University of Catania
Department of Mathematics and Computer Science
PhD in Computer Science (International)

Rocco Alessandro Scollo

A Hybrid Immune Algorithm to Extract Information from Complex Networks

Doctoral Thesis

Supervisor
Prof. Mario F. Pavone

Academic Year 2021/2022 (Cycle XXXV)

To my beloved wife Maria for her love and support.

Contents

List of Algorithms

List of Figures

List of Tables

Introduction

Metaheuristics represent a family of approximate optimization techniques and provide high-quality solutions in a reasonable time for solving hard and complex problems, sacrificing the guarantee of finding optimal solutions [138]. Metaheuristic search methods can be defined as upper-level general methodologies that can be used to guide the construction of underlying heuristics to solve specific optimization problems.

The concept of (meta)heuristics to solve optimization problems was introduced in the 1940s [114], but their popularity increased in the 1990s due to their efficiency and effectiveness in solving combinatorial optimization problems. Metaheuristics find application in a large number of real fields, such as telecommunications, automotive, and robotics, but also machine learning and data mining for bioinformatics and computational biology, logistical problems, production, scheduling and transport. Metaheuristics are based on two opposing criteria [17, 138]: the exploration of the search space (diversification) and the exploitation of the best solutions found (intensification). In the intensification phase, promising regions are explored more thoroughly in the hope of finding better solutions. On the other hand, in diversification unexplored regions need to be visited to ensure that the search space is uniformly explored so that the search is not limited to a small number of regions.

Initially, pure metaheuristics had considerable success because they quickly became state-of-the-art algorithms for many combinatorial optimization problems. The first two decades of research on metaheuristics have led several research communities to have not much interaction with other optimization research communities. Only when it was evident that pure metaheuristics had reached their limits, many researchers move their attention to the hybridization of metaheuristics [16].

In the field of optimization, interest in hybrid metaheuristics has grown significantly over the last years, obtaining good results not only for classical optimization problems but also for real-life applications. Combinations of algorithms such as metaheuristics, mathematical programming, constraint programming and machine learning techniques have provided very powerful search algorithms [137]. In fact, the main motivation behind the combination of different algorithms is to exploit several complementary optimization concepts to solve many hard optimization problems [13].

A possible hybridization is to combine Reinforcement Learning [84, 136] techniques with metaheuristics methods. The aim of a learning component is to support the optimization method, in order to provide useful information during the exploration of the solutions space. In this way, reinforcement learning causes the optimization algorithm to be adaptive, helping it make decisions or adjust parameters. Moreover, reinforcement learning does not require a complete model of the underlying problem, since they learn the model by gathering experience.

In this thesis work, some contributions are presented to the study and analysis of hybrid metaheuristics to address combinatorial optimization problems on graphs. For many combinatorial optimization problems, there is no prior information about the characteristics of the problem or the properties of the landscape. Consequently, my research has focused on the study and design of a general-purpose framework that combines metaheuristics with local search procedures and/or reinforcement learning to solve combinatorial optimization problems. Two complex problems have been analysed: the *Feedback Vertex Set* on graphs and the problem of *Community Detection* on complex networks. The feedback vertex set has been analyzed using a hybrid immunological algorithm, in which the random exploration of the search space is contained by a local search procedure, that tries to exploit the obtained solution with the purpose to solve sub-instances of the original problem. The problem of community detection is a classical clustering problem that arises from complex network analysis. In this case, two approaches have been developed. The first is a fully random search algorithm driven by stochastic operators designed for the problem of community detection. The second approach is guided by a greedy local search procedure, which locally maximizes the objective function. In addition, an in depth analysis has been conducted to investigate how the position of local search can influence the performance of the algorithm, in terms of exploration of the search space and solution quality.

Finally, a generic framework for grouping problems has been designed. Generally, a grouping problem aims to group a set of given items into a fixed or variable number of groups while respecting some specific requirements. Typical examples of grouping problems are graph colouring and data clustering. The proposed framework combines a population-based greedy metaheuristic with reinforcement learning techniques. This algorithm aims to exploit a learning component to extract useful information on the problem considered starting from a set of high-quality solutions, to guide the search consciously.

The remaining part of the thesis is organized as follows. In Chapter 1 the Feedback Set problems are introduced, which are well-known $\mathcal{NP}$-hard combinatorial optimization problem on graphs. The most general feedback set problem is the Feedback Vertex Set problem, which consists in finding a minimum cardinality set of vertices that meets all cycles in a graph. There are different versions of feedback vertex set problems, depending on whether the graph is directed or undirected and/or whether the vertices are weighted or not. Originating from the area of combinational circuit design, they have found applications in many fields, including deadlock prevention and program verification. Also, this chapter introduces some notations and a formal definition of the feedback vertex set problem that is useful in the following chapters.

In Chapter 2 a hybrid immunological algorithm is presented to solve the weighted variant of the Feedback Vertex Set problem. This algorithm takes inspiration from the immune system, and it is based on three main immune operators (cloning, hypermutation and aging), combined with a local search operator designed with the goal to refine in a deterministic way all solutions produced by these operators. The proposed algorithm has been tested on a dataset of more than 800 graph instances, that differ from each other for topology, dimension, density and weight range, and compared with the other three metaheuristics that represent a good reference point for this problem on the benchmark instances considered. After a convergence analysis of the algorithm, a parameter tuning of the algorithm has been conducted improving the preliminary results already obtained. The preliminary results shown in this chapter have been published in [46, 47], while the results obtained after the parameter tuning have been submitted for publication to the *Networks* journal.

In Chapter 3 an evolutionary framework based on a greedy population combined with reinforcement learning techniques for grouping problems is presented. An immunological algorithm evolves a population of solutions generated in a randomized greedy way, alternating a phase of destruction and a phase of reconstruction, while a probability learning procedure learns which solution components can be useful in the construction of new solutions in order to guide the algorithm within the search space. The Weighted Feedback Vertex Set has been considered as case study and several experiments have been performed on large instances of the problem. Experimental results have shown the goodness of the proposed framework, especially on the complex instances of the benchmark dataset. The framework proposed in this chapter has been submitted to the *Journal of Combinatorial Optimization*.

In Chapter 4 the complex networks and the problem of Community Detection are introduced. Community detection is one of the most interesting research topics in network science since it finds application in several fields that range from social science to biology. The aim of the community detection problem is to detect communities or clusters and their hierarchical organization, using the information encoded in the topology of the network and exploiting, if available, their metadata. But the problem of graph clustering is actually not well defined, because the main concept of community is not defined. In light of this, the modularity function was introduced to evaluate the quality of a partition of vertices of a network, making the problem of community detection a combinatorial grouping problem.

In Chapter 5 an immune-inspired algorithm, Opt-IA, is presented with the aim is to detect the community structure of a complex network, maximizing the modularity of partitions found during the search. Opt-IA is an immunological algorithm based on a fully random-search process, as it is guided by purely stochastic operators. The proposed algorithm was compared with several metaheuristics using social and biological complex networks as a dataset. Furthermore, to analyze the convergence behaviour of Opt-IA in different complexity scenarios, a functional sensitivity analysis was conducted using three community structure similarity metrics on synthetic networks. The preliminary results obtained by Opt-IA have been published in [132], while the results on the social networks in the *Algorithms* journal [41]. The results on biological networks and the functional sensitivity analysis on synthetic networks have been accepted for publication in the *Soft Computing* journal.

In Chapter 6, for the problem of community detection with maximization of modulation, a hybrid immunological algorithm is proposed. The proposed algorithm is based on a deterministic local search that tries to improve the solutions of the current population in order to speed up the convergence of the algorithm. Several experiments have been performed on social, biological and synthetic networks, and the results compared with other metaheuristics, in addition to Opt-IA. Furthermore, the reliability of Hybrid-IA was analyzed with respect to three similarity measures, which show how similar the detected communities are to the real ones. Finally, three different ways of running the local search procedure within Hybrid-IA have been investigated, analyzing the performances in terms of quality solution and information gain. The results obtained by Hybrid-IA on social networks and shown in this chapter have been published in the *Algorithms* journal [41], while the results on biological networks and the functional sensitivity analysis on synthetic networks have been published in the *Informatics* journal [126]. The analysis of the position of the local search procedure has been published in [125].

In Chapter 7 two multi-level models for community detection are compared to reduce and simplify the original graph, because detecting communities and analysing community structure are very computationally expensive tasks, especially on large networks. Both models have been applied to two variants of an immune-inspired algorithm, the first one based on a fully random-search process, and the second based on a hybrid approach. Experimental results show that both multi-level optimization approaches help the immune algorithms to improve the quality of the solution found. The proposed multi-level approaches presented in this chapter have been published in [19].

Finally, in Chapter 8 I provide conclusions of this thesis and an outlook to future works.

1

Weighted Feedback Vertex Set Problem

The Feedback Vertex Set (FVS) problem is a combinatorial optimization task that consists in finding a subset of vertices whose removal from a graph makes it acyclic. This kind of problem is also known as the *hitting cycle problem* since the set of vertices must hit every cycle in the input graph. This problem generalizes a number of problems, including the *minimum feedback vertex (arc) set problem* in both directed and undirected graphs, and the *graph bipartition problem*, in which one must remove a minimum-cardinality (minimum-weight) set of vertices so that the remaining graph is bipartite. All these problems are also special cases of *vertex (arc) deletion problems*, which consists in finding a minimum cardinality (or minimum-weight) set of vertices (arcs) whose deletion gives a graph satisfying a given property.

The FVS problem originated from the area of combinational circuit design [83], where a cycle is identified by some circuit elements that receive a new input before being stabilized, but finds applications in many other real-world fields. Solving the

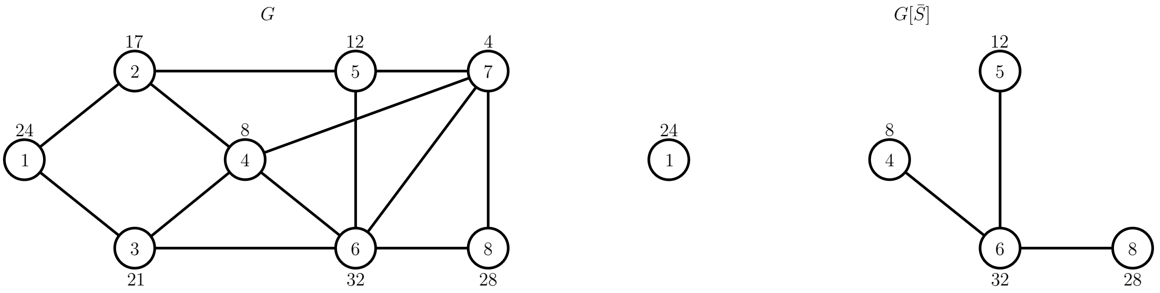
FVS problem plays a prominent role in operating systems and parallel computing for the study of deadlock detection and/or prevention [143]. In the wait-for graph of an operating system, each directed cycle corresponds to a deadlock situation and the feedback vertex set can identify the minimum number of blocked processes that have to be aborted. Also, the FVS problem finds application in complexity theory, where some $\mathcal{NP}$-hard problems on graphs can be solved in polynomial time for graphs with bounded FVS number [87]. Other real-life applications are program verification [127], information security [76] and the study of monopolies in synchronous distributed systems [112].

The problem of finding a feedback vertex set of minimal cardinality, or minimal weight, is known to be a $\mathcal{NP}$-complete problem [85, 151].

1.1 Formal Definition

Before formally defining the feedback vertex set problem, we give some definitions and notations. Let $G = (V, E)$ be an undirected graph, where V is the set of vertices and E is the set of edges. Also, given a set of vertices $X \subseteq V$ of G, we denote with $G[X]$ the subgraph of G induced by the set of vertices X, that is $G[X] = (X, E_{[X]})$, where $E_{[X]} = \{(u, v) \in E : u, v \in X\}$. The degree of a vertex v in the graph G or in the induced graph $G[X]$ is denoted with $d(v)$ and $d_{[X]}(V)$ respectively.

Formally, the feedback vertex set problem can be described as follows. Given an undirected graph, $G = (V, E)$, a feedback vertex set of G is a subset $S \subseteq V$ of vertices such that each cycle in G contains at least one vertex of S. In other words, a feedback vertex set S is a subset of vertices of G whose removal, along with all edges incident to S, results in a forest, i.e., an acyclic graph where all connected components are trees. Also, we can define the residual graph of G generated by S as the subgraph $G[\bar{S}]$ induced by the set of vertices $\bar{S} = V \setminus S$. If $G[\bar{S}]$ is acyclic, then S is a feedback vertex set. If $w : V \to \mathbb{R}^+$ is a function that associates a positive weight to each vertex $v \in V$, then the weight of a feedback vertex set $S \subseteq V$ is the sum of the weights of its vertices, i.e. $\sum_{v \in S} w(v)$. Therefore, the *minimum (weighted) feedback vertex set* problem asks to find a feedback vertex set of minimum-cardinality (minimum-weight). If S is a feedback vertex set, we say that a vertex $v \in S$ is *redundant* if the residual graph $G[\bar{S} \cup \{v\}]$ is still an acyclic graph. It follows that S is a minimal feedback vertex set if it does not contain any redundant vertices. For example, the set $S = \{2, 3, 7\}$ is a feedback vertex set for the graph G in Figure 1.1a, because the residual graph $G[\bar{S}]$ in Figure 1.1b is a forest. The set containing vertices $S = \{1, 2, 3, 7\}$ is also a feedback vertex set but it is not minimal because the vertex 1 is redundant.



(a) (b)

Figure 1.1: A generic graph (a) and the residual graph generated by $S = \{2, 3, 7\}$ (b). S is a feedback vertex set because the residual graph is a forest.

The feedback vertex set problem can be expressed with an integer zero-one programming formulation [60, 61]. Given a feedback vertex S of graph $G = (V, E, w)$, let $x = \{x_v\}_{v \in V}$ be a binary vector such that $x_v = 1$ if $v \in S$, $x_v = 0$ otherwise. Also, let C be the set of cycles in G. The formulation of finding the minimum weighted feedback vertex set of G as an integer programming problem is as follows:

$$\begin{aligned} \min \quad & \sum_{v \in V} w(v) x_v \\ \text{s.t.} \quad & \sum_{v \in \Gamma} x_v \geq 1, \quad \forall \Gamma \in C \\ & x_v \in \{0, 1\}, \quad v \in V. \end{aligned} \tag{1.1}$$

2

Hybrid Immunological Algorithm

In the following we discuss in detail Hybrid-IA, an immunological algorithm developed for the Weighted Feedback Vertex Set problem and belonging to the special class of *Clonal Selection Algorithms* [50, 49, 45]. These types of algorithms take inspiration from the dynamics of clonal selection, a widely accepted theory for modelling the immune system of living organisms. In this model B lymphocytes are able to detect and eliminate foreign entities invading the body, cloning some specific antibodies (`Ab`) that will bind to a specific antigen (`Ag`). At the basis of Hybrid-IA, there is a similar principle in which the antigen (`Ag`) represents the optimization problem, the antibody (`Ab`) represents a point in the solution space (a candidate solution) and the `Ab-Ag` affinity corresponds to the value of the objective function to optimize.

2.1 The Proposed Method

The key operators of the proposed algorithm are the *Cloning* and *Hypermutation* operators: the first simulates the proliferation of the cells, with the main goal to produce a new population of antibodies with higher affinity values, while the second one has the purpose to explore the neighbourhood of each solution into the search space. Moreover, to introduce some diversity in the *memory set* of cells, the *Aging* operator removes old or less promising solutions keeping a high variability of affinity values between the solutions. In addition to these immune operators, a local search procedure will be applied to the survived cells with higher affinity, whose main aim is to refine stochastic moves done during the evolutionary process and try to improve the best solutions found so far.

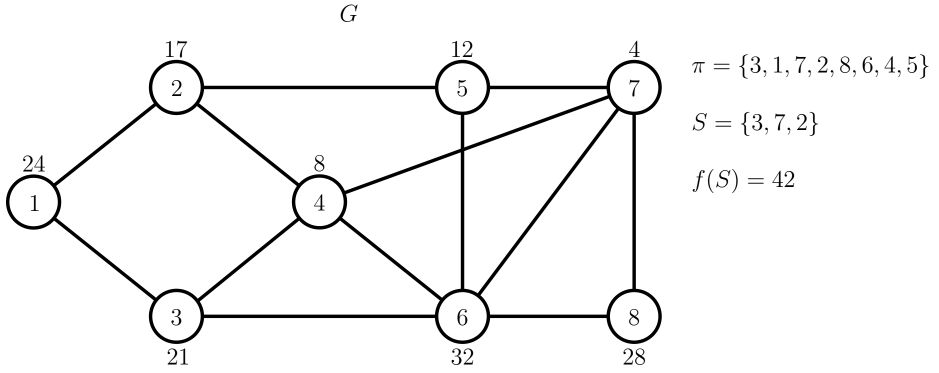


Figure 2.1: A generic weighted graph and a feedback vertex set constructed by a permutation of its vertices. After the removal of the first four vertices of the permutation π, the residual graph is acyclic and thus the subset $S = \{3, 7, 2\}$ is a feedback vertex set. Note that vertex 1 does not belong to the solution S because its degree is 1 after the removal of vertex 3.

2.1.1 Initialization

The proposed algorithm, Hybrid-IA, is based on a population of d antibodies `Ab`, where each cell represents a candidate solution of the weighted feedback vertex set problem. In the initialization phase of the algorithm, each solution of the population is constructed starting from a permutation of the vertices of the graph whose order determines the visiting order of vertices to be removed. In particular, given an undirected graph $G = (V, E)$ and a permutation of its vertices $\pi : \{1, 2, \ldots, |V|\} \rightarrow \{1, 2, \ldots, |V|\}$, the procedures computes a feedback vertex set $S \subseteq V$. In the beginning, the solution S is empty, while the set of the residual graph X contains all vertices of the graph. At each iteration, the procedure removes a vertex $u \in \pi$ from the residual graph $G[X]$, following the order of the permutation, moving it to S. In addition, the algorithm removes from the residual graph $G[X]$ all vertices that have a $d_{[X]}(v) < 2$, since they can not be involved in any cycle and, consequently, they are not relevant in the construction of the solution. The removal process is repeated until the residual graph $G[X]$ does not contain any vertices. In Figure 2.1 is shown a feedback vertex set S for the graph G constructed from a permutation π. In the end, the algorithm removes from the constructed solution S any redundant vertices, going from the heavier vertex to the one with the smallest weight, maintaining the feasibility of the solution and minimizing the value of the objective function. The pseudocode of the described algorithms is shown in Algorithms 2.1 and 2.2.

Algorithm 2.1: Procedure that creates a feedback vertex set starting from a permutation π of the vertices V of a graph G.

```
procedure CreateSolution($G(V, E, w)$, $\pi$)
    $X \leftarrow V$
    $S \leftarrow \emptyset$
    for $u \in \pi$ do
        if $u \in X$ then
            $X \leftarrow X \setminus \{u\}$
            $S \leftarrow S \cup \{u\}$
            while $\exists v \in X : d_{[X]}(v) < 2$ do
                $X \leftarrow X \setminus \{v\}$
            end while
        end if
    end for
    $S \leftarrow$ RemoveRedundantVertices($S$)
    return $S$
end procedure
```

Once a solution is constructed, the permutation will be partitioned reordering the vertices in a such way that all elements belonging to the solution S precede the elements that are not in the solution. This partitioning slightly modifies the permutation and guarantees that the relative ordering of vertices in each group, S and $\bar{S}$, is preserved. In this way any two vertices $u, v \in S$ $(u, v \in \bar{S})$ will have the same relative ordering after the partitioning. Now, the first $l = |S|$ vertices of the permutation π, represent a solution to the problem, and the sum of their weights $\sum_{i=1}^{l} w(\pi_i)$ is the fitness value associated to the permutation.

Algorithm 2.2: Procedure that removes redundant vertices from a feedback vertex set S of a graph G.

```
procedure RemoveRedundantVertices($G(V, E, w)$, $S$)
    $S' \leftarrow S$
    while $S' \neq \emptyset$ do
        $u \leftarrow \mathrm{argmax}_{u \in S'} w(u)$
        $S' \leftarrow S' \setminus \{u\}$
        $X \leftarrow (V \setminus S) \cup \{u\}$
        if $G[X]$ does not contain cycles then
            $S \leftarrow S \setminus \{u\}$
        end if
    end while
    return $S$
end procedure
```

2.1.2 Cloning

After the initialization phase, the evolutionary cycle of Hybrid-IA begins. The *Cloning* operator is the first immune operator applied to the current population $P^{(t)}$, which simply generates a fixed number of clones for each solution of the population. Unlike the classical Clonal Selection Algorithm, where the number of clones is proportional to the fitness value of the solution, Hybrid-IA uses a static version of the operator to avoid a premature convergence of the algorithm. Generating a great number of good solutions in a short time using a proportional version of the cloning operator, which can influence the exploration of the search space due to a population of solutions very similar to each other. Consequently, this may lead the algorithm to prematurely converge towards local optimal. The parameter that regulates how many clones to generate is the duplication factor *dup*, a user-defined parameter. To each solution of the intermediate population $P^{(clo)}$, of size $d \times dup$, is assigned an age that determines how long they can live within the population. In this way, each solution of the population can mature and evolve, trying to produce better solutions, until they reach the maximum age defined by the parameter τ. The age assigned to each solution is chosen randomly in the range $[0 : \frac{2}{3}\tau]$, guaranteeing to each clone to live for at least $\frac{1}{3}\tau$ generations, in the worst case, for evolving and learning. The age assignment and the *Aging* operator, described below, play an important role in the performances of Hybrid-IA because they are able to keep high diversity among the solutions of the populations, avoiding thus premature convergences [54, 141].

