

Where the Local Search Affects Best in an Immune Algorithm

Rocco A. Scollo^[0000-0002-6211-8675], Vincenzo Cutello^[0000-0002-7521-3516], and
Mario Pavone^[0000-0003-3421-3293]

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 [8].

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 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. [3,4,5,2]. 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 Alg. 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 [9,10,7]. 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 [15], 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:

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

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 [14,13] 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. 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 [8]. The presented Hybrid Immune Algorithm, called HYBRID-IA, belongs to the special class *Clonal Selection Algorithms* (CSA) [16,5], 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 [5,2], 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 (\mathbf{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 ($\mathbf{x} \in P^{(t)}$) by using the function $\text{ComputeFitness}(P^{(t)})$. 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 [16]; 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 [6,17].

Algorithm 1 Pseudo-code of HYBRID-IA.

```

1: procedure HYBRID-IA( $d, dup, \rho, \tau_B$ )
2:    $t \leftarrow 0$ 
3:    $P^{(t)} \leftarrow \text{InitializePopulation}(d)$ 
4:    $\text{ComputeFitness}(P^{(t)})$ 
5:   while  $\neg \text{StopCriterion}$  do
6:      $P^{(clo)} \leftarrow \text{Cloning}(P^{(t)}, dup)$ 
7:      $P^{(hyp)} \leftarrow \text{Hypermutation}(P^{(clo)}, \rho)$ 
8:      $\text{ComputeFitness}(P^{(hyp)})$ 
9:      $(P_a^{(t)}, P_a^{(hyp)}) \leftarrow \text{Aging}(P^{(t)}, P^{(hyp)}, \tau_B)$ 
10:     $P^{(select)} \leftarrow (\mu + \lambda)\text{-Selection}(P_a^{(t)}, P_a^{(hyp)})$ 
11:     $P^{(t+1)} \leftarrow \text{LocalSearch}(P^{(select)})$ 
12:     $\text{ComputeFitness}(P^{(t+1)})$ 
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 (7th 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 \mathbf{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 $\alpha = e^{-\rho \hat{f}(\mathbf{x})}$, where ρ determines the shape of the mutation rate, and $\hat{f}(\mathbf{x})$ 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 $P^{(hyp)}$ (9th 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., $\text{age} = \tau_B + 1$), 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, in 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 \mathbf{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 [11]. 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 Equation (1), can be rewritten as:

