**0.011110**	0.021893	0.018713
(5000, 20, 0.8)	**0.331168**	0.311846	0.285959	**0.017992**	0.017609	0.014301	**0.801519**	0.815065	0.832675
(5000, 25, 0.1)	**0.999453**	0.997549	0.997604	**0.997325**	0.988301	0.988875	**0.001094**	0.004889	0.004780
(5000, 25, 0.2)	**0.998614**	0.994864	0.994512	**0.992296**	0.974097	0.972542	**0.002767**	0.010219	0.010916
(5000, 25, 0.3)	**0.998593**	0.993634	0.993192	**0.991259**	0.966756	0.963601	**0.002809**	0.012650	0.013523
(5000, 25, 0.4)	**0.998201**	0.992714	0.991784	**0.988323**	0.959321	0.954605	**0.003589**	0.014465	0.016297
(5000, 25, 0.5)	**0.994865**	0.989942	0.989864	**0.970057**	0.937639	0.940216	**0.010211**	0.019912	0.020066
(5000, 25, 0.6)	**0.995162**	0.989456	0.987228	**0.967998**	0.930606	0.914103	**0.009626**	0.020867	0.025219
(5000, 25, 0.7)	**0.995383**	0.989142	0.988225	**0.967148**	0.922344	0.924432	**0.009187**	0.021477	0.023274
(5000, 25, 0.8)	**0.623561**	0.622423	0.602764	**0.080319**	0.078396	0.069817	**0.543574**	0.544915	0.565846

for solving this kind of task, dues to their feature of producing higher diversity in the population.

3.1 Functional Sensitivity Analysis

As last step of this work, in this subsection, the investigation on the sensitivity of the three community detection methods is reported from functional perspective. The main aim of this analysis is measuring the similarity between the detected communities and original ones. For doing this, commonly used community structure similarity metrics have been considered: (1) *Normalized Mutual Information* (*NMI*) [6], mostly used in community detection, which measures the amount of information correctly extracted, and allows for assessing how similar the detected communities are concerning to real ones; (2) *Adjusted Rand Index* (*ARI*) [13], which focuses on pairwise agreement: for each possible pair of elements it evaluates how similarly the two partitions treat them; and (3) *Normalized Variation of Information* (*NVI*) [18], expressed using the Shannon entropy, which measures the amount of information lost and gained in changing from one clustering to another one: sum of the information needed to describe C, given C', and the information needed to describe C' given C.

The results of the sensitivity analysis are displayed in Table 4 (only for 1000 and 5000 nodes). From this investigation, clearly appears how the method A outperforms the other two in almost all tests performed, uncovering, consequently, more similar communities to the target/real ones, in opposite to the outcomes obtained with respect the modularity evaluation metric. This is caused by the limitation in the modularity optimization which can fail to identify smaller communities; this limitation can depend on the degree of interconnectedness of the communities [9].

4 Conclusions

In this research paper, three different positions where run the local search within an immune algorithm, called Hybrid-IA, have been investigated in order to ascertain which of the three acts best on the algorithm's performance. Community detection problem has been considered for the analysis of this study, and the comparison between the three methods has been conducted with respect the solution quality found and learning process quality. Several artificial networks were generated ($|V| \in \{300, 500, 1000, 5000\}$) through which was possible to inspect the three methods in various complexity scenarios. The obtained outcomes highlight that running the local search just after the hypermutation operator is the best choice for this kind of optimization problem, because in this way higher diversity is produced that help the algorithm specially on larger and complex networks.