2.1.3 Hypermutation

The next immune operator is the *Hypermutation*, which is responsible for the exploration of the neighbourhood of each cloned solution, with the aim to reach promising regions of the search space and produce solutions with better fitness values. The operator works with no mutation probability, unlike other evolutionary algorithms, and performs some mutations on each cloned solution. The number of mutations is determined by a law inversely proportional to the fitness value of each solution considered: the better the fitness value of the solution, the less the number of mutations performed. In this way, the operator carefully explores the neighbourhood of good solutions trying to further improve the fitness value by performing a few mutations, while bad solutions are heavily modified in order to move the search to different points of the solutions space.

Let π be the permutation associated with a candidate solution and $l = |S|$ the cardinality of the solution S constructed from the permutation, as described before. The number of mutations is calculated as follows:

$$M = \lfloor (\alpha \times l) + 1 \rfloor, \tag{2.1}$$

with α that represents the *mutation rate* obtained as

$$\alpha = e^{-\rho \hat{f}(\pi)}, \tag{2.2}$$

where ρ is a user-defined parameter that determines the shape of the curve of the mutation rate, and $\hat{f}$ is the fitness function normalized in the range $[0,1]$. Depending on the value of the mutation shape ρ, the mutation rate α can assume extremely low values, and in this case, the Equation 2.2 guarantees that at least one mutation is performed on best or good solutions.

The basic idea of the hypermutation operator is to perturb the permutation in order to find a new permutation of the vertices of the graph, whose ordering gives a new solution. In light of this, the hypermutation uses the classical *swap* operator, which consists in exchanging (or swapping) the position of two elements of the permutation. In particular, let $\pi = \{\pi_1, \dots, \pi_i, \dots, \pi_l, \dots, \pi_j, \dots, \pi_{|V|}\}$ be a partitioned permutation of a solution, the swap operator chooses randomly two positions $i \in [1, l]$ and $j \in]l, |V|]$ creating a new permutation π' in which the vertices π_i and π_j are exchanged, that is $\pi'_i = \pi_j$ and $\pi'_j = \pi_i$. It is important to note that the new permutation is obtained by swapping two vertices that belong to the two subsets S and $\bar{S}$, that is the swap operator exchanges a vertex in solution with a vertex not in solution. Finally, the procedure described in Algorithm 2.1 constructs a new solution starting from the permutation obtained.

2.1.4 Aging

After the creation by the hypermutation operator of a population of new candidate solutions, the *Aging* operator removes old solutions with the main goal to maintain a high diversity among them. Moreover, this operator allows the algorithm to escape from local optima, jumping to other regions of the search space, and avoiding premature convergence toward suboptimal solutions. The maximum age is defined by the parameter τ, which represents the number of generations allowed to the solutions for maturating within the population. When the age of a solution exceeds the maximum age, it will be removed from the population independently from its fitness value. There is a variant, called *Elitist Aging* operator, that keeps in the population the best current solution, even if its age is older than τ. Depending on the age assignment for each clone in the cloning phase, the aging operator can be applied to both the current and mutated population, i.e. $P^{(t)}$ and $P^{(mut)}$. In the case of HYBRID-IA, the age assigned to the cloned solutions guarantees at least $\frac{1}{3}\tau$ generations, and the aging operator is applied only to the current population $P^{(t)}$.

2.1.5 Selection

After the aging operator, the *Selection operator* generates the new population $P^{(t+1)}$ for the next generation. The replacement strategy used is the $(\mu + \lambda)$-Selection, that consists in selecting the best d solutions from both populations $P^{(t)}$ and $P^{(mut)}$. The age assignment, also, in this case, guarantees that the number of survived solutions that will compete for the new population is greater than d. In this way, the operator ensures that the size of the population is constant during the evolutionary cycle, without the need to generate new candidate solutions.

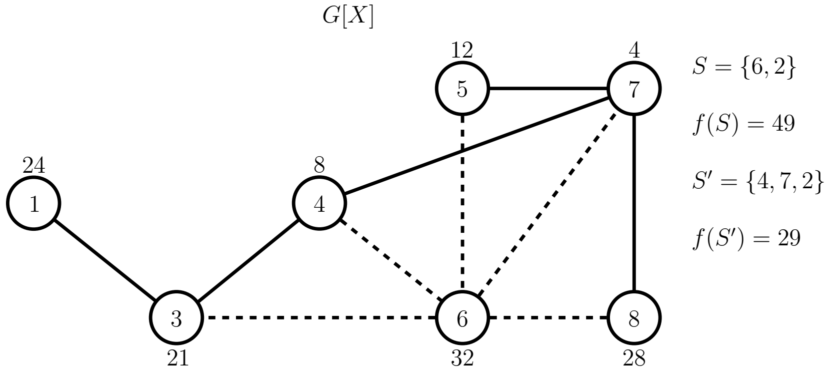


Figure 2.2: A single iteration of the internal loop of the local search procedure for the feedback vertex set problem described in Algorithm 2.3. In this case the vertex 6 of the solution S is replaced by the subset $D_6 = \{4, 7\}$, because $w(D_6) < w(6)$, improving the fitness value of the solution.

2.1.6 Local Search

At the end of the evolutionary cycle, a local search procedure is applied to the selected candidate solutions, with the purpose to refine and improve the solutions that will go to the next generation, and properly speed up the convergence of HYBRID-IA, driving it towards more promising regions [13]. The idea is to explore, in a deterministic way, the neighbourhood of each solution performing some moves that improve the objective function, i.e. to reduce the sum of weights of vertices belonging to a solution. Each solution obtained by the creation solution algorithm is produced starting from a randomly generated permutation that does not take into account the weight of the vertices. For this reason, the local search tries to reduce the weight of a solution by replacing one of its vertices with a set of vertices from the residual graph. In more detail, let S be a solution without redundant vertices of the graph G. The local search procedure starts moving the vertex with the largest weight from the solution, that is $u = \text{argmax}_{u \in S}\, w(u)$, to the residual graph $G[\bar{S}]$, generating a new graph $G' = G[\bar{S} \cup \{u\}]$. Since S does not contain any redundant vertices, u restores one or more cycles in G' and all those cycles pass through u. Using the simple Depth First Search (DFS) [34], the algorithm starts a visit from u to identify a cycle $\mathit{\Gamma}$ along with all vertices involved in it; thus, the algorithm breaks off the cycle $\mathit{\Gamma}$ removing from the residual graph $G[X]$ the vertex v with smaller weight–degree ratio, that is $v = \text{argmin}_{v \in \mathit{\Gamma}}\, w(v)/d_{[X]}(v)$, where $X = \bar{S} \cup \{u\}$. This process is repeated until the residual graph $G[X]$ does not contain any cycles. At the end of the iteration, D_u contains all removed vertices and represents a feedback vertex set for the graph G'. If the sum of the weights of the removed vertices $w(D_u)$ is less than the weight of vertex u, there is an improvement of the fitness value and then the procedure replaces D_u with u in the solution S. Otherwise, the local search rebuilds the residual graph $G[\bar{S}]$ and repeats this process for all the vertices of S. For example, in Figure 2.2 the vertex 6 can be replaced by the subset $D_6 = \{4, 7\}$ improving the objective function and maintaining the feasibility of the solution. Finally, if there was at least a replacement of a vertex u with a subset D_u, the local search performs a redundancy control for the vertices of the new solution. The pseudocode of the local search is shown in Algorithm 2.3.

Algorithm 2.3: Pseudo-code of the local search procedure that tries to improve a feedback vertex set S.

```
procedure LocalSearch($G(V, E, w)$, $S$)
    $S' \leftarrow S$
    while $S' \neq \emptyset$ do
        $u \leftarrow \text{argmax}_{u \in S'} w(u)$
        $S' \leftarrow S' \setminus \{u\}$
        $X \leftarrow (V \setminus S) \cup \{u\}$
        $D_u \leftarrow \emptyset$
        while $G[X]$ contains cycles do
            Select a cycle $\mathit{\Gamma}$ in $G[X]$ starting from $u$
            $v \leftarrow \text{argmin}_{v \in \mathit{\Gamma}} w(v)/d_{[X]}(v)$
            $X \leftarrow X \setminus \{v\}$
            $D_u \leftarrow D_u \cup \{v\}$
        end while
        if $w(D_u) < w(u)$ then
            $S \leftarrow S \setminus \{u\}$
            $S \leftarrow S \cup D_u$
        end if
    end while
    $S \leftarrow$ RemoveRedundantVertices($S$)
    return $S$
end procedure
```

2.1.7 Termination

Hybrid-IA iteratively repeats the immune operators and the local search procedure on the current population until a termination criterion is satisfied. In this research work, the algorithm ends the execution after a fixed number of generations, defined by the parameter T_{Max}, or if it reaches the optimal value if known. In Algorithm 2.4 is shown the pseudocode of Hybrid-IA.

2.2 Experimental Results

To evaluate the effectiveness and competitiveness of Hybrid-IA, in this section all analysis, parameters tuning and experimental results carried out on the algorithm are presented. The proposed algorithm was implemented in C++ using LEMON [53] library, an open-source template library to handle data structures of the input graph

Algorithm 2.4: Pseudo-code of Hybrid-IA.

```
procedure Hybrid-IA($d$, $dup$, $\rho$, $\tau$)
    $t \leftarrow 0$
    $P^{(t)} \leftarrow$ InitializePopulation($d$)
    ComputeFitness($P^{(t)}$)
    while ¬StopCriterion do
        $P^{(clo)} \leftarrow$ Cloning($P^{(t)}, dup$)
        $P^{(mut)} \leftarrow$ Hypermutation($P^{(clo)}, \rho$)
        ComputeFitness($P^{(mut)}$)
        $P^{(t)} \leftarrow$ Aging($P^{(t)}, \tau$)
        $P^{(t+1)} \leftarrow (\mu + \lambda)-$Selection($P^{(t)}, P^{(mut)}$)
        LocalSearch($P^{(t+1)}$)
        ComputeFitness($P^{(t+1)}$)
        $t \leftarrow t + 1$
    end while
end procedure
```

instances, and compiled with GCC 9.4. All the following experiments were performed on a Linux machine equipped with an Intel Xeon E5-2620 processor at 2.40 GHz and 16 GB of memory.

To accurately evaluate the performance of the proposed algorithm on different application fields, a set of benchmark instances, originally proposed in [30], has been considered for all experiments. This benchmark set consists of five classes of undirected vertex-weighted graphs with a different number of vertices, density and weight range: random graphs, squared and not squared grid graphs, toroidal graphs and hypercube graphs. The class of random and hypercube graphs are characterized by the number of vertices n and the number of edges m, while the class of grid and toroidal graphs by the coordinates x and y, which determine the number of vertices. The weight $w(v)$ for each vertex $v \in V$ is randomly extracted from an uniform distribution from the intervals $[10, 25]$, $[10, 50]$ and $[10, 75]$. For each combination of n, m and weight range, or x, y and weight range, there are 5 problem instances that differ only in the assignment of the weights of the vertices. Finally, depending on the number of vertices, these instances are grouped into two classes: small instances, with about 25, 50 and 75 vertices, and large instances with about 100, 200, 300, 400 and 500 vertices.

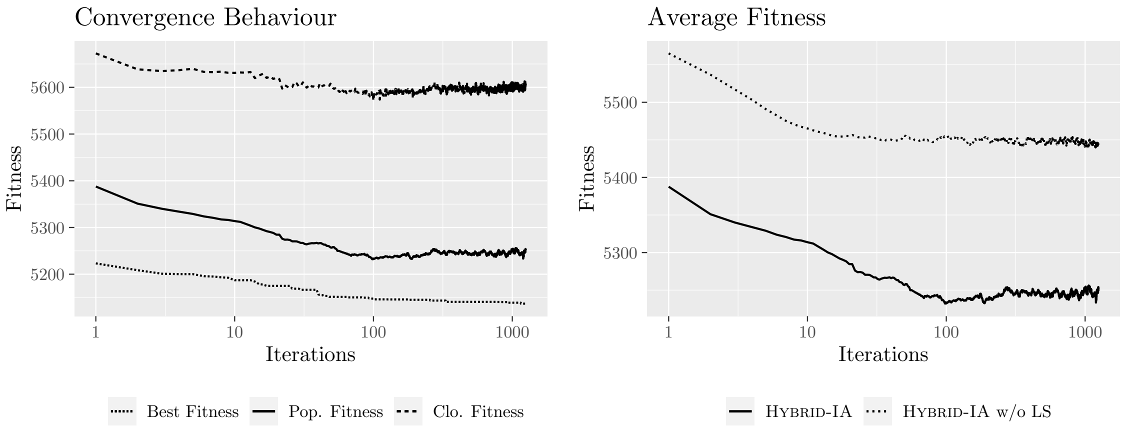
For the comparisons outlined in the following section, three different algorithms have been considered, which represent nowadays a good reference point for this problem on the benchmark instances considered: the *Iterated Tabu Search* (ITS) [30], the *eXploring Tabu Search* (XTS) [24, 52] and a *Memetic Algorithm* (MA) [28]. All these three algorithms use a *k-diamond* graph [29] to represent the neighbourhood of candidate solutions. More precisely, to explore this neighbourhood XTS uses an approximation algorithm [6], while ITS and MA use a dynamic programming algorithm [29] that is able to solve the weighted feedback problem on the k-diamond graphs. Note that, as described in Section 2.1.6, the local search operator involved in the evolutionary cycle of Hybrid-IA, explores the same neighbourhood, but in this case with a more simple heuristic to solve the problem for the sub-instance.

2.2.1 Convergence Behaviour

In the first part of the experimental phase, the analysis has been focused on the convergence behaviour of the proposed algorithm in order to investigate the performances of designed operators for this combinatorial optimization problem.

In Figure 2.3a is shown the convergence behaviour of Hybrid-IA on the random instance `L_R15`, that is an instance with 200 vertices and medium density (see Table 2.12). In particular, the three curves represent the average fitness value of the population, the average fitness value of the mutated population and the best fitness value discovered so far by the algorithm. From this plot can be seen that Hybrid-IA has a good convergence in the first 100 iterations, both from the point of view of the average fitness of the population and the best solution. In subsequent iterations, the average fitness of the population remains almost constant, with variations due to the aging operator, which discards old solutions introducing diversity from the mutated population. This diversity among solutions is also confirmed by the distance that the three curves maintain for the rest of the iterations.

The curves in Figure 2.3b represent the average fitness value of the current population with and without the local search operator described in Section 2.1.6. This plot



(a) **(b)**

Figure 2.3: Convergence behaviour of Hybrid-IA on random instance `L_R15`. (a) Average fitness function values of current population $P^{(t)}$, mutated population $P^{(mut)}$ and the best solution. (b) Average fitness function values of current population $P^{(t)}$ with and without local search procedure.

shows how the local search affects the convergence of the proposed algorithm, improving the exploration of the search space. Moreover, the local search operator helps the algorithm to keep a high degree of diversity among the candidate solutions, as can be seen from Figure 2.4a, which shows the curves of the standard deviation of the two populations. This is useful for avoiding and/or escaping from local optima.

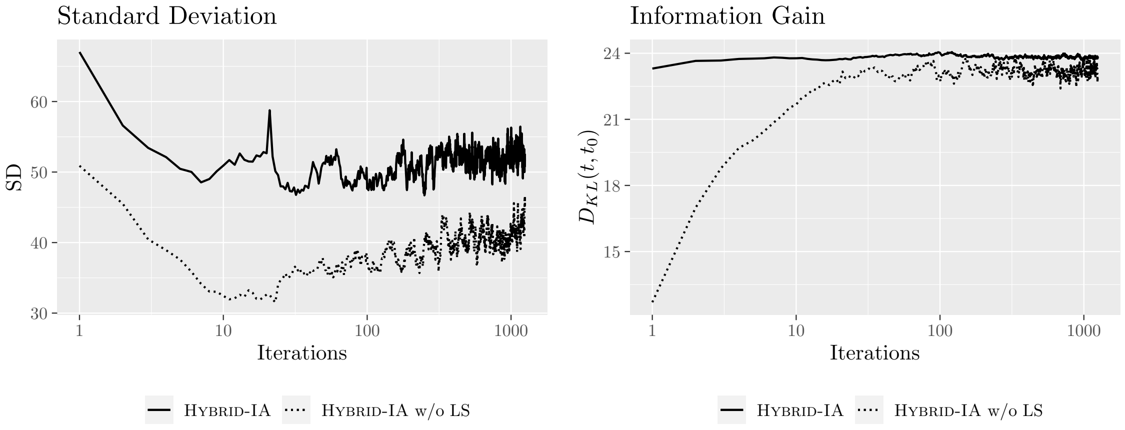
In addition to the convergence analysis, the learning ability of the algorithm was also investigated, using the Kullback-Leibler divergence [89, 90] as a measure of the information gained during the evolutionary process. The idea is to measure how the current population $P^{(t)}$ is different from the initial population $P^{(t_0)}$. Let $S_i^{(t)}$ be the number of solutions S that at iteration t have the fitness function value i; the candidate solutions distribution function $f_i^{(t)}$ can be defined as the ratio between the number $S_i^{(t)}$ and the total number of candidate solutions, that is:

$$f_i^{(t)} = \frac{S_i^{(t)}}{|P^{(t)}|}. \tag{2.3}$$

It follows that the Kullback-Leibler divergence $D_{KL}(t, t_0)$ can be calculated as:

$$D_{KL}(t, t_0) = \sum_i f_i^{(t)} \log \left(\frac{f_i^{(t)}}{f_i^{(t_0)}} \right). \tag{2.4}$$

The plot in Figure 2.4b shows the curves of the information gain obtained by the algorithm with and without the use of the local search operator. Also, in this case, it is clear how the local search helps the algorithm to learn more information from the very first iterations, reaching a peak after 100 iterations and keeping this quantity of information gained for the rest of the execution. On the other side, instead, the curve of the information gained by the population without the local search grows much more slowly and begins to have an oscillatory trend after about 100 iterations. Moreover, this curve does not reach the information gain of the population with local search, keeping on average a certain distance, except on rare occasions.



(a) **(b)**

Figure 2.4: Learning behaviour of Hybrid-IA during evolutionary cycle on random instance `L_R15`. (a) Standard deviation of fitness function and (b) information gain of current population $P^{(t)}$ with and without local search procedure.

2.2.2 Preliminary Results

In this section, all preliminary outcomes obtained by Hybrid-IA are presented. For all preliminary experiments presented, the parameters setting of Hybrid-IA were, respectively: population size $d = 100$; duplication parameter $dup = 2$; maximum age reachable $\tau = 20$; mutation rate parameter $\rho = 0.5$ for small instances, $\rho = 1.3$ for instances with $|V| = 100$ vertices, and $\rho = 3.0$ otherwise. The values of these parameters have been determined through experimental tests and historical knowledge learned on the algorithm.

Small Instances

Tables 2.1 and 2.2 show the results obtained by Hybrid-IA on different sets of instance. In both tables, the first six columns represent the instance, with its name (*Id*), the size of the graph (x and y for grid and toroidal graphs, n and m for random and hypercube graphs), the lower and upper bound of the weight range (*Low* and *Up*), and the optimal solution (*Opt*). In the next columns are reported the results of Hybrid-IA and the other three algorithms compared. Further, in each line of the tables, the best results among all are reported in boldface.

On the squared grid graph instances (top of Table 2.1) is possible to see how Hybrid-IA is able to reach the optimal solution in 8 instances over 9, unlike of MA that instead reaches it in all instances. However, in this instance (`S_SG7`) the performances showed by Hybrid-IA are very close to the optimal solution (+0.2), and anyway better than the other two compared algorithms. On the not squared grid graphs (middle of Table 2.1) instead Hybrid-IA reaches the optimal solution on all instances (9 over 9), outperforming all three algorithms on the instance `S_NG7`, where none of the three compared algorithms is able to find the optimal solution. Also on the toroidal graphs (bottom of Table 2.1) Hybrid-IA is able to reach the global optimal solution in 9 over 9 instances. In the overall, inspecting all results in table 2.1 is possible to see how Hybrid-IA shows competitive and comparable performances to MA algorithm, except in the instance `S_SG7` where it shows slight worst results, whilst instead, it is able to outperform MA in the instance `S_NG7` where it reaches the optimal solution unlike of MA. Analysing the results with respect to the other two algorithms is clear how Hybrid-IA outperform ITS and XTS in all instances.

In Table 2.2 are presented the comparisons on the hypercube and random graphs,

Table 2.1: Preliminary test results of Hybrid-IA on the small instances of squared grid, not squared grid and toroidal graphs.