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

where k is the number of communities identified; $c = \{c_1, \dots, c_i, \dots, c_k\}$ 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:

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

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 [14,13], 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 [16,5,2], 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 [12,3]).

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 Figures 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 (Figures 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 (Figures 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.

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

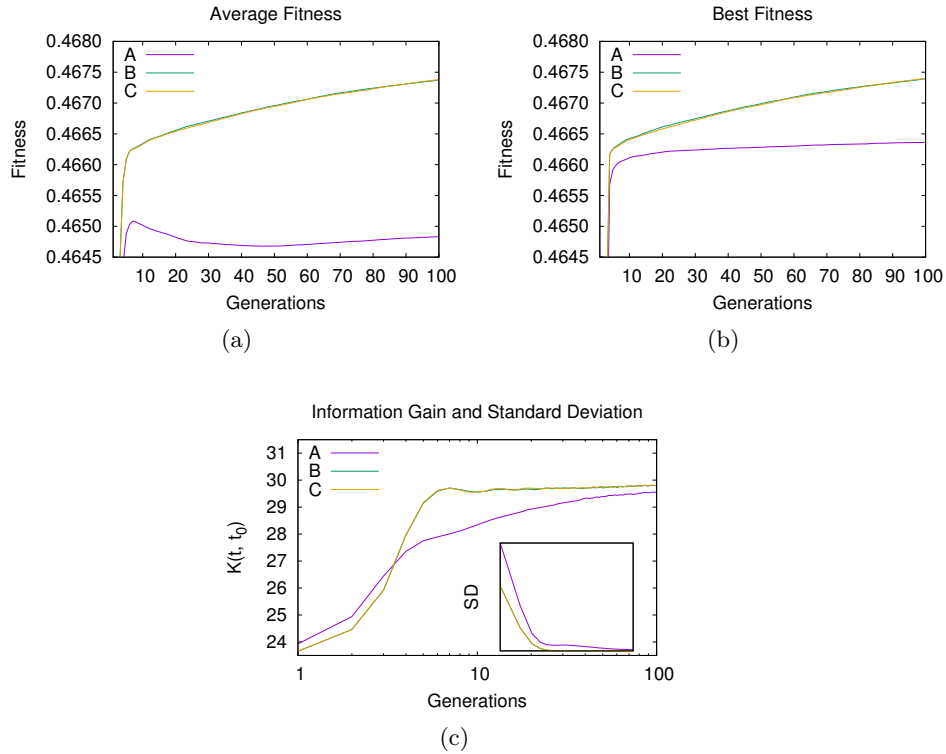


Fig. 1: Convergence behavior of the three methods on LFR(1000, 15, 0.5) graph. (a) Average and (b) best fitness value of the population versus generations. (c) Information gain and standard deviation.

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. The

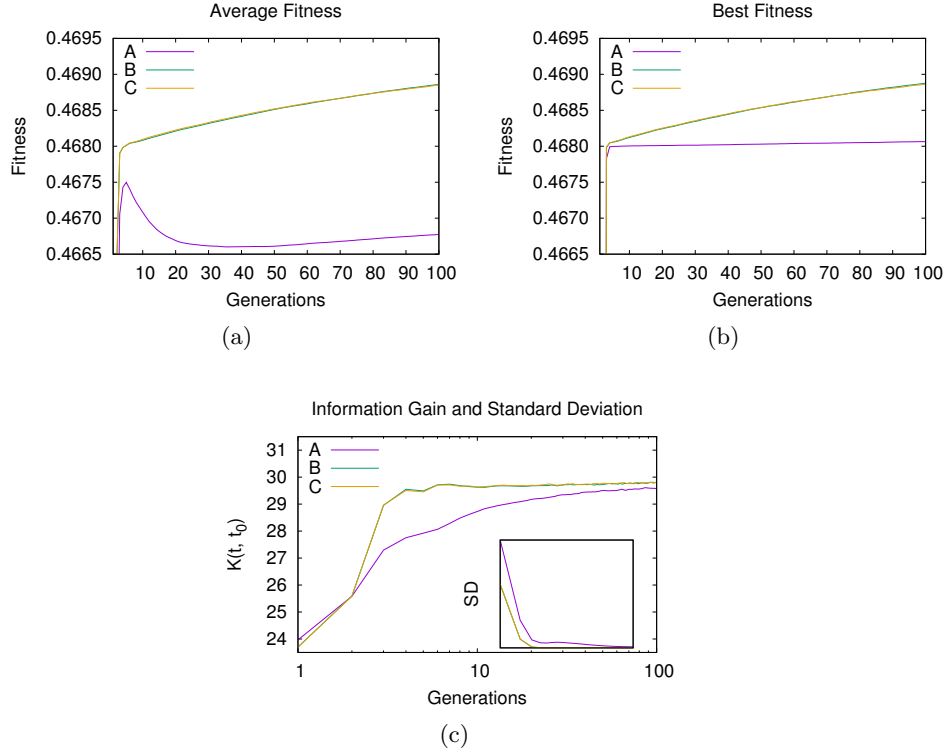


Fig. 2: Convergence behavior of the three methods on LFR(1000, 20, 0.5) graph. (a) Average and (b) best fitness value of the population versus generations. (c) Information gain and standard deviation.

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

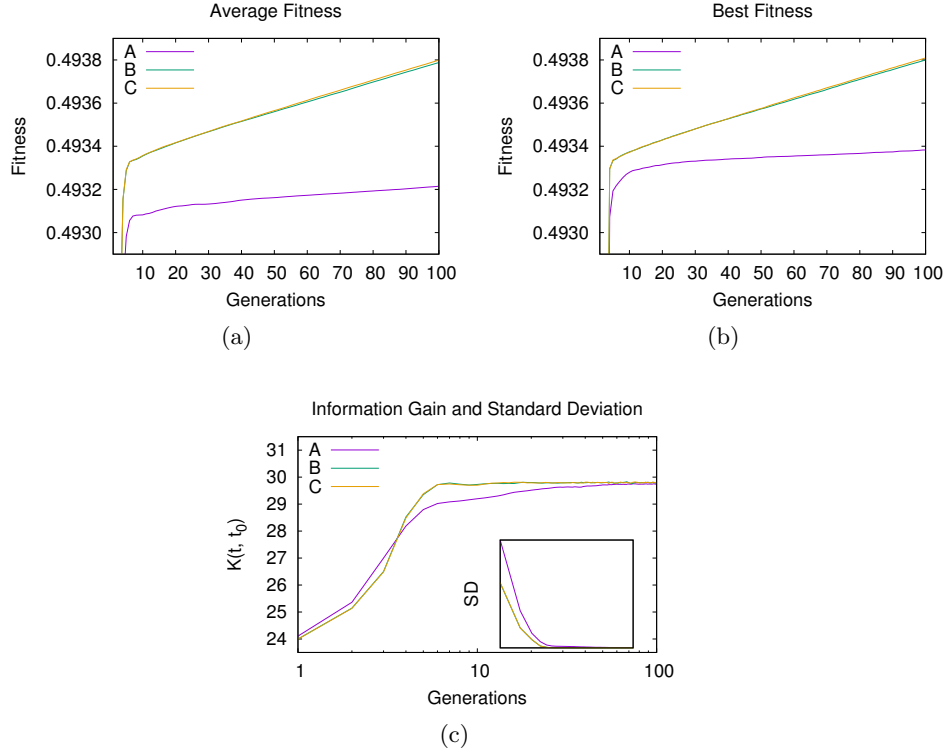


Fig. 3: Convergence behavior of the three methods on LFR(5000, 20, 0.5) graph. (a) Average and (b) best fitness value of the population versus generations. (c) Information gain and standard deviation.

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.

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 as-

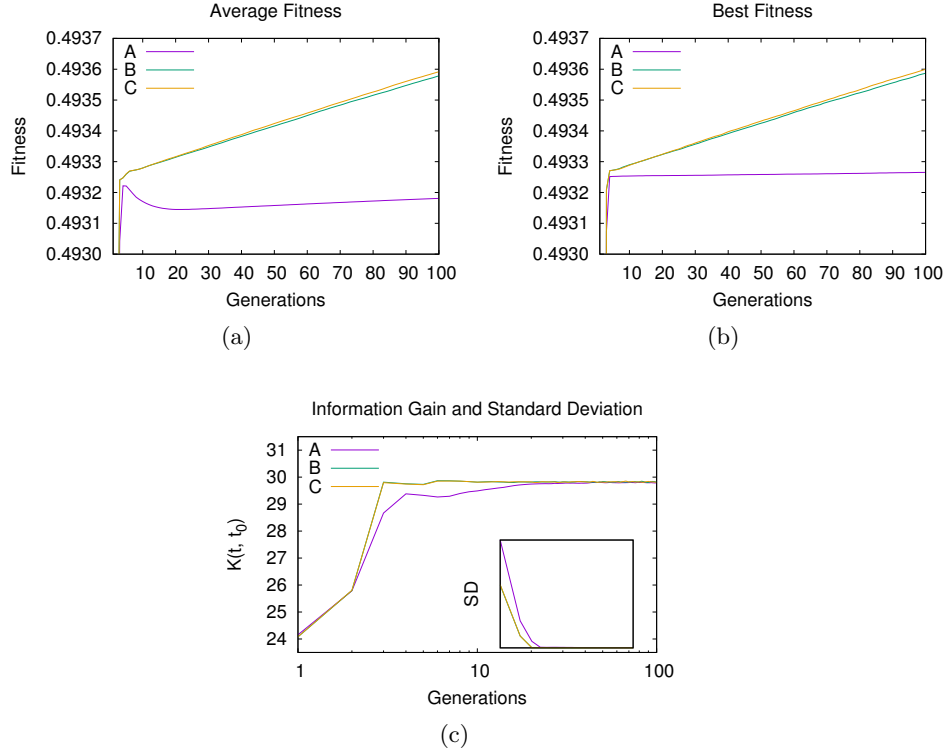


Fig. 4: Convergence behavior of the three methods on LFR(5000, 25, 0.5) graph. (a) Average and (b) best fitness value of the population versus generations. (c) Information gain and standard deviation.

certain 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 (2016)

2. Cutello, V., Fargetta, G., Pavone, M., Scollo, R.A.: Optimization algorithms for detection of social interactions. *Algorithms* **13**(6) (Jun 2020). <https://doi.org/10.3390/a13060139>, 139