References

1. Blum, C., Raidl, G.R.: Hybrid Metaheuristics: Powerful Tools for Optimization. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-319-30883-8
2. Cutello, V., Fargetta, G., Pavone, M., Scollo, R.A.: Optimization algorithms for detection of social interactions. Algorithms **13**(6), 139 (2020). https://doi.org/10.3390/a13060139
3. Cutello, V., Nicosia, G., Pavone, M.: An immune algorithm with stochastic aging and kullback entropy for the chromatic number problem. J. Comb. Optim. **14**, 9–33 (2007). https://doi.org/10.1007/s10878-006-9036-2
4. Cutello, V., Oliva, M., Pavone, M., Scollo, R.A.: A hybrid immunological search for the weighted feedback vertex set problem. In: Matsatsinis, N.F., Marinakis, Y., Pardalos, P. (eds.) LION 2019. LNCS, vol. 11968, pp. 1–16. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-38629-0_1

5. Cutello, V., Oliva, M., Pavone, M., Scollo, R.A.: An immune metaheuristics for large instances of the weighted feedback vertex set problem. In: 2019 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1928–1936, December 2019. https://doi.org/10.1109/SSCI44817.2019.9002988
6. Danon, L., Díaz-Guilera, A., Duch, J., Arenas, A.: Comparing community structure identification. J. Stat. Mech: Theory Exp. **2005**(09), P09008–P09008 (2005). https://doi.org/10.1088/1742-5468/2005/09/p09008
7. Di Stefano, A., Vitale, A., Cutello, V., Pavone, M.: How long should offspring lifespan be in order to obtain a proper exploration? In: 2016 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1–8, December 2016. https://doi.org/10.1109/SSCI.2016.7850270
8. Didimo, W., Montecchiani, F.: Fast layout computation of clustered networks: algorithmic advances and experimental analysis. Inf. Sci. **260**, 185–199 (2014). https://doi.org/10.1016/j.ins.2013.09.048
9. Fortunato, S., Barthélemy, M.: Resolution limit in community detection. Proc. Natl. Acad. Sci. **104**(1), 36–41 (2007). https://doi.org/10.1073/pnas.0605965104
10. Fouladvand, S., Osareh, A., Shadgar, B., Pavone, M., Sharafi, S.: DENSA: an effective negative selection algorithm with flexible boundaries for self-space and dynamic number of detectors. Eng. Appl. Artif. Intell. **62**, 359–372 (2017). https://doi.org/10.1016/j.engappai.2016.08.014
11. Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. Proc. Natl. Acad. Sci. **99**(12), 7821–7826 (2002). https://doi.org/10.1073/pnas.122653799
12. Gulbahce, N., Lehmann, S.: The art of community detection. BioEssays **30**(10), 934–938 (2008). https://doi.org/10.1002/bies.20820
13. Hubert, L., Arabic, P.: Comparing partitions. J. Classif. **2**, 193–218 (1985). https://doi.org/10.1007/BF01908075
14. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. Bell Syst. Tech. J. **49**(2), 291–307 (1970). https://doi.org/10.1002/j.1538-7305.1970.tb01770.x
15. Kullback, S.: Statistics and Information Theory. Wiley, Hoboken (1959)
16. Lancichinetti, A., Fortunato, S.: Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. Phys. Rev. E **80** (2009). https://doi.org/10.1103/PhysRevE.80.016118
17. Lancichinetti, A., Fortunato, S., Radicchi, F.: Benchmark graphs for testing community detection algorithms. Phys. Rev. E **78** (2008). https://doi.org/10.1103/PhysRevE.78.046110
18. Meilă, M.: Comparing clusterings-an information based distance. J. Multivar. Anal. **98**, 873–895 (2007). https://doi.org/10.1016/j.jmva.2006.11.013
19. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E **69** (2004). https://doi.org/10.1103/PhysRevE.69.026113
20. Pavone, M., Narzisi, G., Nicosia, G.: Clonal selection: an immunological algorithm for global optimization over continuous spaces. J. Glob. Optim. **53**, 769–808 (2012). https://doi.org/10.1007/s10898-011-9736-8
21. Vitale, A., Di Stefano, A., Cutello, V., Pavone, M.: The influence of age assignments on the performance of immune algorithms. In: Lotfi, A., Bouchachia, H., Gegov, A., Langensiepen, C., McGinnity, M. (eds.) UKCI 2018. AISC, vol. 840, pp. 16–28. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-97982-3_2