	Instance								
Id	x	y	Low	Up	Opt	Hybrid-IA	ITS	XTS	MA
Squared Grid									
S_NG1	8	3	10	25	96.8	**96.8**	**96.8**	**96.8**	**96.8**
S_NG2	8	3	10	50	157.4	**157.4**	**157.4**	**157.4**	**157.4**
S_NG3	8	3	10	75	220.0	**220.0**	**220.0**	**220.0**	**220.0**
S_NG4	9	6	10	25	295.6	**295.6**	295.8	295.8	**295.6**
S_NG5	9	6	10	50	488.6	**488.6**	489.4	**488.6**	**488.6**
S_NG6	9	6	10	75	755.0	**755.0**	**755.0**	755.2	**755.0**
S_NG7	12	6	10	25	398.2	**398.2**	399.8	398.8	398.4
S_NG8	12	6	10	50	671.8	**671.8**	673.4	**671.8**	**671.8**
S_NG9	12	6	10	75	1015.2	**1015.2**	1017.4	1015.4	**1015.2**
Not Squared Grid									
S_SG1	5	5	10	25	114.0	**114.0**	**114.0**	**114.0**	**114.0**
S_SG2	5	5	10	50	199.8	**199.8**	**199.8**	**199.8**	**199.8**
S_SG3	5	5	10	75	312.4	**312.4**	312.6	**312.4**	**312.4**
S_SG4	7	7	10	25	252.0	**252.0**	252.4	**252.0**	**252.0**
S_SG5	7	7	10	50	437.6	**437.6**	439.8	**437.6**	**437.6**
S_SG6	7	7	10	75	713.6	**713.6**	718.4	717.4	**713.6**
S_SG7	9	9	10	25	442.2	442.4	444.2	442.8	**442.2**
S_SG8	9	9	10	50	752.2	**752.2**	754.6	753.0	**752.2**
S_SG9	9	9	10	75	1134.4	**1134.4**	1138.0	**1134.4**	**1134.4**
Toroidal									
S_T1	5	5	10	25	101.4	**101.4**	**101.4**	**101.4**	**101.4**
S_T2	5	5	10	50	124.4	**124.4**	**124.4**	**124.4**	**124.4**
S_T3	5	5	10	75	157.8	**157.8**	**157.8**	158.8	**157.8**
S_T4	7	7	10	25	195.4	**195.4**	197.4	**195.4**	**195.4**
S_T5	7	7	10	50	234.2	**234.2**	**234.2**	**234.2**	**234.2**
S_T6	7	7	10	75	269.6	**269.6**	**269.6**	**269.6**	**269.6**
S_T7	9	9	10	25	309.6	**309.8**	310.4	**309.8**	**309.8**
S_T8	9	9	10	50	369.6	**369.6**	370.0	**369.6**	**369.6**
S_T9	9	9	10	75	431.8	**431.8**	432.2	432.2	**431.8**

which present larger problem dimensions with respect to the previous ones. Analysing the results obtained on the hypercube graph instances, it is very clear how Hybrid-IA outperform all three algorithms in all instances (9 over 9), reaching even the optimal solution on the S_H7 instance where instead the three algorithms fail. On the random graphs, Hybrid-IA still shows comparable results to MA on all instances, even reaching the optimum on the instances S_R20 and S_R23 where instead MA, and the other two algorithms fail. In the overall, analyzing all results of this table is easy to assert that Hybrid-IA is comparable, and sometimes the best, with respect to MA also on this set of instances, winning even on three instances the comparison with it. Extending the analysis to the comparison with ITS and XTS algorithms, it is quite clear that Hybrid-IA outperforms them in all instances, finding always better solutions than these two compared algorithms.

Large Instances

The comparisons on the random graphs are reported in Table 2.3. For this comparison instances with different problem dimensions have been taken into account (from 100 to 500 nodes); with different densities of edges; and different ranges of weights. This helps us in testing the efficiency of Hybrid-IA on different degrees of problem complexity, and different optimization scenarios. Inspecting the Table 2.3, it is possible to assert that the proposed immune metaheuristic outperforms all compared algorithms, reaching the best solution in 27 instances over 36, and improving the best-known value so far (i.e. K^*) in 11 of these instances. The MA algorithm, which is based on a genetic algorithm, is instead able to reach the best solution in 17 instances over 36, and only in 5 of them, it is able to find better solutions than Hybrid-IA and the other two algorithms. About the other two algorithms, XTS is able to reach the best solutions in 16 instances, and in 4 of these it reaches better solutions than the other compared algorithms; whilst ITS finds the best solution only on 2 instances. It is important to emphasize that: (1) in all those instances where Hybrid-IA reaches better values than K^*, the improvement is almost always quite considerable; (2) in those instances where Hybrid-IA is not able to find the best solutions, it is, anyway, always the second best; (3) if Hybrid-IA is compared to MA in all those instances where both are not able to find the best solution, i.e. where XTS is the winner, anyway the proposed immune

Table 2.2: Preliminary test results of HYBRID-IA on the small instances of hypercube and random graphs.

	INSTANCE								
Id	n	m	Low	Up	Opt	HYBRID-IA	ITS	XTS	MA
					HYPERCUBE				
S_H1	16	32	10	25	72.2	**72.2**	**72.2**	**72.2**	**72.2**
S_H2	16	32	10	50	93.8	**93.8**	**93.8**	**93.8**	**93.8**
S_H3	16	32	10	75	97.4	**97.4**	**97.4**	**97.4**	**97.4**
S_H4	32	80	10	25	170.0	**170.0**	**170.0**	**170.0**	**170.0**
S_H5	32	80	10	50	240.6	**240.6**	241.0	**240.6**	**240.6**
S_H6	32	80	10	75	277.6	**277.6**	**277.6**	**277.6**	**277.6**
S_H7	64	192	10	25	353.4	**353.4**	354.6	353.8	353.8
S_H8	64	192	10	50	475.6	**475.6**	476.0	**475.6**	**475.6**
S_H9	64	192	10	75	503.8	**503.8**	**503.8**	504.8	**503.8**
					RANDOM				
S_R1	25	33	10	25	63.8	**63.8**	**63.8**	**63.8**	**63.8**
S_R2	25	33	10	50	99.8	**99.8**	**99.8**	**99.8**	**99.8**
S_R3	25	33	10	75	125.2	**125.2**	**125.2**	**125.2**	**125.2**
S_R4	25	69	10	25	157.6	**157.6**	**157.6**	**157.6**	**157.6**
S_R5	25	69	10	50	272.2	**272.2**	**272.2**	**272.2**	**272.2**
S_R6	25	69	10	75	409.4	**409.4**	**409.4**	**409.4**	**409.4**
S_R7	25	204	10	25	273.4	**273.4**	**273.4**	**273.4**	**273.4**
S_R8	25	204	10	50	507.0	**507.0**	**507.0**	**507.0**	**507.0**
S_R9	25	204	10	75	785.8	**785.8**	**785.8**	**785.8**	**785.8**
S_R10	50	85	10	25	174.6	**174.6**	175.4	176.0	**174.6**
S_R11	50	85	10	50	280.8	**280.8**	**280.8**	281.6	**280.8**
S_R12	50	85	10	75	348.0	**348.0**	**348.0**	349.2	**348.0**
S_R13	50	232	10	25	386.2	**386.2**	389.4	386.8	**386.2**
S_R14	50	232	10	50	708.6	**708.6**	**708.6**	**708.6**	**708.6**
S_R15	50	232	10	75	951.6	**951.6**	**951.6**	**951.6**	**951.6**
S_R16	50	784	10	25	602.0	**602.0**	602.2	**602.0**	**602.0**
S_R17	50	784	10	50	1171.8	**1171.8**	1172.2	1172.0	**1171.8**
S_R18	50	784	10	75	1648.8	**1648.8**	1649.4	**1648.8**	**1648.8**
S_R19	75	157	10	25	318.2	**318.2**	321.0	320.0	**318.2**
S_R20	75	157	10	50	521.6	**522.2**	526.2	525.0	522.6
S_R21	75	157	10	75	751.0	**751.0**	757.2	754.2	**751.0**
S_R22	75	490	10	25	635.8	**635.8**	638.6	**635.8**	**635.8**
S_R23	75	490	10	50	1226.6	**1226.6**	1230.6	1228.6	1227.6
S_R24	75	490	10	75	1789.4	**1789.4**	1793.6	**1789.4**	**1789.4**
S_R25	75	1739	10	25	889.8	**889.8**	891.0	**889.8**	**889.8**
S_R26	75	1739	10	50	1664.2	**1664.2**	1664.8	**1664.2**	**1664.2**
S_R27	75	1739	10	75	2452.2	**2452.2**	2452.8	**2452.2**	**2452.2**

Table 2.3: Preliminary test results of Hybrid-IA on large instances of random graphs.

	Instance								
Id	n	m	Low	Up	K^*	Hybrid-IA	ITS	XTS	MA
L_R1	100	247	10	25	498.4	**498.4**	501.4	500.8	**498.4**
L_R2	100	247	10	50	836.8	**833.8**	845.8	840.0	836.8
L_R3	100	247	10	75	1207.6	**1207.2**	1223.8	1208.0	1207.6
L_R4	100	841	10	25	826.8	**826.8**	828.2	**826.8**	**826.8**
L_R5	100	841	10	50	1724.4	**1724.4**	1729.6	1724.6	**1724.4**
L_R6	100	841	10	75	2420.6	**2420.6**	2425.6	**2420.6**	**2420.6**
L_R7	100	3069	10	25	1134.0	**1134.0**	**1134.0**	**1134.0**	**1134.0**
L_R8	100	3069	10	50	2179.0	**2179.0**	**2179.0**	**2179.0**	**2179.0**
L_R9	100	3069	10	75	3228.6	**3228.6**	3228.8	3228.8	**3228.6**
L_R10	200	796	10	25	1468.2	**1466.6**	1488.4	1468.8	1468.2
L_R11	200	796	10	50	2399.0	2400.0	2442.6	2414.4	**2399.0**
L_R12	200	796	10	75	3089.6	**3087.0**	3157.0	3099.6	3089.6
L_R13	200	3184	10	25	1986.2	1987.2	2003.6	1986.8	**1986.2**
L_R14	200	3184	10	50	3650.6	**3649.2**	3683.6	3650.6	3651.8
L_R15	200	3184	10	75	5135.8	**5133.8**	5158.6	5137.2	5135.8
L_R16	200	12139	10	25	2447.8	**2447.8**	2450.0	2448.4	**2447.8**
L_R17	200	12139	10	50	4148.6	**4148.6**	4149.4	**4148.6**	4149.0
L_R18	200	12139	10	75	5528.4	**5528.4**	5531.4	**5528.4**	**5528.4**
L_R19	300	1644	10	25	2045.4	**2044.8**	2072.6	2045.4	2048.0
L_R20	300	1644	10	50	4175.4	4177.4	4239.4	4195.2	**4175.4**
L_R21	300	1644	10	75	6065.2	6072.4	6154.4	6102.8	**6065.2**
L_R22	300	7026	10	25	3203.0	3203.4	3231.0	**3203.0**	3207.6
L_R23	300	7026	10	50	6211.0	6214.2	6261.4	**6211.0**	6217.2
L_R24	300	7026	10	75	8585.4	**8573.2**	8660.6	8585.4	8613.2
L_R25	300	27209	10	25	3726.6	**3726.6**	3729.2	**3726.6**	**3726.6**
L_R26	300	27209	10	50	5734.8	**5734.8**	5738.0	**5734.8**	**5734.8**
L_R27	300	27209	10	75	10467.0	**10467.0**	10469.6	**10467.0**	**10467.0**
L_R28	400	2793	10	25	2989.6	**2987.2**	3015.2	2991.0	2989.6
L_R29	400	2793	10	50	6410.0	6421.6	6528.0	6435.8	**6410.0**
L_R30	400	2793	10	75	8597.2	**8581.2**	8730.0	8637.0	8597.2
L_R31	400	12369	10	25	4428.8	4434.0	4451.8	**4428.8**	4437.4
L_R32	400	12369	10	50	6785.8	6792.4	6837.4	**6785.8**	6800.6
L_R33	400	12369	10	75	10599.4	**10599.2**	10661.8	10599.4	10601.0
L_R34	400	48279	10	25	5060.4	**5060.4**	5060.8	**5060.4**	5060.6
L_R35	400	48279	10	50	7106.8	**7106.8**	7109.2	**7106.8**	7108.0
L_R36	400	48279	10	75	15103.2	**15103.2**	15114.6	**15103.2**	15117.8

metaheuristic find always better solutions than MA.

In Tables 2.4 and 2.5 are reported the outcomes on the hypercube and toroidal graphs. These instances present graphs dimension from 100 to 512 nodes, and from 100 to 529 vertices, respectively. For the hypercube instances in Table 2.4, it is quite clear how Hybrid-IA reaches the optimal solution in all instances (9 instances in the overall), improving the best-known value in 6 of these instances. Also on the toroidal graphs in Table 2.5, is possible to see how Hybrid-IA outperforms the compared algorithms finding the best solution in 10 instances over 15, unlike of MA that is able to reach the best solution only in 7 instances. XTS and ITS, instead, never have been able to reach the best solutions in any instance, except in L_T3 instance for XTS. Moreover, whilst MA finds better solutions than Hybrid-IA in 5 instances, our immune metaheuristic not only outperforms MA in 8 instances, but it improves significantly the best-known values on them. It is worth emphasizing that, due to the particular topology of the toroidal graphs, the local search finds some difficulties primarily when acting on the selection of a new vertex based on its degree, or weight-degree ratio, since the vertices tend to have all the same degree. A possible heuristic approach is to modify the mutation rate (see Section 2.2.3).

Table 2.4: Preliminary test results of Hybrid-IA on large instances of hypercube graphs.

	Instance								
Id	n	m	Low	Up	K^*	Hybrid-IA	ITS	XTS	MA
L_H1	128	448	10	25	731.8	**731.4**	740.0	742.0	731.8
L_H2	128	448	10	50	1066.8	**1066.8**	1071.0	**1066.8**	1067.2
L_H3	128	448	10	75	1161.6	**1161.6**	1163.6	**1161.6**	1162.4
L_H4	256	1024	10	25	1487.4	**1486.6**	1542.6	1534.2	1487.4
L_H5	256	1024	10	50	2279.6	**2279.4**	2311.4	2282.0	2279.6
L_H6	256	1024	10	75	2572.4	**2572.4**	2590.8	2576.4	**2572.4**
L_H7	512	2304	10	25	3119.0	**3118.8**	3240.8	3146.0	3119.0
L_H8	512	2304	10	50	4852.2	**4843.6**	4921.8	4872.4	4852.2
L_H9	512	2304	10	75	5553.4	**5552.4**	5588.6	5563.8	5553.4

Thus, analysing all results of both tables, it is possible to assert that Hybrid-IA is competitive, and very often better than the other compared algorithms, especially with respect to MA algorithm. The comparisons have been done on a total of 60 different graphs, and at different problem dimensions: Hybrid-IA reached the global best solution in 46 instances, and on 25 of these it found solutions better than the best-known values (K^*), whilst MA in only 23. If we focus on the comparison between

Hybrid-IA and MA, we have that MA finds better solutions than our algorithm in only 10 instances, unlike Hybrid-IA that outperforms MA in 35 instances over 60. In the remaining 15 instances both algorithms reach the same solutions that correspond also to the best known.

Table 2.5: Preliminary test results of Hybrid-IA on large instances of toroidal graphs.

	Instance								
Id	x	y	Low	Up	K^*	Hybrid-IA	ITS	XTS	MA
L_T1	10	10	10	25	388.0	**387.8**	388.8	389.0	388.0
L_T2	10	10	10	50	457.6	**457.4**	458.6	457.6	457.6
L_T3	10	10	10	75	504.6	**504.6**	504.8	**504.6**	**504.6**
L_T4	14	14	10	25	748.8	**747.6**	750.8	748.8	749.6
L_T5	14	14	10	50	874.4	**874.2**	875.6	875.4	874.4
L_T6	14	14	10	75	1016.2	**1016.2**	1017.2	1016.4	**1016.2**
L_T7	17	17	10	25	1102.8	1105.2	1110.2	1107.4	**1102.8**
L_T8	17	17	10	50	1304.4	1304.6	1307.6	1306.0	**1304.4**
L_T9	17	17	10	75	1498.6	**1497.6**	1502.4	1499.6	1498.6
L_T10	20	20	10	25	1539.6	1540.2	1548.6	1540.0	**1539.6**
L_T11	20	20	10	50	1795.4	1795.6	1803.4	1797.6	**1795.4**
L_T12	20	20	10	75	2033.0	**2031.8**	2042.6	2033.6	2033.0
L_T13	23	23	10	25	2034.8	2039.0	2043.4	2043.8	**2034.8**
L_T14	23	23	10	50	2406.4	**2406.2**	2412.2	2410.8	2406.4
L_T15	23	23	10	75	2697.2	**2695.4**	2705.4	2704.2	2697.2

2.2.3 Parameter Tuning

As outlined in Chapter 2, the proposed algorithm has four parameters that affect the performances: the population size d, the duplication factor dup, the mutation shape ρ and the maximum age τ. However, starting from previous results [46, 47] on the same benchmark instances and from some preliminary experiments, the crucial parameter that affects the exploration of the neighbourhood of candidate solutions is the mutation shape ρ. This parameter controls the mutation rate α defined in Equation 2.2, that is the percentage of vertices to swap between the solution S and $V \setminus S$ in the mutation operator. For this reason, the parameter tuning was focused only on the mutation shape ρ, trying to find the best value in the closed range $[1.0, 4.0]$, with steps of 0.2. As can be seen in Figure 2.5, using a mutation shape greater than 4.0 means a significant reduction of the number of mutations for a high number of good candidate solutions, limiting the exploration of their neighbourhood and of the search space in general.

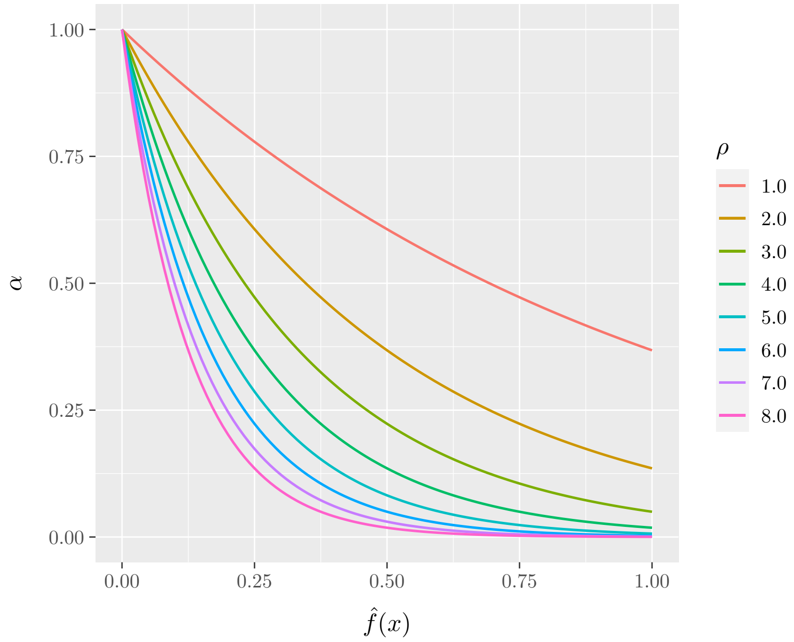


Figure 2.5: Mutation rate α obtained with Equation 2.2 for different values of the mutation shape ρ, with respect to the normalized fitness value.

For the tuning experiments, we selected a subset of the benchmark instances described before. More precisely, we chose for the training set the large random graphs with low and medium density and the large grid graphs, as representations of regular graphs (grid, toroidal and hypercube). This set consists of instances with a number of vertices that goes from 100 to 500, with a graph density that ranges from 16.99% to 3.40% for medium and low-density random graphs, and from 3.64% to 0.72% for grid graphs, and with a weight range of $[10, 50]$ and $[10, 75]$. In this way, we tried to cover all possible features of the benchmark set used. Note that, to save computational time in this phase, we left out of the training set the large random graphs with high density, because, for this kind of instance, Hybrid-IA already reaches good results with a default configuration for this parameter [47]. Finally, for all 16 values of the mutation shape ρ we performed 10 independent runs for each instance of the training set, and for $\{1000, 1250, 1500, 1750, 2000\}$ generations, respectively for $\{100, 200, 300, 400, 500\}$

vertices, as termination criteria.

For the analysis of the tuning experiments, the rank-based as described in [10] has been used. The values of the mutation shape ρ were ordered based on the solutions obtained on the instances of the training set, grouped by size and density of the graph, that is by n and m for random instances and only by n for grid instances. In particular, for each instance the value of ρ that found the best average (over 10 runs), obtain rank 1, the second best average obtain rank 2, and so on. Then, for each n and m an average of the rank for each value of ρ was calculated. The values of ρ with the best average rank for each n and m, are shown in Table 2.6.

Table 2.6: Results of the tuning experiments for the mutation shape parameter on the (a) large random and (b) squared grid graphs.

RANDOM			
n	m	Δ	ρ
100	247	4.99%	1.4
100	841	16.99%	1.4
200	796	4.00%	2.4
200	3184	16.00%	2.8
300	1644	3.67%	2.8
300	7026	15.67%	3.2
400	2793	3.50%	3.2
400	12369	15.50%	3.4
500	4241	3.40%	3.4
500	19211	15.40%	3.6

(a)

GRID			
x	y	Δ	ρ
10	10	3.64%	1.4
14	14	1.90%	2.0
17	17	1.31%	2.4
20	20	0.95%	2.8
23	23	0.72%	3.2

(b)

In general, from these tuning results, we can see that the mutation shape ρ grows increasing the size of the graph, that is, large instances require fewer changes during the mutation phase than smaller problem instances. However, with the same number of vertices, the mutation shape is slightly higher for instances with higher density, and this difference seems to decrease when the size of the graph increases. For reasons of space and readability, in Appendix A.1 are reported all figures that show the outcomes of the tuning experiments.