3. Cutello, V., Nicosia, G., Pavone, M.: An immune algorithm with stochastic aging and kullback entropy for the chromatic number problem. *Journal of Combinatorial Optimization* **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: 2019 Learning and Intelligent Optimization (LION), Lecture Notes in Computer Science vol. 11968. pp. 1–16 (5 2019). 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 (12 2019). <https://doi.org/10.1109/SSCI44817.2019.9002988>
6. 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 (Dec 2016). <https://doi.org/10.1109/SSCI.2016.7850270>
7. Didimo, W., Montecchiani, F.: Fast layout computation of clustered networks: Algorithmic advances and experimental analysis. *Information Sciences* **260**, 185–199 (2014). <https://doi.org/10.1016/j.ins.2013.09.048>
8. 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. *Engineering Applications of Artificial Intelligence* **62**, 359–372 (2017). <https://doi.org/10.1016/j.engappai.2016.08.014>
9. Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. *Proceedings of the National Academy of Sciences* **99**(12), 7821–7826 (2002). <https://doi.org/10.1073/pnas.122653799>
10. Gulbahce, N., Lehmann, S.: The art of community detection. *BioEssays* **30**(10), 934–938 (2008). <https://doi.org/10.1002/bies.20820>
11. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal* **49**(2), 291–307 (Feb 1970). <https://doi.org/10.1002/j.1538-7305.1970.tb01770.x>
12. Kullback, S.: *Statistics and Information Theory*. John Wiley & Sons (1959)
13. Lancichinetti, A., Fortunato, S.: Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Physical Review E* **80**, 016118 (Jul 2009). <https://doi.org/10.1103/PhysRevE.80.016118>
14. Lancichinetti, A., Fortunato, S., Radicchi, F.: Benchmark graphs for testing community detection algorithms. *Physical Review E* **78**, 046110 (Oct 2008). <https://doi.org/10.1103/PhysRevE.78.046110>
15. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. *Physical Review E* **69**, 026113 (Feb 2004). <https://doi.org/10.1103/PhysRevE.69.026113>
16. Pavone, M., Narzisi, G., Nicosia, G.: Clonal selection: an immunological algorithm for global optimization over continuous spaces. *Journal of Global Optimization* **53**, 769–808 (2012). <https://doi.org/10.1007/s10898-011-9736-8>
17. 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.) *Advances in Computational Intelligence Systems*. pp. 16–28. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-319-97982-3_2