2.2.4 Results

The aim of this section is to analyse the performance of the proposed algorithm, comparing all the outcomes on this benchmark set with other metaheuristic algorithms available from the literature. For all the experimental results presented in the following sections, the configuration of parameters of Hybrid-IA is as follows: the algorithm maintains a population of candidate solutions of size $d = 100$, with a duplication factor $dup = 2$, and for a maximum of $\tau = 20$ iterations, except for the best solution (see Section 2.1.4). The mutation shape ρ, instead, changes based on the features of the input instance, in accordance with the results of the parameter tuning described in Section 2.2.3 and reported in Table 2.6. Finally, Hybrid-IA stops its execution after a fixed number of generations T_{Max}, that is 500 for small instances, and $\{1000, 1250, 1500, 1750, 2000\}$ for (respectively) large instances. It is important to highlight that the results derived from [28] have been obtained considering as stop condition a certain value of consecutive iterations without improvement of the best solution. This value is determined with a formula related to the size and density of the input instance, and the counter is reset every time an improvement occurs. Moreover, MA, at the end of its execution, restarts the evolutionary cycle penalizing the vertices belonging to the best solution found, in order to explore new regions of the search space. For this reason, the actual number of iterations of MA is not known a priori.

For all algorithms compared, the *Average Gap Value* (AGV) has been calculated, that is the sum of the difference between the value obtained by the algorithm and the value of the best-known solution for an instance. The AGV can be computed as follow:

$$AGV = \frac{1}{N}\sum_{i=1}^{N}(f_i - K^*), \tag{2.5}$$

where f_i is the fitness value of the solution found by the algorithm on instance i, K^* is the best fitness value known and N is the total number of instances for each value of $|V|$. This value can be useful to understand the performance of an algorithm when

Table 2.7: Results of Hybrid-IA on small instances of random graphs.

	Instance								
Id	n	m	Low	Up	Opt	Hybrid-IA	ITS	XTS	MA
S_R1	25	33	10	25	63.8	**63.8**	**63.8**	**63.8**	**63.8**
S_R2	25	33	10	50	99.8	**99.8**	**99.8**	**99.8**	**99.8**
S_R3	25	33	10	75	125.2	**125.2**	**125.2**	**125.2**	**125.2**
S_R4	25	69	10	25	157.6	**157.6**	**157.6**	**157.6**	**157.6**
S_R5	25	69	10	50	272.2	**272.2**	**272.2**	**272.2**	**272.2**
S_R6	25	69	10	75	409.4	**409.4**	**409.4**	**409.4**	**409.4**
S_R7	25	204	10	25	273.4	**273.4**	**273.4**	**273.4**	**273.4**
S_R8	25	204	10	50	507.0	**507.0**	**507.0**	**507.0**	**507.0**
S_R9	25	204	10	75	785.8	**785.8**	**785.8**	**785.8**	**785.8**
S_R10	50	85	10	25	174.6	**174.6**	175.4	176.0	**174.6**
S_R11	50	85	10	50	280.8	**280.8**	**280.8**	281.6	**280.8**
S_R12	50	85	10	75	348.0	**348.0**	**348.0**	349.2	**348.0**
S_R13	50	232	10	25	386.2	**386.2**	389.4	386.8	**386.2**
S_R14	50	232	10	50	708.6	**708.6**	**708.6**	**708.6**	**708.6**
S_R15	50	232	10	75	951.6	**951.6**	**951.6**	**951.6**	**951.6**
S_R16	50	784	10	25	602.0	**602.0**	602.2	**602.0**	**602.0**
S_R17	50	784	10	50	1171.8	**1171.8**	1172.2	1172.0	**1171.8**
S_R18	50	784	10	75	1648.8	**1648.8**	1649.4	**1648.8**	**1648.8**
S_R19	75	157	10	25	318.2	**318.2**	321.0	320.0	**318.2**
S_R20	75	157	10	50	521.6	**521.6**	526.2	525.0	522.6
S_R21	75	157	10	75	751.0	**751.0**	757.2	754.2	**751.0**
S_R22	75	490	10	25	635.8	**635.8**	638.6	**635.8**	**635.8**
S_R23	75	490	10	50	1226.6	**1226.6**	1230.6	1228.6	1227.6
S_R24	75	490	10	75	1789.4	**1789.4**	1793.6	**1789.4**	**1789.4**
S_R25	75	1739	10	25	889.8	**889.8**	891.0	**889.8**	**889.8**
S_R26	75	1739	10	50	1664.2	**1664.2**	1664.8	**1664.2**	**1664.2**
S_R27	75	1739	10	75	2452.2	**2452.2**	2452.8	**2452.2**	**2452.2**

the size increases for a specific class of instances.

Small Instances

In the following tables, the results of the four algorithms on all small instances are reported. The first five columns represent the instance, with its name (*Id*), the size of the graph (n and m for random and hypercube graphs, x and y for grid and toroidal graphs), the lower and upper bound of the weight range (*Low* and *Up*), and the optimal solution (*Opt*). The last four columns report the fitness values of the best solutions found by the compared algorithms. These fitness values are the average value obtained on 5 problem instances, as described in the first part of Section 2.2.4.

Table 2.7 shows the results of all algorithms on small random instances. From this table, it is possible to note how Hybrid-IA obtains a perfect score reaching the optimal solution for all instances. A similar result is obtained by MA, which reaches the optimal solutions for 25 instances out of 27. MA fails to reach the optimum on two instances with 75 vertices, `S_R20` and `S_R23`, both with a weight range of $[10, 50]$ but with different density. XTS reaches the optimal solution for 18 instances out of 27, failing mainly on instances with low density, both with 50 and 75 vertices. Finally, ITS has the worst results, reaching the optimal solution only on 13 instances out of 27. This algorithm never reaches the optimal solution on instances with a weight range $[10, 25]$, when the number of vertices is 50 and 75. Moreover, its performance also deteriorates as the size of the instance increases, in particular on low and medium density.

Table 2.8: Results of Hybrid-IA on small instances of squared grid graphs.

	Instance								
Id	x	y	Low	Up	Opt	Hybrid-IA	ITS	XTS	MA
S_SG1	5	5	10	25	114.0	**114.0**	**114.0**	**114.0**	**114.0**
S_SG2	5	5	10	50	199.8	**199.8**	**199.8**	**199.8**	**199.8**
S_SG3	5	5	10	75	312.4	**312.4**	312.6	**312.4**	**312.4**
S_SG4	7	7	10	25	252.0	**252.0**	252.4	**252.0**	**252.0**
S_SG5	7	7	10	50	437.6	**437.6**	439.8	**437.6**	**437.6**
S_SG6	7	7	10	75	713.6	**713.6**	718.4	717.4	**713.6**
S_SG7	9	9	10	25	442.2	**442.2**	444.2	442.8	**442.2**
S_SG8	9	9	10	50	752.2	**752.2**	754.6	753.0	**752.2**
S_SG9	9	9	10	75	1134.4	**1134.4**	1138.0	**1134.4**	**1134.4**

In Tables 2.8–2.11 the results on small regular instances are reported, that is squared and not squared grid, toroidal and hypercube graphs respectively. Also, in this case, Hybrid-IA obtained a perfect score, reaching the optimal solution for all 36 instances. As for random instances, MA has similar performances (33 out of 36), failing only on three instances with (about) 75 vertices and weight range $[10, 25]$, `S_NG7`, `S_T7` and `S_H7`. XTS reaches the optimal solution on 24 instances out of 36, failing mainly on instances with 75 vertices and weight range $[10, 25]$ and $[10, 75]$. Also for this kind of

Table 2.9: Results of Hybrid-IA on small instances of not squared grid graphs.

	Instance								
Id	x	y	Low	Up	Opt	Hybrid-IA	ITS	XTS	MA
S_NG1	8	3	10	25	96.8	**96.8**	**96.8**	**96.8**	**96.8**
S_NG2	8	3	10	50	157.4	**157.4**	**157.4**	**157.4**	**157.4**
S_NG3	8	3	10	75	220.0	**220.0**	**220.0**	**220.0**	**220.0**
S_NG4	9	6	10	25	295.6	**295.6**	295.8	295.8	**295.6**
S_NG5	9	6	10	50	488.6	**488.6**	489.4	**488.6**	**488.6**
S_NG6	9	6	10	75	755.0	**755.0**	**755.0**	755.2	**755.0**
S_NG7	12	6	10	25	398.2	**398.2**	399.8	398.8	398.4
S_NG8	12	6	10	50	671.8	**671.8**	673.4	**671.8**	**671.8**
S_NG9	12	6	10	75	1015.2	**1015.2**	1017.4	1015.4	**1015.2**

Table 2.10: Results of Hybrid-IA on small instances of toroidal graphs.

	Instance								
Id	x	y	Low	Up	Opt	Hybrid-IA	ITS	XTS	MA
S_T1	5	5	10	25	101.4	**101.4**	**101.4**	**101.4**	**101.4**
S_T2	5	5	10	50	124.4	**124.4**	**124.4**	**124.4**	**124.4**
S_T3	5	5	10	75	157.8	**157.8**	**157.8**	158.8	**157.8**
S_T4	7	7	10	25	195.4	**195.4**	197.4	**195.4**	**195.4**
S_T5	7	7	10	50	234.2	**234.2**	**234.2**	**234.2**	**234.2**
S_T6	7	7	10	75	269.6	**269.6**	**269.6**	**269.6**	**269.6**
S_T7	9	9	10	25	309.6	**309.6**	310.4	309.8	309.8
S_T8	9	9	10	50	369.6	**369.6**	370.0	**369.6**	**369.6**
S_T9	9	9	10	75	431.8	**431.8**	432.2	432.2	**431.8**

instance ITS has the worst performances (17 out of 36), failing to reach the optimal solution when the size of the instances increases, except in rare occasions.

These considerations are reflected in Figures 2.6–2.10, which show the AGV (see Equation 2.5) of the four algorithms. The curves of Hybrid-IA and MA have the same trend on small instances, with a slight difference in favour of Hybrid-IA on instances with 75 vertices, due to a higher number of optimal solutions found. On the other hand, the curves of XTS and ITS tend to grow slightly on instances with 50 vertices and more on instances with 75 vertices for all classes, with the exception of hypercube graphs, which seem to be the easiest instances to solve.

Table 2.11: Results of HYBRID-IA on small instances of hypercube graphs.

	INSTANCE								
Id	n	m	Low	Up	Opt	HYBRID-IA	ITS	XTS	MA
S_H1	16	32	10	25	72.2	**72.2**	**72.2**	**72.2**	**72.2**
S_H2	16	32	10	50	93.8	**93.8**	**93.8**	**93.8**	**93.8**
S_H3	16	32	10	75	97.4	**97.4**	**97.4**	**97.4**	**97.4**
S_H4	32	80	10	25	170.0	**170.0**	**170.0**	**170.0**	**170.0**
S_H5	32	80	10	50	240.6	**240.6**	241.0	**240.6**	**240.6**
S_H6	32	80	10	75	277.6	**277.6**	**277.6**	**277.6**	**277.6**
S_H7	64	192	10	25	353.4	**353.4**	354.6	353.8	353.8
S_H8	64	192	10	50	475.6	**475.6**	476.0	**475.6**	**475.6**
S_H9	64	192	10	75	503.8	**503.8**	**503.8**	504.8	**503.8**

Large Instances

In this section the experimental results on large instances are reported, in order to test the effectiveness of HYBRID-IA on complex problem instances. Table 2.12 reports the outcomes on large random graphs. The columns of the table are the same as Table 2.7, except for the column *Opt*, replaced by the column K^* that represents the best solution known, that is the best solution among ITS, XTS and MA. For this kind of graph HYBRID-IA reaches the best solution in 29 out of 45 instances, improving the best-known solution K^* in 11 instances. In this statistic, MA and XTS have similar performance, achieving the best solution in 21 and 20 out of 45 instances respectively. ITS obtains the best solution in only 2 instances, confirming the low performance of small instances.

The proposed algorithm reaches the best solution in all instances with $n = 100$, improving K^* in 3 instances. The same behaviour can be observed on graphs with $n = 200$, where it fails only on two occasions, `L_R11` and `L_R13`, with a gap from MA of 0.4 and 1.4 respectively. When the size of the graphs increases HYBRID-IA does not reach the best solutions, especially for low and medium density instances. However, the gap from the best solutions K^* is quite small for almost all instances with n equals to 300, 400 and 500, expect for `L_R29`, `L_R37` and `L_R39`, where the gap reaches a peak of 17.0. On the other hand, HYBRID-IA finds the best solution for `L_R24`, `L_R30`,

Table 2.12: Results of Hybrid-IA on large instances of random graphs.

	Instance								
Id	n	m	Low	Up	K^*	Hybrid-IA	ITS	XTS	MA
L_R1	100	247	10	25	498.4	**498.4**	501.4	500.8	**498.4**
L_R2	100	247	10	50	836.8	**835.0**	845.8	840.0	836.8
L_R3	100	247	10	75	1207.6	**1207.2**	1223.8	1208.0	1207.6
L_R4	100	841	10	25	826.8	**826.8**	828.2	**826.8**	**826.8**
L_R5	100	841	10	50	1724.4	**1724.4**	1729.6	1724.6	**1724.4**
L_R6	100	841	10	75	2420.6	**2420.4**	2425.6	2420.6	2420.6
L_R7	100	3069	10	25	1134.0	**1134.0**	**1134.0**	**1134.0**	**1134.0**
L_R8	100	3069	10	50	2179.0	**2179.0**	**2179.0**	**2179.0**	**2179.0**
L_R9	100	3069	10	75	3228.6	**3228.6**	3228.8	3228.8	**3228.6**
L_R10	200	796	10	25	1468.2	**1465.4**	1488.4	1468.8	1468.2
L_R11	200	796	10	50	2399.0	2399.4	2442.6	2414.4	**2399.0**
L_R12	200	796	10	75	3089.6	**3088.0**	3157.0	3099.6	3089.6
L_R13	200	3184	10	25	1986.2	1987.6	2003.6	1986.8	**1986.2**
L_R14	200	3184	10	50	3650.6	**3649.2**	3683.6	3650.6	3651.8
L_R15	200	3184	10	75	5135.8	**5133.8**	5158.6	5137.2	5135.8
L_R16	200	12139	10	25	2447.8	**2447.8**	2450.0	2448.4	**2447.8**
L_R17	200	12139	10	50	4148.6	**4148.6**	4149.4	**4148.6**	4149.0
L_R18	200	12139	10	75	5528.4	**5528.4**	5531.4	**5528.4**	**5528.4**
L_R19	300	1644	10	25	2045.4	2046.2	2072.6	**2045.4**	2048.0
L_R20	300	1644	10	50	4175.4	4177.8	4239.4	4195.2	**4175.4**
L_R21	300	1644	10	75	6065.2	6067.4	6154.4	6102.8	**6065.2**
L_R22	300	7026	10	25	3203.0	3204.0	3231.0	**3203.0**	3207.6
L_R23	300	7026	10	50	6211.0	6213.8	6261.4	**6211.0**	6217.2
L_R24	300	7026	10	75	8585.4	**8573.2**	8660.6	8585.4	8613.2
L_R25	300	27209	10	25	3726.6	**3726.6**	3729.2	**3726.6**	**3726.6**
L_R26	300	27209	10	50	5734.8	**5734.8**	5738.0	**5734.8**	**5734.8**
L_R27	300	27209	10	75	10467.0	**10467.0**	10469.6	**10467.0**	**10467.0**
L_R28	400	2793	10	25	2989.6	2990.4	3015.2	2991.0	**2989.6**
L_R29	400	2793	10	50	6410.0	6420.2	6528.0	6435.8	**6410.0**
L_R30	400	2793	10	75	8597.2	**8583.4**	8730.0	8637.0	8597.2
L_R31	400	12369	10	25	4428.8	4430.8	4451.8	**4428.8**	4437.4
L_R32	400	12369	10	50	6785.8	6789.4	6837.4	**6785.8**	6800.6
L_R33	400	12369	10	75	10599.4	**10591.2**	10661.8	10599.4	10601.0
L_R34	400	48279	10	25	5060.4	**5060.4**	5060.8	**5060.4**	5060.6
L_R35	400	48279	10	50	7106.8	**7106.8**	7109.2	**7106.8**	7108.0
L_R36	400	48279	10	75	15103.2	**15103.2**	15114.6	**15103.2**	15117.8
L_R37	500	4241	10	25	4056.4	4067.0	4102.8	4063.0	**4056.4**
L_R38	500	4241	10	50	7170.4	7172.8	7285.0	7204.6	**7170.4**
L_R39	500	4241	10	75	11135.6	11152.4	11285.6	11179.6	**11135.6**
L_R40	500	19211	10	25	5724.2	5726.4	5745.8	**5724.2**	5741.4
L_R41	500	19211	10	50	7677.8	7678.8	7725.0	**7677.8**	7678.2
L_R42	500	19211	10	75	14124.8	**14121.4**	14167.8	14124.8	14164.8
L_R43	500	75349	10	25	6361.6	**6361.2**	6366.4	6362.0	6361.6
L_R44	500	75349	10	50	8668.4	**8668.4**	8671.2	**8668.4**	**8668.4**
L_R45	500	75349	10	75	16932.4	**16932.4**	16939.2	**16932.4**	16933.8

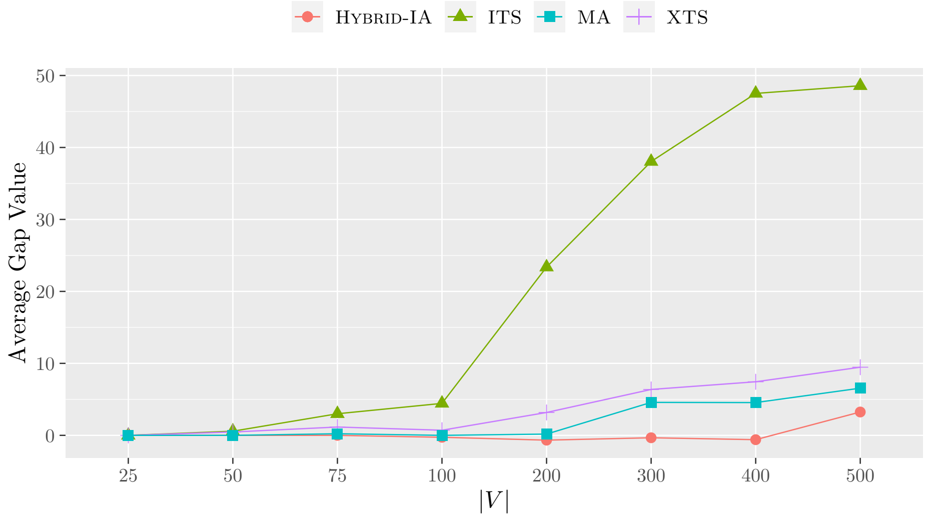


Figure 2.6: The average gap value of Hybrid-IA on the small and large random graphs.

L_R33 and L_R42, all with a weight range $[10, 75]$, greatly improving K^* for the first three instances.

Inspecting the plot in Figure 2.6, Hybrid-IA obtains a negative AGV for the instances with n that ranges from 100 to 400, while less than 5.0 points from K^* for instances with $n = 500$. However, the AGV of Hybrid-IA is always lower than that obtained by the other three algorithms. Although MA and XTS have achieved a similar number of best solutions for these instances, on average MA obtains a slightly better AGV than XTS.

In Tables 2.13 and 2.14 the results on toroidal and hypercube large graphs are reported. For this kind of instance, Hybrid-IA obtains the best performance, where the algorithm is able to find the best solution in all toroidal instances (15 out of 15), and in 6 out of 9 hypercube problem instances. As for the random instances, MA and XTS have similar statistics also for this kind of graph, with MA that obtains the best solution in 2 out of 15 toroidal instances and in 3 out of 9 hypercube instances, while XTS obtains the best solution in 1 out of 15 toroidal instances and in 2 out of 9 hypercube instances. Moreover, no best solutions were found by ITS for both kinds of graphs. For the toroidal instances, the AGV in Figure 2.7 of Hybrid-IA is quite small for instances with 100 vertices and grows in a negative way as the size of graph

Table 2.13: Results of Hybrid-IA on large instances of toroidal graphs.

	Instance								
Id	x	y	Low	Up	K^*	Hybrid-IA	ITS	XTS	MA
L_T1	10	10	10	25	388.0	**387.8**	388.8	389.0	388.0
L_T2	10	10	10	50	457.6	**457.4**	458.6	457.6	457.6
L_T3	10	10	10	75	504.6	**504.6**	504.8	**504.6**	**504.6**
L_T4	14	14	10	25	748.8	**748.0**	750.8	748.8	749.6
L_T5	14	14	10	50	874.4	**874.2**	875.6	875.4	874.4
L_T6	14	14	10	75	1016.2	**1016.2**	1017.2	1016.4	**1016.2**
L_T7	17	17	10	25	1102.8	**1101.4**	1110.2	1107.4	1102.8
L_T8	17	17	10	50	1304.4	**1303.4**	1307.6	1306.0	1304.4
L_T9	17	17	10	75	1498.6	**1497.8**	1502.4	1499.6	1498.6
L_T10	20	20	10	25	1539.6	**1535.8**	1548.6	1540.0	1539.6
L_T11	20	20	10	50	1795.4	**1794.2**	1803.4	1797.6	1795.4
L_T12	20	20	10	75	2033.0	**2031.6**	2042.6	2033.6	2033.0
L_T13	23	23	10	25	2034.8	**2033.8**	2043.4	2043.8	2034.8
L_T14	23	23	10	50	2406.4	**2401.2**	2412.2	2410.8	2406.4
L_T15	23	23	10	75	2697.2	**2694.6**	2705.4	2704.2	2697.2

Table 2.14: Results of Hybrid-IA on large instances of hypercube graphs.

	Instance								
Id	n	m	Low	Up	K^*	Hybrid-IA	ITS	XTS	MA
L_H1	128	448	10	25	731.8	**731.4**	740.0	742.0	731.8
L_H2	128	448	10	50	1066.8	**1066.8**	1071.0	**1066.8**	1067.2
L_H3	128	448	10	75	1161.6	**1161.6**	1163.6	**1161.6**	1162.4
L_H4	256	1024	10	25	1487.4	**1486.6**	1542.6	1534.2	1487.4
L_H5	256	1024	10	50	2279.6	2279.8	2311.4	2282.0	**2279.6**
L_H6	256	1024	10	75	2572.4	2572.8	2590.8	2576.4	**2572.4**
L_H7	512	2304	10	25	3119.0	3119.2	3240.8	3146.0	**3119.0**
L_H8	512	2304	10	50	4852.2	**4844.0**	4921.8	4872.4	4852.2
L_H9	512	2304	10	75	5553.4	**5545.2**	5588.6	5563.8	5553.4

increases, till to reach the maximum gap for instances with 529 vertices. For hypercube graphs, Hybrid-IA fails to find the best solution in only three instances, `L_H5`, `L_H6` and `L_H7`, with a gap from the best solution very small, 0.4 at most from MA. As can be seen from the plot in Figure 2.8, the gap between Hybrid-IA and MA is minimal for instances with $n = 128$ and $n = 256$. On the other hand, the gap of Hybrid-IA from MA increases for instances with $n = 512$, where the improvement of K^* obtained for `L_H8` and `L_H9` is considerable.

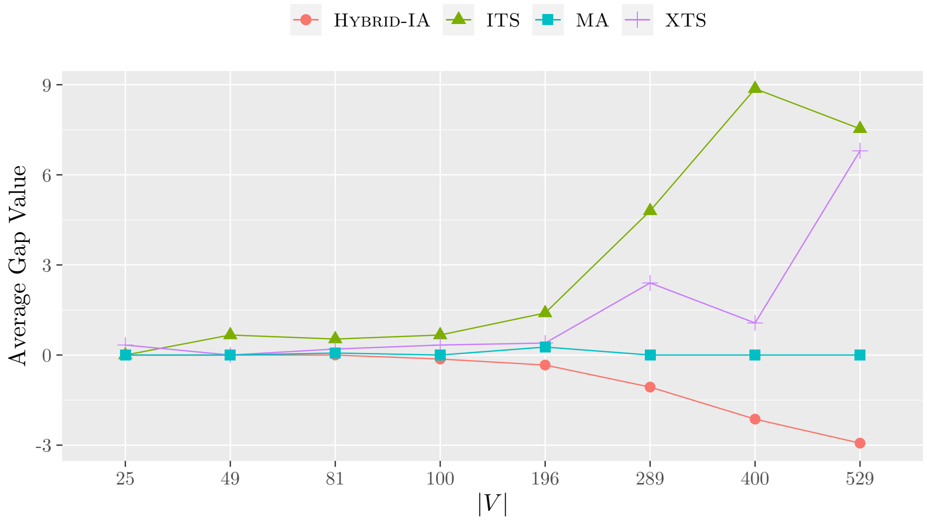


Figure 2.7: The average gap value of Hybrid-IA on the small and large toroidal graphs.

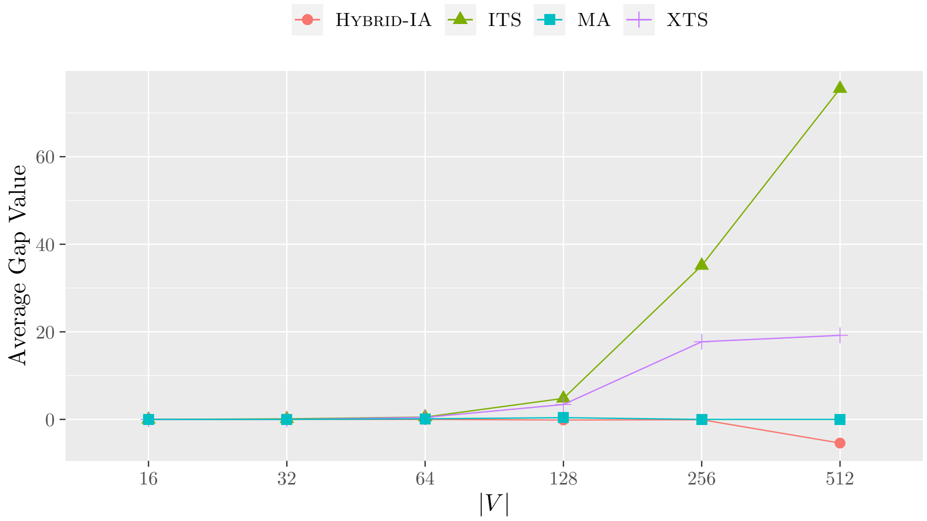


Figure 2.8: The average gap value of Hybrid-IA on the small and large hypercube graphs.

Finally, in Tables 2.15 and 2.16 are reported the results of all algorithms on squared and not squared grid graphs, while in Figures 2.9 and 2.10 the AGV for the same set of instances. For this kind of graph, Hybrid-IA has slightly lower performance obtaining only the best solution in 14 out of 30 grid instances, against the 12 best solutions found out of 30 for MA. XTS has obtained the best solution in 6 out of 30, while ITS obtains the best solution in only 1 out of 30 grid instances. For grid graphs with 100 vertices, Hybrid-IA obtains the best solution for all 6 instances, improving

Table 2.15: Results of Hybrid-IA on large instances of squared grid graphs.

	Instance								
Id	x	y	Low	Up	K^*	Hybrid-IA	ITS	XTS	MA
L_SG1	10	10	10	25	566.8	**566.2**	570.6	566.8	567.0
L_SG2	10	10	10	50	947.0	**945.6**	948.8	949.4	947.0
L_SG3	10	10	10	75	1557.8	**1556.0**	1566.0	1565.2	1557.8
L_SG4	14	14	10	25	1206.4	1207.4	1209.4	1207.6	**1206.4**
L_SG5	14	14	10	50	2007.8	**2003.0**	2008.6	2010.2	2007.8
L_SG6	14	14	10	75	3399.2	3403.0	3401.2	3406.0	**3399.2**
L_SG7	17	17	10	25	1829.4	1835.2	1834.2	1830.4	**1829.4**
L_SG8	17	17	10	50	3051.4	**3051.2**	3070.6	3062.8	3051.4
L_SG9	17	17	10	75	5071.2	5087.0	5089.8	**5071.2**	5072.2
L_SG10	20	20	10	25	2602.2	2622.0	2619.8	2607.8	**2602.2**
L_SG11	20	20	10	50	4299.8	**4290.8**	4321.2	4306.6	4299.8
L_SG12	20	20	10	75	7240.4	7277.6	7272.6	**7240.4**	7252.2
L_SG13	23	23	10	25	3453.6	3473.4	3462.8	3460.6	**3453.6**
L_SG14	23	23	10	50	5831.4	**5818.8**	5865.4	5837.4	5831.4
L_SG15	23	23	10	75	9681.0	9724.4	9723.4	9718.6	**9681.0**

the value of K^* for 5 instances. For instances with 200 vertices, the algorithm fails to find the best solution for the weight ranges $[10, 25]$ and $[10, 75]$, while maintaining a small gap from MA. On the other hand, it finds the best solution for the range $[10, 50]$ (L_SG5 and L_NG5), resulting in AGV similar to that obtained by MA and lower than XTS. When the size of the graphs increases, the algorithm exhibits the same behaviour. For instances with n equals to 300, 400 and 500, Hybrid-IA fails to obtain good solutions for instances with weight range $[10, 25]$ and $[10, 75]$, while it is able to find better (L_SG8, L_SG11, L_SG14, L_NG8 and L_NG14), or at least similar (L_NG11), solutions for the range $[10, 50]$. However, these best solutions found do not significantly reduce the AGV value of Hybrid-IA, which remains higher than MA and XTS on both grid graphs, except on not squared grid with 493 vertices, mainly due to the best solutions obtained (L_NG14 and L_NG15).

In the overall, Hybrid-IA has been tested on a benchmark set with a total of 162 instances, with different topologies, sizes and densities. From the analysis of the results reported in this section, we can see how Hybrid-IA achieves excellent performance, both in terms of the best solution found and AGV, on random, toroidal and hypercube

Table 2.16: Results of Hybrid-IA on large instances of not squared grid graphs.

	Instance								
Id	x	y	Low	Up	K^*	Hybrid-IA	ITS	XTS	MA
L_NG1	13	7	10	25	512.2	**511.6**	513.0	513.0	512.2
L_NG2	13	7	10	50	803.4	**803.4**	**803.4**	**803.4**	**803.4**
L_NG3	13	7	10	75	1382.8	**1381.6**	1390.8	1386.0	1382.8
L_NG4	18	11	10	25	1204.6	1205.4	1208.0	1206.4	**1204.6**
L_NG5	18	11	10	50	2041.2	**2036.6**	2049.8	2047.6	2041.2
L_NG6	18	11	10	75	3417.4	3421.2	3431.0	3422.2	**3417.4**
L_NG7	23	13	10	25	1923.8	1934.8	1930.6	**1923.8**	1925.2
L_NG8	23	13	10	50	3178.2	**3171.6**	3194.8	3187.6	3178.2
L_NG9	23	13	10	75	5258.0	5273.6	5286.6	5286.0	**5258.0**
L_NG10	26	15	10	25	2522.6	2544.6	2532.8	**2522.6**	2529.4
L_NG11	26	15	10	50	4144.2	4146.0	4164.8	4148.4	**4144.2**
L_NG12	26	15	10	75	7031.4	7070.8	7063.4	**7031.4**	7035.8
L_NG13	29	17	10	25	3255.0	3265.0	3270.0	3257.0	**3255.0**
L_NG14	29	17	10	50	5410.8	**5398.0**	5430.4	5422.6	5410.8
L_NG15	29	17	10	75	8953.0	**8949.0**	8993.2	8985.6	8953.0

graphs, while it has slightly lower performance on graphs with a particular topology, such as grid graphs, that can be classified as the hardest ones. Among these instances, the proposed algorithm was able to find the best solution in 127 of them, improving the best-known values K^* in 40 of them, while MA and XTS found the best solution in 96 and 71 instances respectively.

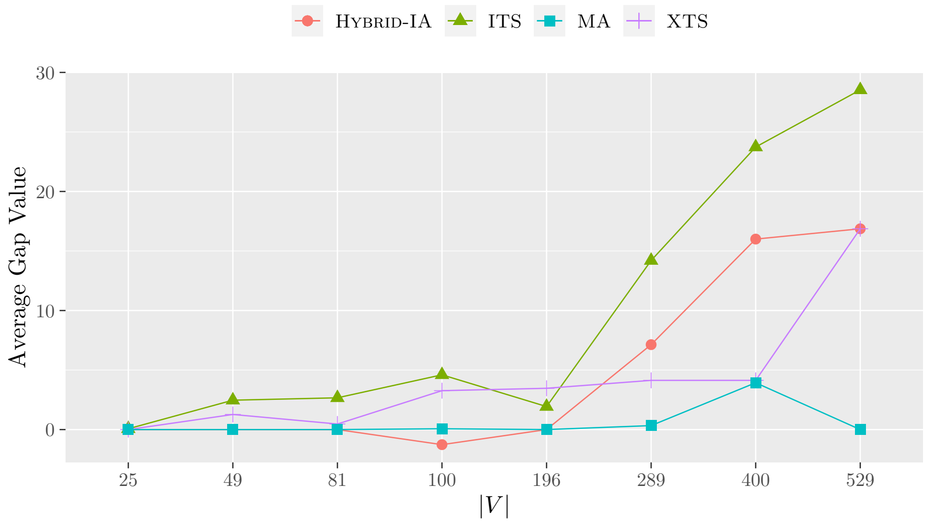


Figure 2.9: The average gap value of Hybrid-IA on the small and large squared grid graphs.

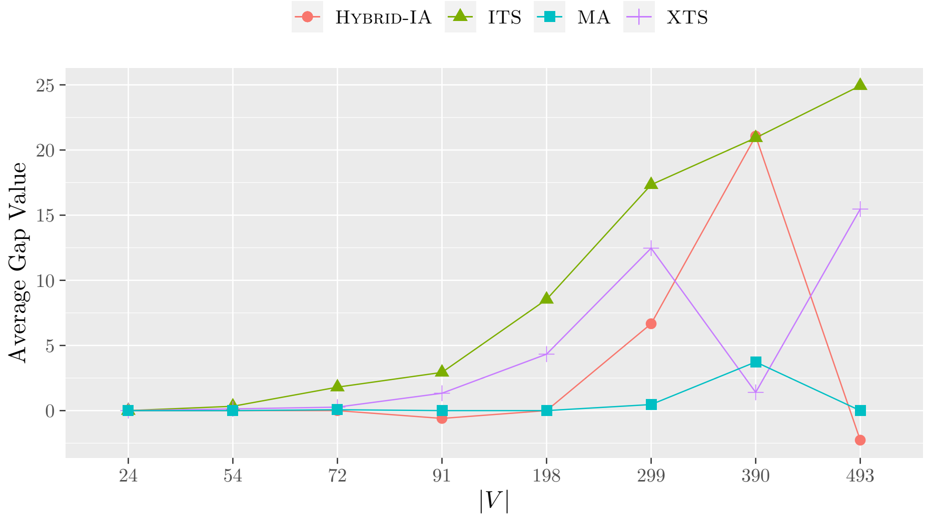


Figure 2.10: The average gap value of Hybrid-IA on the small and large not squared grid graphs.

2.3 Conclusions

In this section, an immune metaheuristic is presented and designed to tackle and solve one of the most challenging combinatorial optimization problems, such as the weighted variant of the feedback vertex set that finds applicability in many real-world tasks. The proposed algorithm, simply called Hybrid-IA, is based on the clonal selection metaphor (proliferation and differentiation of the cells), and it takes advantage of three main immune operators that allow the algorithm to make a faithful exploration of the search space, avoid to get trapped into local optima and exploit all information learned as better as possible. Also, a Local Search has been developed in Hybrid-IA whose goal is to deterministically refine all solutions produced instead of the stochasticity. Several experiments have been performed for evaluating both the efficacy of the developed local search and the robustness and reliability of Hybrid-IA. A comparison with three other different metaheuristics has been performed, which represent nowadays the state-of-the-art on this problem: *Iterated Tabu Search* (ITS), *eXploring Tabu Search* (XTS) and *Memetic Algorithm* (MA).

In the overall, Hybrid-IA has been run and tested on a total of 162 different graph instances (in topology, dimension, and vertex weights), on which it was able in finding the best solution in 127 of them. Moreover, among these 127 instances, Hybrid-IA has improved the best-known values in 40 of them. An analysis of the convergence and learning process has been also performed in order to understand the dynamics features of Hybrid-IA, as well as how well the local search affects the performances of the presented immune metaheuristic. From these analyses emerges quite clear the goodness of the designed local search, which is able to considerably improve the performances of the algorithm, and allows it to learn much more information. Finally, inspecting all outcomes, from the dynamic behaviours to experimental results, it is possible to assert that Hybrid-IA is a robust, efficient and reliable optimization algorithm.

3

Hybrid Immunological Algorithm with Reinforcement Learning

In recent years, part of the scientific community has focused its research on combining learning techniques and heuristic/metaheuristic algorithms [147]. One of these learning patterns is Reinforcement Learning (RL), which aims to learn the best move among those available in the current state, continuously interacting with an unknown environment. Unlike supervised learning techniques, reinforcement learning adjusts its future actions based on the obtained feedback from the environment in which it operates [72].

Integrating RL at metaheuristic level can be useful to learn properties of good initial solutions or an evaluation function in order to guide a metaheuristic toward high-quality solutions. For example, in [157] the opposition-based learning is combined with a memetic algorithm for solving the maximum diversity problem, while in [144] the authors propose a multi-objective memetic algorithm with clustering technique and statistical learning for the discrete permutation flowshop scheduling problem. RL techniques have been also used with Iterated Local Search [11, 145] and Tabu Search methods [94, 95]. Recently, in [158, 156] a general-purpose local search framework with RL for solving grouping problems have been introduced, which employs a dynamically updated probability matrix to generate starting solutions for a local search procedure.

In this work, we are interested in employing RL techniques to process information collected from the search process of the population-based iterated greedy (PBIG) algorithms [21], with the purpose of improving the performance of evolutionary algorithms. The population-based iterated greedy approach belongs to the class of iterated greedy (IG) algorithms, which works on populations of solutions, rather than being restricted to a single incumbent solution. At each iteration, the PBIG algorithm first partially destroys a population of solutions and then a greedy heuristic is used to construct a complete solution from the partially destroyed solutions. Starting the solution construction from partial solutions has two important advantages. The first one is that the solution construction process is substantially faster, and the second one is the exploitation of good parts of solutions [78, 120]. This approach has proved their potential for effectively solving a wide range of difficult optimization problems [21, 20, 22].

3.1 The Proposed Method

In this section, we outline the proposed framework, an evolutionary algorithm based on a greedy population combined with reinforcement learning techniques. The proposed algorithm, called GREEDY-IA, is based on the immunological algorithm described in Chapter 2, where a population of solutions, starting from an initial population generated in a quasi-greedy way, is repeatedly improved by alternating a phase of destruction and reconstruction until a termination criterion is satisfied. The reinforcement learning mechanism exploits the information discovered during these two phases with the aim to guide the search towards promising regions of the solution space.

In general, construction methods are based on heuristics designed to extend partial or empty solutions. These partial solutions may be extended by adding a solution component chosen from a finite set of solution components. In this framework, each solution component is associated with a probability that determines its chances of being selected to be part of the solution. These probabilities are stored in a *probability matrix* and are used during the construction and reconstruction phases. This matrix evolves at each iteration by means of reinforcement learning, depending on the contribution that components give to the solutions.

The Weighted Feedback Vertex Set problem was used as case study to demonstrate the application of the proposed framework, but can easily be applied to any grouping problems for which a constructive heuristic is known. The algorithm works as follows. First, the solutions of the initial population are generated by the function GreedyWFVS (or a generic randomized greedy heuristic), where all solution components have the same probability to be part of the solution. Then, each solution is duplicated and partially destroyed by the mutation operator, resulting in a partial solution. Starting from this partial solution, a new complete solution is constructed using the procedure GreedyWFVS, updating the probability matrix if the new solution improves the parent one. Finally, the evolutionary cycle ends by applying the selection operator, as described in Section 2.1.5. A high-level description of our algorithm is given in Algorithm 3.1.

In the following sections, the three main phases of the algorithm are described in more detail, that is the greedy construction algorithm, the destruction phase and the probability updating procedure.

3.1.1 Greedy Construction Algorithm

The GreedyWFVS procedure takes a partial or empty solution S as input and constructs a complete solution, that is a feedback vertex set for the graph $G(V, E)$. At each construction step, this algorithm adds a solution component to the partial solution S from a finite set of solution components C. In the case of the Feedback Vertex Set problem, $C \subset V \setminus S$ represents the set of vertices involved in one or more cycles,

Algorithm 3.1: Pseudo-code of Greedy-IA.

```
procedure Greedy-IA($n$, $d$, $l$ $\rho$, $\alpha$, $\beta$, $\gamma$, $\delta$, $p_0$)
    $t \leftarrow 0$
    $P \leftarrow$ InitializeProbabilityVector()
    $\mathcal{P}^{(t)} \leftarrow$ GenerateInitialPopulation($n, P, l$)
    ComputeFitness($\mathcal{P}^{(t)}$)
    while $\neg$StopCriterion do
        $\mathcal{P}^{(c)} \leftarrow$ Cloning($\mathcal{P}^{(t)}, d$)
        $\mathcal{P}^{(h)} \leftarrow$ Hypermutation($\mathcal{P}^{(c)}, \rho$)
        $\mathcal{P}'^{(t)} \leftarrow$ GreedyWFVS($\mathcal{P}^{(h)}, P, l$)
        ComputeFitness($\mathcal{P}'^{(t)}$)
        UpdateProbabilityVector($P, \mathcal{P}^{(t)}, \mathcal{P}'^{(t)}, \alpha, \beta, \gamma$)
        SmoothProbabilityVector($P, p_0, \delta$)
        $\mathcal{P}^{(t+1)} \leftarrow$ Selection($\mathcal{P}^{(t)}, \mathcal{P}'^{(t)}$)
        $t \leftarrow t + 1$;
    end while
end procedure
```

therefore with a degree greater than 1, in the residual graph induced by the partial solution S, as described in Section 1.1. Note that when S is a complete solution it holds that $C = \emptyset$. On the other hand, at the start of the algorithm, when S is empty, it holds that $C = V$.

At each step, the next solution component to be added to S is selected in the following way. First, all solution components in C are rated according to the following greedy function defined as:

$$c(v) = w(v)/d_S(v)^{\tau}, \tag{3.1}$$

where τ is a parameter that weighs the degree of node $v \in C$ with respect to the reduced subgraph $G[C]$ induces by the partial solution S. To diversify the search the value of parameter τ is randomly selected from $\{1, 2\}$ at the beginning of the construction procedure. Then the algorithm selects a solution component v between the l best candidates (l is a user-defined parameter), that is between those components with lowest cost values, using the probability matrix P. In the case of the Feedback Vertex Set problem, the probability matrix P can be represented as a vector, because we have only two classes, in which $p_v = \Pr(v \in S)$. Obviously, the probability that a component v does not belong to solution S is given by $1 - p_v$. In particular, to decide which of the l solution components to add to the solution S, the algorithm generates a random value uniformly distributed over $[0, 1]$. If this random number is lower than $p_{v_{best}}$, then v_{best} is chosen, otherwise, v is chosen uniformly at random between the first l best candidates. Once a vertex is removed, the set of solution components C is updated by recursively removing all the components that could no longer be part of the solution, that is, those vertices with a degree less than 2. The solution construction process stops when $C = \emptyset$. After a complete solution is constructed, a local refinement procedure tries to reduce the weight of a solution by replacing one of its vertices with a set of vertices from the residual graph (see Algorithm 2.3).

Finally, any redundant vertices are removed from the complete solution S, maintaining the feasibility of the solution and minimizing the value of the objective function (see Algorithm 2.2). Also in this case, the vertices of the complete solution S are rated using the reverse greedy function in Equation 3.1 and assessed from the vertex with the highest cost to the vertex with the lowest cost. The value of the parameter τ is the same chosen at the beginning of the construction procedure. In Algorithm 3.2 is shown the pseudocode of GreedyWFVS.

3.1.2 Destruction Phase

In the destruction phase, any complete S solution belonging to the offspring population, generated by the cloning operator as described in Section 2.1.2, is partially destroyed by removing a subset of vertices from solution S. The number of vertices to remove from the solution S is calculated as follows:

$$M = \lfloor (r \cdot |S|) + 1 \rfloor, \tag{3.2}$$

with r that represents the *destruction rate*. The destruction rate r depends proportionally on the quality of the solution and is obtained as:

$$r = e^{-\hat{f}(S)}/\rho, \tag{3.3}$$

Algorithm 3.2: Greedy procedure that creates a feedback vertex set for the graph $G(V, E, w)$ starting from a partial solution S and exploiting the probability vector P.

```
procedure GreedyWFVS($G(V, E, w)$, $S$, $P$, $l$)
    $C \leftarrow V \setminus S$
    while $\exists v \in C : d_{[C]}(v) < 2$ do
        $C \leftarrow C \setminus \{v\}$
    end while
    $\tau \leftarrow rand(1, 2)$
    while $C \neq \emptyset$ do
        $v_{best} \leftarrow \operatorname{argmin}_{u \in C} w(u)/d_{[C]}(u)^{\tau}$
        if $rand() < p_{v_{best}}$ then
            $v \leftarrow v_{best}$
        else
            Let $L \subset C$ contain the $l$ vertices with the lowest cost
            Choose $v$ uniformly at random from $L$
        end if
        $S \leftarrow S \cup \{v\}$
        $C \leftarrow C \setminus \{u\}$
        while $\exists v \in C : d_{[C]}(v) < 2$ do
            $C \leftarrow C \setminus \{v\}$
        end while
    end while
    $S \leftarrow$ LocalSearch($S, \tau$)
    $S \leftarrow$ RemoveRedundantVertices($S, \tau$)
    return $S$
end procedure
```

where $\hat{f}$ is the fitness value of the solution S, normalized in the range $[0, 1]$, and ρ is the destruction shape, a parameter that determines the degree of destruction. Since the quality of the solutions in the population is high on average, the destruction rate considered allows maintaining the number of vertices to remove relatively low even from the worst solutions, unlike what happens with Equation 2.2. Figure 3.1 shows the curve of the destruction rate r for different values of ρ.

Then, the procedure removes the M vertices from the solution S in an iterative way. Two different strategies to remove vertices have been considered. The first strategy uses a uniform distribution in which each vertex has the same probability to be removed. The second strategy uses a weighted distribution in which a vertex is removed with a probability proportional to the weight-degree ratio, which is the same cost function used in the randomized greedy construction procedure (see Equation 3.1).

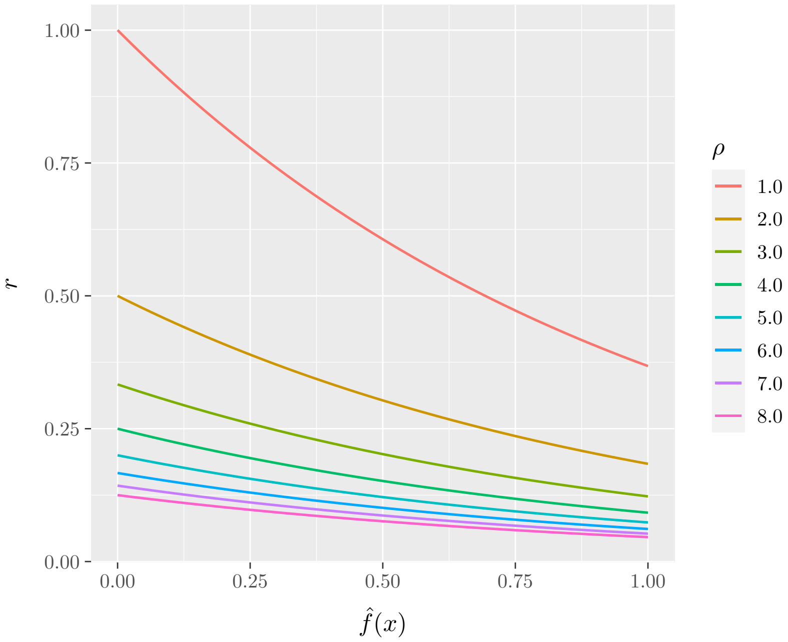


Figure 3.1: Destruction rate r obtained with Equation 3.3 for different values of the destruction shape ρ, with respect to the normalized fitness value.

Finally, the partial solution resulting S' from the destruction phase is then constructed using the procedure GreedyWFVS.

3.1.3 Probability Learning

After the destruction phase, the partial solution is reconstructed using the Greedy-WFVS procedure used in the initial phase. In the case of the WFVS problem, and in particular, during the greedy construction algorithm, the problem of selecting the next node to be inserted in the solution, can be viewed as a reinforcement learning problem. The idea behind the procedure of probability learning is to increase the probability of selecting a vertex in the case of an improvement and decrease the probability in the other case.

At the timestep t, let S' be the solution reconstructed from the solution S using the probability vector $P(t)$. If the new solution improves the parent one, that is if $\Delta(S, S') = f(S) - f(S') > 0$, then the probability learning procedure is applied to update the probability vector. Specifically, if a vertex v stays in the same set (S or $\bar{S} = V \setminus S$) after the reconstruction phase, that is, it remains in solution or out of solution, the procedure rewards that set and update its probability p_v in according to the following equation:

$$p_v(t+1) = \begin{cases} \alpha + (1-\alpha)p_v(t) & v \in S \wedge v \in S' \\ (1-\alpha)p_v(t) & v \notin S \wedge v \notin S', \end{cases} \tag{3.4}$$

where α $(0 < \alpha < 1)$ is the award factor. Note that a reward for the probability to belong to the set $\bar{S}$, is equivalent to a decrease of probability to belong to the set S. On the other hand, when a vertex is inserted into the solution or removed from the solution, the procedure penalizes the former set and compensates the new one according to the following equation:

$$p_v(t+1) = \begin{cases} (1-\gamma)(1-\beta)p_v(t) & v \in S \wedge v \notin S' \\ \gamma + (1-\gamma)\beta + (1-\gamma)(1-\beta)p_v(t) & v \notin S \wedge v \in S', \end{cases} \tag{3.5}$$

where β $(0 < \beta < 1)$ is the penalization factor and γ $(0 < \gamma < 1)$ the compensation factor. These probability updating rules are a modified version of the rules proposed in [158]. Additionally, to forget some old decisions that are no longer helpful and may mislead the construction procedure during the current search, the probabilities are periodically reduced using a smoothing technique. The strategy to smooth the probability vector works as follows. After the updating procedure, if the probability to belong to the solution S of a vertex $v \in V$ reaches a given threshold p_0, i.e., $p_v(t) > p_0$, it is reduced as follows:

$$p_v(t+1) = \delta \cdot p_v(t), \tag{3.6}$$

where δ $(0 < \delta < 1)$ is a smoothing coefficient. On the other hand, a reduction of the probability to belong to the set $\bar{S}$ is equivalent to an increase of probability to belong to the set S by a factor $1 - \delta$, that is:

$$p_v(t+1) = (1-\delta)(1-p_v(t)) + p_v(t). \tag{3.7}$$

3.2 Experimental Results

In this section, all experimental results of the proposed framework are presented in order to evaluate its effectiveness on grouping problems. All experiments were conducted with the same parameter configuration. We used a population of $n = 100$ solutions with a duplication factor $d = 1$. The size of the candidates' list used in the construction phase was set to $l = 10$, while the destruction shape was set to $\rho = 2.0$. The parameters of the probability learning procedure, reward (α), penalization (β) and compensation (γ) coefficients, were set all three to 0.01. The smoothing probability threshold and the smoothing coefficient were set to $p_0 = 0.95$ and $\delta = 0.5$ respectively. The choice of these values was empirical and dictated by an initial experimental phase. Consequently, these parameters need to be further analysed in the future.

As said before, we considered the Weighted Feedback Vertex Set problem as case study. In particular, we focused our experiments on large instances of the benchmark set introduced in [30] and described in Section 2.2. We decided to leave out the small instances because the immunological algorithm Hybrid-IA already has good performances reaching the optimal value for all kinds of graphs.

In the following tables are shown all experimental results of Greedy-IA with both strategies for the destruction phase (see Section 3.1.2). The structure of these tables is as follows. The first five columns provide the instance identifier, the size (x and y for grid and toroidal graphs, n and m for random and hypercube graphs) and the lower and upper bound of the weight range (*Low* and *Up*). The sixth column provides the best value K^* between ITS [30], XTS [24, 52] and MA [28] algorithms. The last four columns provide the results of Greedy-IA, where the U or the W as subscript identify the *Uniform* or the *Weighted* destruction phase (see Section 3.1.2). The columns labelled with *Avg* provide the average solution quality over the five random instances, while the *Diff* columns represent the difference between the result obtained by Greedy-IA and the best result among the three compared algorithms, that is, $Avg - K^*$. The last row of each table provides the average of solution quality and the

Average Gap Value (AGV) (see Equation 2.5 in Section 2.2.4) over the whole table. Finally, in boldface are highlighted the results of Greedy-IA that improve, or at least equal, the lower bound represented by K^*. Figures 3.2-3.6 show the AGV of Greedy-IA as a function of the size of instances.

Table 3.1 shows the results of Greedy-IA for large random graphs. From these results, we can observe that Greedy-IA reaches K^* for 9 out of 15 high-density instances, while for 5 out of 15 instances, the obtained results are slightly above K^*, with a difference of at most $+1.4$. Only in one occasion, `L_R43`, Greedy-IA with the uniform destruction operator improves K^* (-0.4), obtaining what is most likely the optimal solution. For low-density random instances, both versions of the Greedy-IA algorithm have very similar performance on almost all instances. The proposed algorithm reaches K^* for 2 out of 15 instances (`L_R1` and `L_R3`), and improves the fitness value for 7 out of 15 instances. For medium-density instances, instead, Greedy-IA obtains the best solution for 5 out of 15 instances, improving K^* for 3 instances (`L_R14`, `L_R15` and `L_R31`). On the other hand, for low and medium instances where Greedy-IA fails, the difference of obtained solutions from K^* is minimal for instances with 100 vertices ($+0.2$) and grows slightly as the instance size increases, reaching a maximum of $+7.2$ for a 500 vertices instance (`L_R42`). Inspecting the plot in Figure 3.2, Greedy-IA obtains on average an AGV comparable to that obtained by MA for instances with 100 and 200 vertices, and always lower for larger instances, i.e., with 300, 400 and 500 vertices.

The experimental results of Greedy-IA for large toroidal graphs are reported in Table 3.2. For these graphs, Greedy-IA obtains the best solution for 11 out of 15 instances. Both destruction strategies adopted, Greedy-IA$_{\text{U}}$ and Greedy-IA$_{\text{W}}$, obtain similar results for instances with 100 and 196 vertices, reaching K^* for `L_T2`, `L_T5` and `L_T6`, while the maximum difference of $+1.6$ and $+1.8$ from K^* is obtained for the instance `L_T1`. For all instances with 289, 400 and 529 vertices, both versions of the proposed algorithm have similar performance. Greedy-IA obtains the best solution for almost all instances, with an improvement that slightly increases as the instance

Table 3.1: Results of Greedy-IA on large instances of random graphs.

	Instance					Greedy-IA$_{\mathrm{U}}$		Greedy-IA$_{\mathrm{W}}$	
Id	n	m	Low	Up	K^*	Avg.	Diff.	Avg.	Diff.
L_R1	100	247	10	25	498.4	**498.4**	+0.0	**498.4**	+0.0
L_R2	100	247	10	50	836.8	**835.0**	-1.8	**835.4**	-1.4
L_R3	100	247	10	75	1207.6	**1207.6**	+0.0	**1207.6**	+0.0
L_R4	100	841	10	25	826.8	**826.8**	+0.0	**826.8**	+0.0
L_R5	100	841	10	50	1724.4	**1724.4**	+0.0	**1724.4**	+0.0
L_R6	100	841	10	75	2420.6	2420.8	+0.2	2420.8	+0.2
L_R7	100	3069	10	25	1134.0	**1134.0**	+0.0	**1134.0**	+0.0
L_R8	100	3069	10	50	2179.0	**2179.0**	+0.0	**2179.0**	+0.0
L_R9	100	3069	10	75	3228.6	**3228.6**	+0.0	**3228.6**	+0.0
L_R10	200	796	10	25	1468.2	**1465.2**	-3.0	**1466.6**	-1.6
L_R11	200	796	10	50	2399.0	2400.2	+1.2	2401.0	+2.0
L_R12	200	796	10	75	3089.6	3090.2	+0.6	3092.0	+2.4
L_R13	200	3184	10	25	1986.2	1987.0	+0.8	1987.0	+0.8
L_R14	200	3184	10	50	3650.6	**3649.4**	-1.2	3652.8	+2.2
L_R15	200	3184	10	75	5135.8	**5135.6**	-0.2	5144.0	+8.2
L_R16	200	12139	10	25	2447.8	2448.6	+0.8	2448.0	+0.2
L_R17	200	12139	10	50	4148.6	**4148.6**	+0.0	4149.6	+1.0
L_R18	200	12139	10	75	5528.4	5528.8	+0.4	5528.8	+0.4
L_R19	300	1644	10	25	2045.4	**2044.6**	-0.8	**2045.0**	-0.4
L_R20	300	1644	10	50	4175.4	4176.6	+1.2	4176.6	+1.2
L_R21	300	1644	10	75	6065.2	6067.8	+2.6	6069.6	+4.4
L_R22	300	7026	10	25	3203.0	3204.2	+1.2	3206.4	+3.4
L_R23	300	7026	10	50	6211.0	6218.8	+7.8	6213.8	+2.8
L_R24	300	7026	10	75	8585.4	8587.8	+2.4	8590.2	+4.8
L_R25	300	27209	10	25	3726.6	3727.0	+0.4	3727.0	+0.4
L_R26	300	27209	10	50	5734.8	**5734.8**	+0.0	**5734.8**	+0.0
L_R27	300	27209	10	75	10467.0	**10467.0**	+0.0	**10467.0**	+0.0
L_R28	400	2793	10	25	2989.6	2992.0	+2.4	**2989.0**	-0.6
L_R29	400	2793	10	50	6410.0	6413.4	+3.4	6413.8	+3.8
L_R30	400	2793	10	75	8597.2	**8590.6**	-6.6	**8588.6**	-8.6
L_R31	400	12369	10	25	4428.8	4429.4	+0.6	**4427.0**	-1.8
L_R32	400	12369	10	50	6785.8	6787.4	+1.6	6789.0	+3.2
L_R33	400	12369	10	75	10599.4	10600.8	+1.4	10600.8	+1.4
L_R34	400	48279	10	25	5060.4	5060.6	+0.2	5061.0	+0.6
L_R35	400	48279	10	50	7106.8	**7106.8**	+0.0	**7106.8**	+0.0
L_R36	400	48279	10	75	15103.2	**15103.2**	+0.0	**15103.2**	+0.0
L_R37	500	4241	10	25	4056.4	**4054.6**	-1.8	**4053.0**	-3.4
L_R38	500	4241	10	50	7170.4	7180.8	+10.4	**7170.0**	-0.4
L_R39	500	4241	10	75	11135.6	11137.8	+2.2	11138.8	+3.2
L_R40	500	19211	10	25	5724.2	5725.2	+1.0	5728.8	+4.6
L_R41	500	19211	10	50	7677.8	7680.6	+2.8	7684.4	+6.6
L_R42	500	19211	10	75	14124.8	14132.0	+7.2	14132.2	+7.4
L_R43	500	75349	10	25	6361.6	**6361.2**	-0.4	6362.0	+0.4
L_R44	500	75349	10	50	8668.4	**8668.4**	+0.0	**8668.4**	+0.0
L_R45	500	75349	10	75	16932.4	16933.8	+1.4	16933.8	+1.4
AVG					5401.27	5402.12	+0.85	5402.35	+1.08

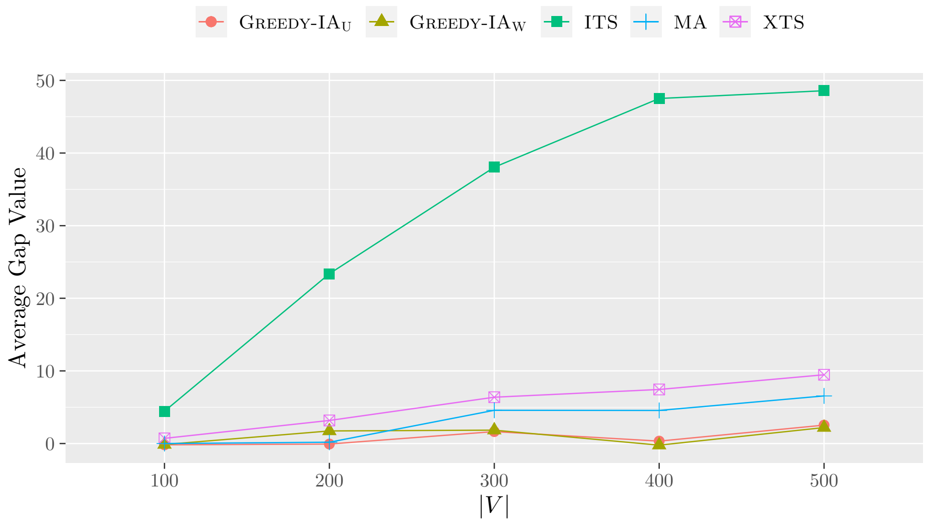


Figure 3.2: The average gap value of Greedy-IA on the large random graphs.

Table 3.2: Results of Greedy-IA on large instances of toroidal graphs.

	Instance					Greedy-IA$_U$		Greedy-IA$_W$	
Id	x	y	Low	Up	K^*	Avg.	Diff.	Avg.	Diff.
L_T1	10	10	10	25	388.0	389.6	+1.6	389.8	+1.8
L_T2	10	10	10	50	457.6	**457.6**	+0.0	**457.6**	+0.0
L_T3	10	10	10	75	504.6	504.8	+0.2	504.8	+0.2
L_T4	14	14	10	25	748.8	749.2	+0.4	749.6	+0.8
L_T5	14	14	10	50	874.4	**874.4**	+0.0	876.4	+2.0
L_T6	14	14	10	75	1016.2	**1016.2**	+0.0	**1016.2**	+0.0
L_T7	17	17	10	25	1102.8	1103.8	+1.0	**1102.0**	-0.8
L_T8	17	17	10	50	1304.4	**1304.2**	-0.2	1305.2	+0.8
L_T9	17	17	10	75	1498.6	1500.4	+1.8	**1498.4**	-0.2
L_T10	20	20	10	25	1539.6	**1536.2**	-3.4	**1537.0**	-2.6
L_T11	20	20	10	50	1795.4	1795.6	+0.2	**1794.8**	-0.6
L_T12	20	20	10	75	2033.0	2034.4	+1.4	2034.4	+1.4
L_T13	23	23	10	25	2034.8	2035.8	+1.0	**2031.8**	-3.0
L_T14	23	23	10	50	2406.4	**2403.8**	-2.6	**2404.6**	-1.8
L_T15	23	23	10	75	2697.2	**2696.8**	-0.4	2698.2	+1.0
AVG					1360.12	1360.19	+0.07	1360.05	-0.07

size increases (from -0.2 for `L_T8` to -3.0 for `L_T13`). For `L_T12`, both versions of the algorithm reach a solution greater +1.4 than K^*. From Figure 3.3, the AGV obtained by Greedy-IA is less than 1.0 for instances with 100 and 196 vertices, and decreases as the instance size increases, reaching negative values for instances with 400 and 529 vertices.

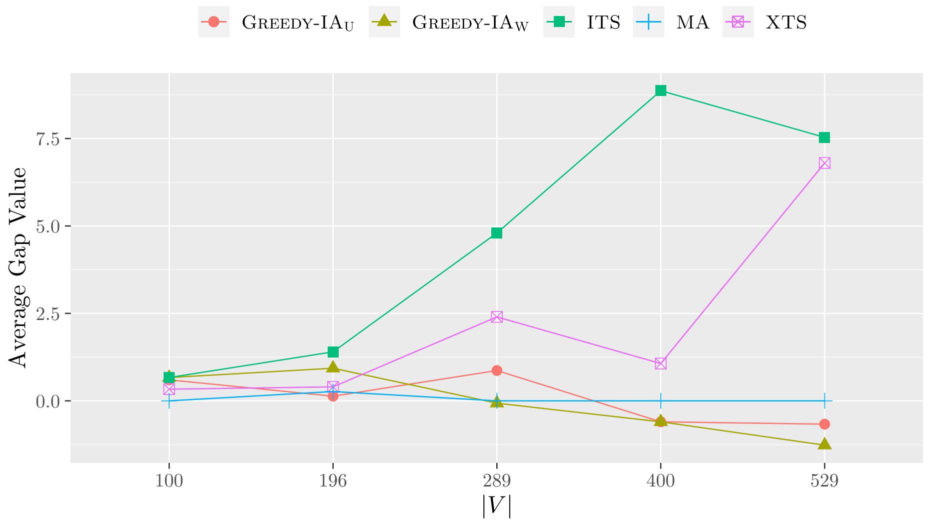


Figure 3.3: The average gap value of Greedy-IA on the large toroidal graphs.

Table 3.3 shows the experimental outcomes for large hypercube graphs. Also, in this case, both versions of the algorithm Greedy-IA have very similar performance for instances with 128 and 256 vertices. Both destruction strategies achieve slightly higher solutions than K^*, with a similar difference, except for the instance L_H4, where Greedy-IA$_U$ obtain the best solution (-0.2). On the other hand, for instances with 512 vertices, the situation is the opposite. Greedy-IA$_U$ and Greedy-IA$_W$ achieve the best results by improving the K^* for all three weight ranges, with a maximum increment of -16.4 for L_H8 obtained by Greedy-IA$_W$. The AGV in Figure 3.4 reflects the results shown in Table 3.3. The AGV of Greedy-IA is comparable to that obtained by MA for instances with 128 and 256 vertices, with a gap between 1.0 and 2.0, while it is negative for instances with 512 vertices.

Finally, in Tables 3.4 and 3.5 are reported the results of Greedy-IA for large squared and not squared grid graphs. From these results it can be noted as the algorithm, with both destruction strategies, significantly improves the lower bound K^*. In particular, Greedy-IA obtains the best solution in 15 out of 15 squared grid instances and in 14 out of 15 not squared grid instances. Only in one case, the algorithm does not achieve the best solution, i.e., for L_NG2, with a difference of $+0.2$ for both algorithms. For instances with 400 and 390 vertices, the biggest improvement is for

Table 3.3: Results of Greedy-IA on large instances of hypercube graphs.

	Instance					Greedy-IA$_{\mathrm{U}}$		Greedy-IA$_{\mathrm{W}}$	
Id	n	m	Low	Up	K^*	Avg.	Diff.	Avg.	Diff.
L_H1	128	448	10	25	731.8	733.8	+2.0	733.8	+2.0
L_H2	128	448	10	50	1066.8	1068.6	+1.8	1069.4	+2.6
L_H3	128	448	10	75	1161.6	1162.4	+0.8	1162.4	+0.8
L_H4	256	1024	10	25	1487.4	**1487.2**	-0.2	1487.8	+0.4
L_H5	256	1024	10	50	2279.6	2281.8	+2.2	2282.6	+3.0
L_H6	256	1024	10	75	2572.4	2573.2	+0.8	2573.2	+0.8
L_H7	512	2304	10	25	3119.0	**3116.8**	-2.2	**3116.6**	-2.4
L_H8	512	2304	10	50	4852.2	**4842.0**	-10.2	**4835.8**	-16.4
L_H9	512	2304	10	75	5553.4	**5550.2**	-3.2	**5551.8**	-1.6
AVG					2536.02	2535.11	-0.91	2534.82	-1.20

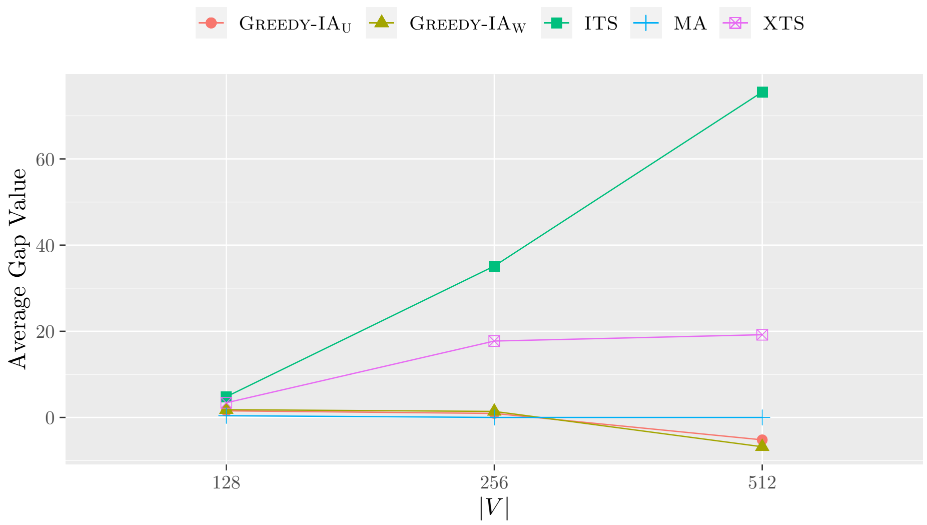


Figure 3.4: The average gap value of Greedy-IA on the large hypercube graphs.

instances with weight range $[10, 75]$, `L_SG12` and `L_NG12`, with a difference of -27.8 and -35.8 respectively. For the other ranges, $[10, 25]$ and $[10, 50]$, the improvement is smaller. Same situation for instances with 529 and 493 vertices, where the best solution increment occurs for instances with weight range $[10, 75]$, `L_SG15` (-44.2) and `L_NG15` (-72.8). From the plots in Figures 3.5-3.6, Greedy-IA obtains a negative AGV for both kinds of graphs, with a gap from K^* that increases as the instance size increases.

In the overall, from the results reported in Tables 3.1-3.5 and Figures 3.2-3.6, Greedy-IA with both destruction strategies has similar performance on all tested

Table 3.4: Results of Greedy-IA on large instances of squared grid graphs.

	Instance					Greedy-IA$_\mathrm{U}$		Greedy-IA$_\mathrm{W}$	
Id	x	y	Low	Up	K^*	Avg.	Diff.	Avg.	Diff.
L_SG1	10	10	10	25	566.8	567.0	+0.2	**566.6**	-0.2
L_SG2	10	10	10	50	947.0	**946.4**	-0.6	**945.8**	-1.2
L_SG3	10	10	10	75	1557.8	**1557.4**	-0.4	**1556.8**	-1.0
L_SG4	14	14	10	25	1206.4	**1205.0**	-1.4	**1204.6**	-1.8
L_SG5	14	14	10	50	2007.8	**2003.8**	-4.0	**2002.6**	-5.2
L_SG6	14	14	10	75	3399.2	**3393.4**	-5.8	**3392.8**	-6.4
L_SG7	17	17	10	25	1829.4	**1820.2**	-9.2	**1823.2**	-6.2
L_SG8	17	17	10	50	3051.4	**3050.8**	-0.6	**3051.0**	-0.4
L_SG9	17	17	10	75	5071.2	**5062.0**	-9.2	**5065.2**	-6.0
L_SG10	20	20	10	25	2602.2	**2594.4**	-7.8	**2591.6**	-10.6
L_SG11	20	20	10	50	4299.8	**4290.2**	-9.6	**4290.0**	-9.8
L_SG12	20	20	10	75	7240.4	**7212.6**	-27.8	**7215.2**	-25.2
L_SG13	23	23	10	25	3453.6	**3444.2**	-9.4	**3441.8**	-11.8
L_SG14	23	23	10	50	5831.4	**5820.8**	-10.6	**5814.2**	-17.2
L_SG15	23	23	10	75	9681.0	**9645.8**	-35.2	**9636.8**	-44.2
AVG					3516.36	3507.60	-8.76	3506.55	-9.81

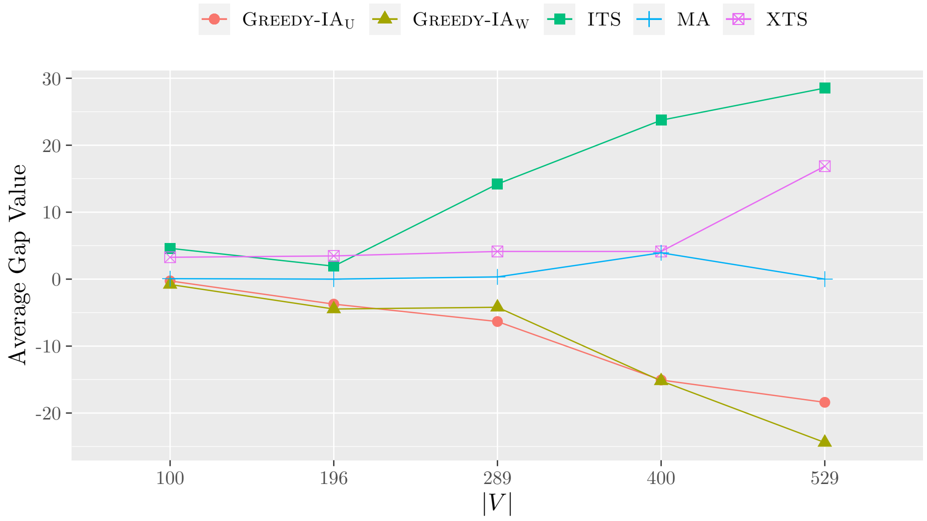


Figure 3.5: The average gap value of Greedy-IA on the large squared grid graphs.

random instances, with solutions that differ from K^* on average by +0.85 for Greedy-IA$_\mathrm{U}$ and +1.08 for Greedy-IA$_\mathrm{W}$. The same considerations can be done for toroidal instances, where both destruction strategies are equivalent, obtaining on average results that differ slightly from K^*, +0.07 for the uniform and -0.07 for the weighted. For the hypercube graphs, Greedy-IA obtains on average a negative AGV (-0.91 for

Table 3.5: Results of Greedy-IA on large instances of not squared grid graphs.

	Instance					Greedy-IA$_{\text{U}}$		Greedy-IA$_{\text{W}}$	
Id	x	y	Low	Up	K^*	Avg.	Diff.	Avg.	Diff.
L_NG1	13	7	10	25	512.2	**511.6**	-0.6	**512.2**	+0.0
L_NG2	13	7	10	50	803.4	803.6	+0.2	803.6	+0.2
L_NG3	13	7	10	75	1382.8	**1382.4**	-0.4	**1382.0**	-0.8
L_NG4	18	11	10	25	1204.6	**1202.0**	-2.6	**1203.8**	-0.8
L_NG5	18	11	10	50	2041.2	**2040.4**	-0.8	**2038.6**	-2.6
L_NG6	18	11	10	75	3417.4	**3411.4**	-6.0	**3409.8**	-7.6
L_NG7	23	13	10	25	1923.8	**1919.0**	-4.8	**1919.6**	-4.2
L_NG8	23	13	10	50	3178.2	**3172.6**	-5.6	**3176.4**	-1.8
L_NG9	23	13	10	75	5258.0	**5249.0**	-9.0	**5252.8**	-5.2
L_NG10	26	15	10	25	2522.6	**2516.4**	-6.2	**2515.8**	-6.8
L_NG11	26	15	10	50	4144.2	**4142.0**	-2.2	**4142.6**	-1.6
L_NG12	26	15	10	75	7031.4	**7005.4**	-26.0	**6995.6**	-35.8
L_NG13	29	17	10	25	3255.0	**3241.2**	-13.8	**3246.0**	-9.0
L_NG14	29	17	10	50	5410.8	**5399.8**	-11.0	**5398.0**	-12.8
L_NG15	29	17	10	75	8953.0	**8886.0**	-67.0	**8880.2**	-72.8
AVG					3402.57	3392.19	-10.39	3391.80	-10.77

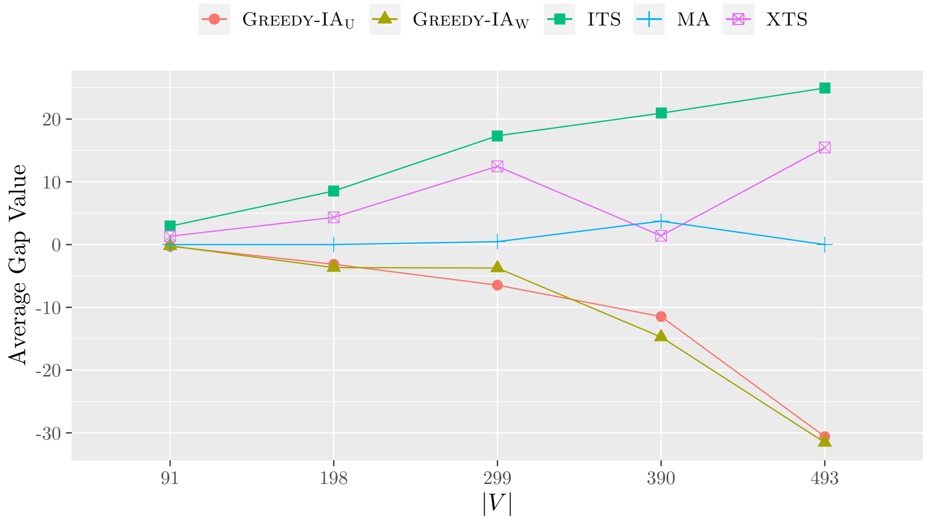


Figure 3.6: The average gap value of Greedy-IA on the large not squared grid graphs.

Greedy-IA$_{\text{U}}$ against -1.20 Greedy-IA$_{\text{W}}$), with a net improvement on instances with 512 vertices. Finally, Greedy-IA achieves the best results for almost all grid graphs, with an average improvement of -9.81 and of -10.77 on squared and not squared grid instances respectively.

3.3 Conclusions

In this section, we have presented a generalized evolutionary framework based on a greedy population combined with reinforcement learning techniques to solve grouping problems. The proposed algorithm, Greedy-IA, is based on an Immunological Algorithm that maintains a population of solutions generated with a randomized greedy procedure. At each iteration, these solutions are repeatedly improved by alternating a phase of destruction and reconstruction, while probability learning exploits useful information from visited local optima to guide the search process toward new promising regions of the search space.

To evaluate the proposed algorithm, we have considered the Weighted Feedback Vertex Set as a case study, a well-known combinatorial optimization problem with several real applications. From the analysis of experimental results, the proposed algorithm outperforms the other compared algorithms on the grid instances of the benchmark dataset, while it has comparable performance on random, toroidal and hypercube instances. Moreover, since the initial setting of the fundamental parameters is arbitrary, the proposed algorithm has great room for improvement.

In future work, we plan to investigate in more detail how the updating rules affect the probabilities of the solution components during the evolutionary process. Furthermore, we would like to apply the proposed framework to other combinatorial optimization problems.

4

Community Detection

In the modern interdisciplinary sciences, complex networks are a powerful interpretation tool useful for the analysis and representation of a wide number of real-world systems and are widely involved in many areas, such as for instance neuroscience, biology, social sciences, economics, and physics. With this graph-based model, it is possible to represent connections and interactions of the underlying entities, where vertices are the elementary parts of the real systems, while edges represent their mutual interactions [108, 18]. Complex networks may contain specific groups of highly interconnected vertices organized in compartments or structure, where each of them has a role and/or a function that satisfy a specific property of cohesion. In terms of graph theory, compartments are represented by partitions of the set of nodes with high internal links density, called *communities* or *modules*, which are loosely associated with other groups [66, 62].

Finding compartments in a graph-theoretic context is a fundamental issue in the study of network systems, in which often they exhibit significantly different functions and, therefore, a global analysis of the network would be inappropriate and impractical. A detailed analysis of individual communities, instead, may shed some light on the organization of systems and leads to more significant insights into the roles of individuals. This approach can also allow the visualization and analysis of large and complex networks focused on a new higher-level structure, in which each identified community can be compressed into a node belonging to the latter.

Community detection is one of the most important research topics in network science and graph analysis, as it allows to understand the dynamics of a complex network at different scales [66, 75, 55], such as for instance connections and interactions between underlying entities, and, consequently, uncover important information that become useful and crucial in many application areas. For instance, the modules detected by biological networks [9] are generally responsible for a common phenotype and are useful in providing insights related to biological functionality. Disease phenotypes are generally caused by the failure of groups of genes that are referred to as the disease form. Since the genes responsible for a phenotype often have common functions, there is a strong association between pathological and functional modules [69, 5, 8]. The detection of modules within biological networks, generally responsible for a common phenotype, is useful and crucial in providing insights into the biological functionality of these genes. The techniques that allow the identification of modules, known as *community detection* techniques, are methods that play a key role in obtaining the functional modules which appear to be closely related to pathological forms, the recognition of which would be useful for the molecular understanding and etiology of the disease. From this would arise the development of specific drugs whose targets would be the genes belonging to these modules.

The aim of community detection in graphs is to identify the modules and their hierarchical organization, by using only the information encoded in the graph topology [115, 62, 105]. In particular, it refers to the division of the nodes of a network into groups such that connections are dense within groups but sparser between them. In other words, a cluster corresponds to a set of nodes with more edges inside the set than to the rest of the graph. Although not all networks support such divisions, the existence of good divisions is often taken as evidence of underlying structure or possible interactive behaviours, making community detection a useful tool to understand how complex networks are structured and work.

Community detection problem gained the attention of scientist communities to bring valuable explanations to complex network analysis. For example, in biology, applying graph clustering methods on relations among genes or proteins, modelled by networks (Protein-Protein Interaction Network) is possible thanks to group proteins having the same specific patterns and mechanisms operating within the cell [32], or through analysis of the network produced by neuron interactions, understanding the functional architecture of the brain [51]. In the same way, it is possible to identify, in information networks, clusters of web pages that share some common topics and similarities in a given social network to find individuals with common interests or friendships. However, modelling and examining complex systems is a very difficult process because the systems used for the real-world data representation contain highly important information: social relationships among people or information exchange and interactions between molecular structures. It follows, then, that the study of community structures in a network is a central issue in better understanding such dynamics, and, for this, it has inspired intense research activities. Indeed, detecting highly linked communities can lead to many benefits, such as understanding how the elements of a network interact and affect each other. Informally, a community in a network is defined as a set of elements that are highly linked within the group and weakly linked to the outside.

A plethora of diverse algorithms and techniques have been proposed for the detection of the communities in real-world networks [106, 107]. They differ from one to the other in criteria implementations for solving community detection problem. They differ also in defining criteria for the identification of communities. These approaches have been applied successfully in different domains of applications and many real-world areas (such as biological, chemical, ecological, economic, political, social, etc.).

4.1 Modularity Optimization

Community detection is then a powerful tool for understanding the structure of complex networks, and ultimately extracting useful information from them. Note that a close connection implies a faster rate of information transmission, instead of a loosely connected community. On the one hand, a network is represented by a number of individual nodes connected by edges, with a certain degree of interaction between some nodes; on the other hand, communities are defined as groups of nodes, densely interconnected, but in sparse order with the rest of the network.

Modularity proposed by Newman et al. [109], is an evaluation measure commonly used for assessing the quality of node partitions detected in a network. Hence, the community detection problem can be easily summed up in finding clustering that maximized Q, whose decision version has been proved to be a $\mathcal{NP}$-complete problem [23]. Originally defined for undirected graphs, has been subsequently extended to directed and weighted graphs [106].

Modularity maximization is one of the most popular and widely used methods for community partition. It detects communities by searching over possible partitions of a graph, over which modularity is maximized. The modularity is 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 same degree distribution.

Formally modularity is defined as follows: 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 by:

$$Q = \frac{1}{2M}\left[\sum_{i=1}^{N}\sum_{j=1}^{N}\left(A_{ij} - \frac{d_i d_j}{2M}\right)\delta(i,j)\right], \tag{4.1}$$

where A is the adjacency matrix of G, d_i and d_j are the degrees of nodes i and j

respectively; $\delta(i, j) = 1$ if i and j belong to the same community, 0 otherwise.

As asserted in [23], the modularity value for unweighted and undirected graphs lies in the range $[-0.5, 1]$, therefore, a low Q value (close to the lower bound) reflects a bad graph partitioning, and implies the absence of real communities; good partitions are instead identified by a higher modularity value that implies the presence of highly cohesive communities. For a trivial clustering, with a single cluster, the modularity value is 0. Interestingly, the modularity has the tendency to produce large communities and, therefore, fails in detecting communities that are comparatively small with respect to the network [63]. Taking into account the Q modularity as an evaluation measure, community detection can easily be seen as a combinatorial optimization problem as the problem aims to find a clustering that maximizes Q.

5

Stochastic Immunological Algorithm

Immune-inspired computation nowadays represents a large and established family of successful algorithms that take inspiration from the mechanisms and dynamics of the immune system with which it protects living organisms. What makes the immune system source of inspiration from an algorithmic perspective is its ability to detect, recognize, and distinguish entities own to the organism from foreign ones, together with its ability to learn new information and remember those foreign entities already recognized. Three principal theories are at the basis of the immune-inspired algorithms: (1) clonal selection [42, 111] (2) negative selection [64, 113]; and (3) immune networks [131]. Among these, what has proven to be quite efficient is the one based on the clonal selection principle (called *Clonal Selection Algorithms* – CSA) [45, 44] mostly in search and optimization applications.

5.1 The Proposed Method

The proposed immune algorithm, Opt-IA, belongs then to this last class of algorithms, and is based on three main immune operators: (i) *static cloning*, whose aim is to generate a new population based on the highest fitness values; (ii) *hypermutation*, that explores the neighbourhood of each point of the search space; and (iii) *stochastic aging*, which removes solutions from the current population via a stochastic law, helping then Opt-IA in escaping from local optima. In addition to these, some diversification strategies have been also designed, whose aim is to keep high and proper diversity in the population and to perform an appropriate exploration of the search space. The Opt-IA algorithm is based on two main concepts, following the biological metaphor: the antigen (Ag), which represents the problem to be solved, and the antibody (Ab), or B cell that is instead a solution for the problem to be solved.

At each timestep t, Opt-IA maintains a population of size d of B cells ($P^{(t)}$), and each B cell Ab represents a subdivision of the vertices of the graph $G = (V, E)$ in communities. In details, if n is the cardinality of the set of vertices V, a B cell $\vec{x} = \{x_1, \ldots, x_n\}$ will be a sequence of n integers, between 1 and n, where $x_i = j$ indicates that the vertex i belongs to the community j. A description of Opt-IA is summarized in the pseudo-code shown in Algorithm 5.1. The proposed algorithm takes as input: the network from which to detect the communities (G); population size (d); the number of copies to be generated for each B cell (dup); the mutation rate (M); the probability that an element will be removed from the population by the aging operator (P_{die}), and the maximum number of generations allowed (T_{max}). It returns as output the communities detected and the relative community number, as well as the *best*, *mean*, *worst* and standard deviation (*StD*) of the modularity used for the comparisons.

As a first step, i.e. at the timestep $t = 0$, Opt-IA randomly generates d solutions using the uniform distribution, creating then the initial population $P^{(t=0)}$ (line 3 of Algorithm 5.1): any vertex is assigned to a community, randomly chosen in the range $[1, n]$, with $n = |V|$. In this way, many communities with few assigned vertices

Algorithm 5.1: Pseudo-code of the stochastic immunological algorithm OPT-IA.

```
 1: procedure OPT-IA($G$, $d$, $dup$, $M$, $P_{die}$, $T_{max}$)
 2:     $t \leftarrow 0$
 3:     $P^{(t)} \leftarrow$ InitializePopulation($d$)
 4:     ComputeFitness($P^{(t)}$)
 5:     repeat
 6:         $P^{(clo)} \leftarrow$ Cloning($P^{(t)}$, $dup$)
 7:         $P^{(mut)} \leftarrow$ Hypermutation($P^{(clo)}$, $M$)
 8:         ComputeFitness($P^{(mut)}$)
 9:         $P^{(agi)} \leftarrow$ StochasticAging($P^{(t)}$, $P_{die}$)
10:         $P^{(pre)} \leftarrow$ Precompetition($P^{(agi)}$)
11:         $P^{(t+1)} \leftarrow (\mu + \lambda)-$Selection($P^{(pre)}$, $P^{(mut)}$)
12:         $t \leftarrow t + 1$
13:     until (termination criterion is satisfied)
14: end procedure
```

will be generated; it will be the task of the developed hypermutation operator (see Section 5.1.2) to compact the communities. Once the population is initialized, the next step is to evaluate the fitness function for each B cell $\vec{x} \in P^{(t)}$ by the function $ComputeFitness(P^{(t)})$ (line 4 of Algorithm 5.1). In this research work, for each B cell $\vec{x}$, such function simply computes the value given by Equation 4.1.

After the initialization of the population, and the computation of the fitness of each generated solution, the artificial evolution process begins, where the key operators take place. As in any evolutionary algorithms, OPT-IA will end its evolution process once a termination criterion is reached, which has been fixed in our experiments to a maximum number of generations allowed (T_{Max}).

5.1.1 The Cloning Operator

The first immune operator to be performed is the cloning operator (line 6 of Algorithm 5.1), which has the main goal of producing a new population with higher affinities (i.e. fitness values), and together with the hypermutation perform a careful local search. Just as it happens in nature that all those cells able to better recognize foreign entities will generate more copies of them, the cloning operator duplicates all those solutions that seem to be promising: simply it copies/clones *dup* times each element of the population, creating a new intermediate population $P^{(clo)}$ of dimensions $d \times dup$. It was developed a static version of the cloning operator, unlike what really happens in biology[1], because this shows the disadvantage to guide easily and quickly the algorithm towards local optima. Furthermore, to avoid premature convergence, we also made dup independent from the fitness function value of the B cell. In a nutshell, if we had chosen to increase the number of clones for high-fitness elements, we would have achieved quickly a very homogeneous population, causing in turn a poor exploration of the search space.

5.1.2 The Hypermutation Operator

The hypermutation operator (line 7 of Algorithm 5.1) acts on each element of the population $P^{(clo)}$ performing M mutations with the main aim to explore the search space, and the neighbourhood of all solutions found so far. Similarly to the parameter dup, also the mutation rate M is a user-defined parameter and is not related to the fitness function of the solution, to avoid possible premature convergence. Importantly, unlike classical evolutionary algorithms, no mutation probability was considered. Furthermore, the introduction of blind mutations produces individuals with higher affinity (i.e. higher fitness function values), which will be then selected to form improved mature progenies. In this research work different types of mutation operators have been developed, which can act on a single vertex in the solution, like a local operator, or on a group of vertices, like a global operator: *equiprobability*, *existing*, *total random*, *destroy* and *fuse* operators.

Equiprobability operator. This mutation operator is locally applied and tries to find a better neighbour not yet explored. Simply, it randomly selects a vertex i, and a community c_j among those existing at that moment, and, of course, different from the one to which i belongs (i.e. $c_i \neq c_j$), and, then, the vertex i is moved into the community c_j.

[1] In nature, the number of clones produced for a B cell is proportional to its ability to detect and recognise the Ag.

Existing operator. This mutation operator, randomly selects a vertex i and moves it to the community c_j, where j is a randomly selected vertex belonging to a different community of i, that is $c_i \neq c_j$. In this way, the larger the size of community c_j, the greater the probability that vertex i will be assigned to that community.

Total Random operator. This operator, like the *equiprobability* operator, randomly selects a vertex i, and a community c_j among the $n = |V|$ possible.

Destroy operator. This operator works through a more global perspective than the first operator, as it acts directly on the communities rather than the single vertices. It is carried out as follows: two different communities are randomly selected, c_i and c_j, which are, respectively, the community from where the vertices will be moved; and the one that will receive such vertices. In particular, c_i is selected among the currently existing communities, whilst c_j is randomly assigned a value in the range $[1, n]$, with $n = |V|$. Note that this means that c_j could have a value that does not correspond to any existing community. After that, a probability is randomly chosen, between 1% and 50%, and on the basis of this probability, every vertex in c_i will move into c_j. As already said, if c_j is among the existing ones, then the new vertices will increase the community itself; otherwise, a new community is created and added to the others.

Fuse operator. This last operator has the main aim to try to reduce and compact the communities. Thus, it chooses randomly two different communities c_i and c_j and moves all vertices belonging to c_i into the community c_j, decreasing consequently by one the number of communities.

The effectiveness and efficiency of these mutation operators have already been proven in [132] through a comparative analysis (Figure 5.1). In particular, the first two have proven useful in improving the modularity function value, and in escaping to local optima; the *fuse* operator, instead, albeit does not offer good results individually, also has been taken into account since its goal is to aggregate communities. Trying then to take advantage of the characteristics of each operator, the *equiprobability* and *destroy* operators are applied with the same probability, while the last with low probability for the above reasons (around 49.5%, 49.5%, and 1%).

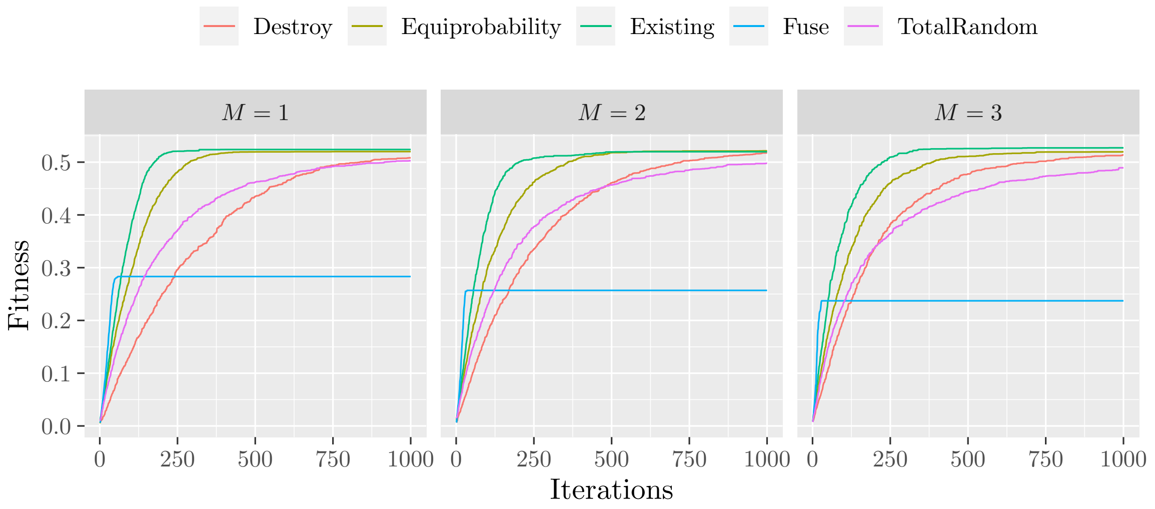


Figure 5.1: A comparative analysis on the performances of some mutation operators designed for community detection.

Finally, once the hypermutation was performed, and, then modify all the cloned solutions, for each mutated B cell is computed its fitness value through the function $ComputeFitness(P^{(t)})$.

5.1.3 Aging, Precompetition and Selection Operators

The aging operator has the main goal of helping the algorithm in jumping out of local optima and keep a high diversity inside the population. In this research work, the stochastic aging operator (line 9 of Algorithm 5.1) has been developed that guides Opt-IA to reduce premature convergences as much as possible. In detail, the elements of the population are removed at each iteration with a P_{die} probability, which is a user-defined parameter. Diversification of solutions in a population is a crucial feature to avoid getting trapped into local optima. However, this can also become a limitation in carrying out a careful, and accurate exploration of their neighbourhoods, which also plays a key role in the success of the algorithm. Thus, in order to have the right balance between these two key features, the stochastic aging is applied only to the $P^{(t)}$ population. In this way, diversity will be introduced in the population of the best current solutions, which will then compete with their offspring in generating the new population of the next generation; whilst the B cells in $P^{(mut)}$ will have the task of properly exploring the neighbourhoods.

After the aging operator, to strengthen heterogeneity in the population $P^{(t)}$, a pre-competition step (line 10 of Algorithm 5.1) has been developed, which simply randomly selects two different B cells from $P^{(t)}$: if the two solutions, although different, have the same number of communities, then the one with lower fitness value will be removed from $P^{(t)}$ with a 50% probability. This strategy allows, thus, to maintain a more heterogeneous population during the evolutionary cycle, maintaining solutions with a different number of communities, in order to better explore the search space.

The last operator before the end of the iteration is the creation of the new population for the next timestep $t + 1$. In Opt-IA, the $(\mu + \lambda)$-selection operator (line 11 of Algorithm 5.1) has been developed, which selects the best d B cells from the two populations $P^{(pre)}$ and $P^{(mut)}$, without fitness repetitions. This selection operator, with $\mu \leq d$ and $\lambda = (d \times dup)$, identifies the best $\mu = d$ elements from the offspring set ($P^{(mut)}$), and old parent B cells ($P^{(pre)}$), therefore ensuring monotonicity in the evolution dynamics. If two selected elements have the same fitness, then, only one of them will be chosen randomly.

5.2 Behaviour Analysis

In this section, we study the features of Opt-IA and analyse its behaviour in order to prove its efficiency and reliability. We start by describing the data sets, i.e. the complex networks used for our studies and comparisons, along with the experimental protocol adopted for all the performed tests. Likewise, we show the experimental tuning on the population size, mutation rate and aging probability. Then, once the best setting of Opt-IA's parameters is determined, we show its dynamics and learning abilities. Finally, we present the analysis of the time complexity of Opt-IA via the well-known *Time-To-Target* plots, which are a classical methodology for running time analysis for any stochastic algorithm.

Table 5.1: Social and biological network instances used in the experiments.

Name	*Type*	$\|V\|$	$\|E\|$	$\Delta(G)$
Grevy's Zebras	*social-ecological*	28	111	29.37%
Zachary's Karate Club	*social*	34	78	13.90%
Bottlenose Dolphins	*social-ecological*	62	159	8.41%
Books about US Politics	*social*	105	441	8.08%
American College Football	*social*	115	613	9.35%
Jazz Musicians	*social*	198	2742	14.06%
Cattle PPI	*biological*	268	303	0.85%
E. coli TRN	*biological*	418	519	0.60%
C. elegans MRN	*biological*	453	2025	1.98%
H. pylori PPI	*biological*	724	1403	0.54%
E. coli MRN	*biological*	1039	4741	0.88%
S. cerevisiae PPI (1)	*biological*	2018	2705	0.13%
S. cerevisiae PPI (2)	*biological*	2284	6646	0.25%

5.2.1 Datasets and Experimental Protocol

The Opt-IA algorithm, presented in Section 5.1, has been tested and studied on eight different real-world networks, which include six biological, and six social networks. These networks are related to two different areas and are obtained from real-world systems. Features and size of each network are detailed in Table 5.1.

More in detail, we considered six social networks: a small size network, called *Zachary's Karate Club*, and two larger ones, respectively called *Books about US politics* and *American College Football*. The Zachary's Karate Club network represents the friendships between members of a university's karate club in the US over a period of 2 years [154]. It has come to be a standard test network for clustering algorithms. Each vertex represents a member of the club and each edge represents the relationship between the two corresponding members of the club. The network called *American College Football* was presented in [66] and represents the football match schedule for the 2000 season. Vertices in the graph represent the teams, while edges represent regular season games between the two corresponding teams. In the *Books about US politics* network, the vertices represent the books on American politics purchased from amazon.com, while edges connect pairs of books which are frequently co-purchased.

The books in this network, compiled by V. Krebs [88], were classified by Newman [105] into liberal or conservative categories, with the exception of a small number of books without a clear ideological bias. *Jazz Musicians* [67] is the collaboration network between Jazz musicians. Each vertex is a Jazz musician and an edge denotes that two musicians have played together in a band.

We also took into consideration two different types of networks, which identify social-ecological networks, called respectively *Bottlenose Dolphins* [98] and *Grevy's Zebras* [135]. Bottlenose dolphins are aquatic mammals in the genus Tursiops. The network is built from a community of bottlenose dolphin community living in New Zealand and observed between 1994 and 2001. Edges denote frequent association. In the *Grevy's Zebras* network, edges represent the interactions between two Equus grevyi (vertex in the network) if they existed between them during the study.

Finally, six biological networks were also considered, namely: *E. coli transcription* [129], *C. elegans metabolic reaction* [57], *Cattle protein–protein interactions* [31], *H. pylori protein-protein interactions* [149, 116], *E. coli metabolic reaction* [123], and two *S. cerevisiae protein–protein interactions* networks [153, 25]. In particular, in the gene expression *E. coli TRN* network, which is a commonly used benchmark, the vertices represent operons, i.e. functioning units of DNA containing a cluster of genes, and edges are directed from a gene that encodes a transcription factor to a gene that it directly regulates it [129].

Overall, all these networks, social and biological data in real work systems, are well-known and commonly used datasets for evaluating the efficacy and efficiency of designed algorithms for the community detection problem.

All the performed experiments were carried out on 30 independent runs, whilst the considered stopping criterion was fixed at a maximum number of generations allowed ($T_{max} = 4 \times 10^3$). Since Opt-IA is basically a blind search algorithm, it follows that without knowledge about the domain and without the inclusion of any deterministic refinement approach, obviously Opt-IA must evolve for more generations than any hybrid/memetic approach or a hyper-heuristic. Nevertheless, the computational time of Opt-IA for reaching the overall best solution is, however, acceptable, as it is shown in Section 5.2.4.

Finally, for each experiment, and then for each network, the following outcomes were computed and shown: *Best* modularity found on all runs, *Mean* value of the best solutions found per each run, worst modularity value found on the overall, standard deviation (σ) and the number of communities discovered (N_C).

5.2.2 Parameters Tuning

As described in Section 5.1 (see Algorithm 5.1), the crucial parameters that affect the performances of Opt-IA are, respectively: (*i*) the population size (d), (*ii*) the duplication factor (dup), (*iii*) the mutation rate (M) and (*iv*) the probability to remove a B cell at each iteration (P_{die}). Therefore, we need to find the best values for these parameters.

In a preliminary work [132], a small and sparse network (*almost_lattice* network with 64 vertices) was used to find the best setting for the parameter values. The network was chosen since it shows a particular complex landscape. From this preliminary study, the best parameter combinations obtained were, respectively, $d = 8$, and $dup = 4$, whilst mutation rate M varied between 1 to 3. The efficacy of such a setting was also validated by the comparison of Opt-IA with the well-known greedy optimization method Louvain [12], one of the most popular algorithms for community detection. The comparison, which for convenience is reported in Table 5.3 (see Section 5.3), shows that the proposed algorithm outperforms Louvain in all networks tested.

Following these very good results, we ran Opt-IA on all the networks which are shown in Table 5.1 using the same parameter configuration. Although the algorithm was able to find the optimal solutions for some of the social networks, on the larger ones, such as for instance, on the biological networks, we obtained, instead, very poor results. In light of this, we performed a new study on the parameter tuning, but this time we took into account the *Cattle PPI* biological network as a testbed, which is a large and sparse graph, and consequently a hard enough testbed. It is important to highlight, also, that all experiments we conducted for the parameters tuning were performed on 30 independent runs, to have more robust and reliable outcomes.

Based on the knowledge acquired on Opt-IA in previous research works [111, 46, 47], the population size was set to $d = 100$, since it is mainly related to the dimension and complexity of the problem tackled. For all the other parameters, instead, the tuning was determined by evaluating the fitness trend at their different values. The first step of this study was conducted on the mutation rate parameter with five different values ($M = \{1, 2, 3, 4, 5\}$), using the following parameter configuration settings: population size $d = 100$, duplication parameter $dup = 4$, probability of random aging operator $P_{die} = 0.02$ and $T_{Max} = 4000$.

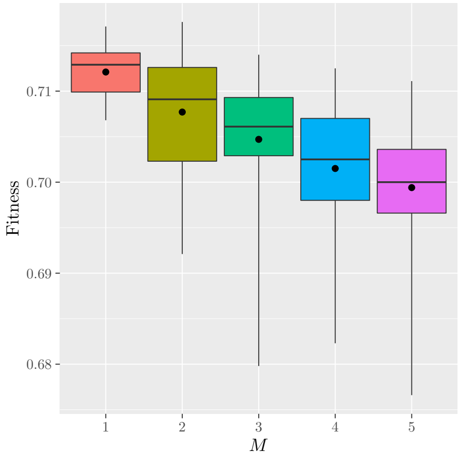


Figure 5.2: Results of the tuning for the parameter M on the *Cattle PPI* network.

The outcomes of these experiments are reported in Figure 5.2 where we can see the distribution of fitness values over 30 independent runs at the varying of M on the *Cattle PPI* network. By observing the graph in the figure, it is possible to assert that by increasing the number of mutations the performance of Opt-IA decreases considerably with respect to both the best modularity found and the mean. The best performances are obtained using small mutation rate values, that is $M = \{1, 2\}$. It is important to note that although for $M = 1$ Opt-IA reaches a better median, the performances for $M = 1$ and $M = 2$ are however equivalent when compared to the best value found. For this reason, both these values for M have been taken into account for the next experiments. The good results obtained when performing a lower number of mutations are primarily due to the effect of the aging and precompetition operators, which produce good heterogeneity, and consequently, require the perturbation operator to carry out the search in the neighbourhood of the current solutions.

Once the mutation rate $M = \{1, 2\}$ was set, we performed a second step of parameter tuning to determine the best combination of parameters dup and P_{die}. In particular, the duplication parameter was varied in the set $\{2, \ldots, 10\}$, whilst P_{die} in $\{0, 0.01, 0.02, 0.05, 0.15, 0.3, 0.5\}$.

From both Figures 5.3 and 5.4, it is clear that better performances are obtained when increasing the value of dup. In particular, higher modularity values are obtained for $dup = \{8, 9, 10\}$ for both M values. In other words, having more copies of each solution helps Opt-IA in carrying out a careful and more accurate search in its neighbourhood. Focusing further the analysis only on these last three values, it is possible to assert that for $dup = 9$ and $dup = 10$, Opt-IA finds the best modularity more often than for $dup = 8$.

Inspecting now the plots from the perspective of parameter P_{die} and considering these last two values for dup, we can see that the better performances of Opt-IA were obtained for $P_{die} = 0.02$. With this value, indeed, the algorithm was able to find a better mean, and a lower standard deviation (σ). Each candidate solution will have a 0.02 probability to be removed from the population and such a low probability value is enough to produce a good heterogeneity in the population (when, of course, combined with the precompetition operator). From these last experiments, it also emerges that for $M = 1$ the performances of Opt-IA are considerably better than for $M = 2$, since the algorithm reaches the best solution more often, with a better mean and standard deviation, proving, in turn, greater robustness and soundness.

In conclusion, from the overall experimental analysis, the best parameter combination obtained is the following: $d = 100$, $dup = \{9, 10\}$, $M = 1$ and $P_{die} = 0.02$. Note that, although $dup = 10$ showed slightly better performances on the considered

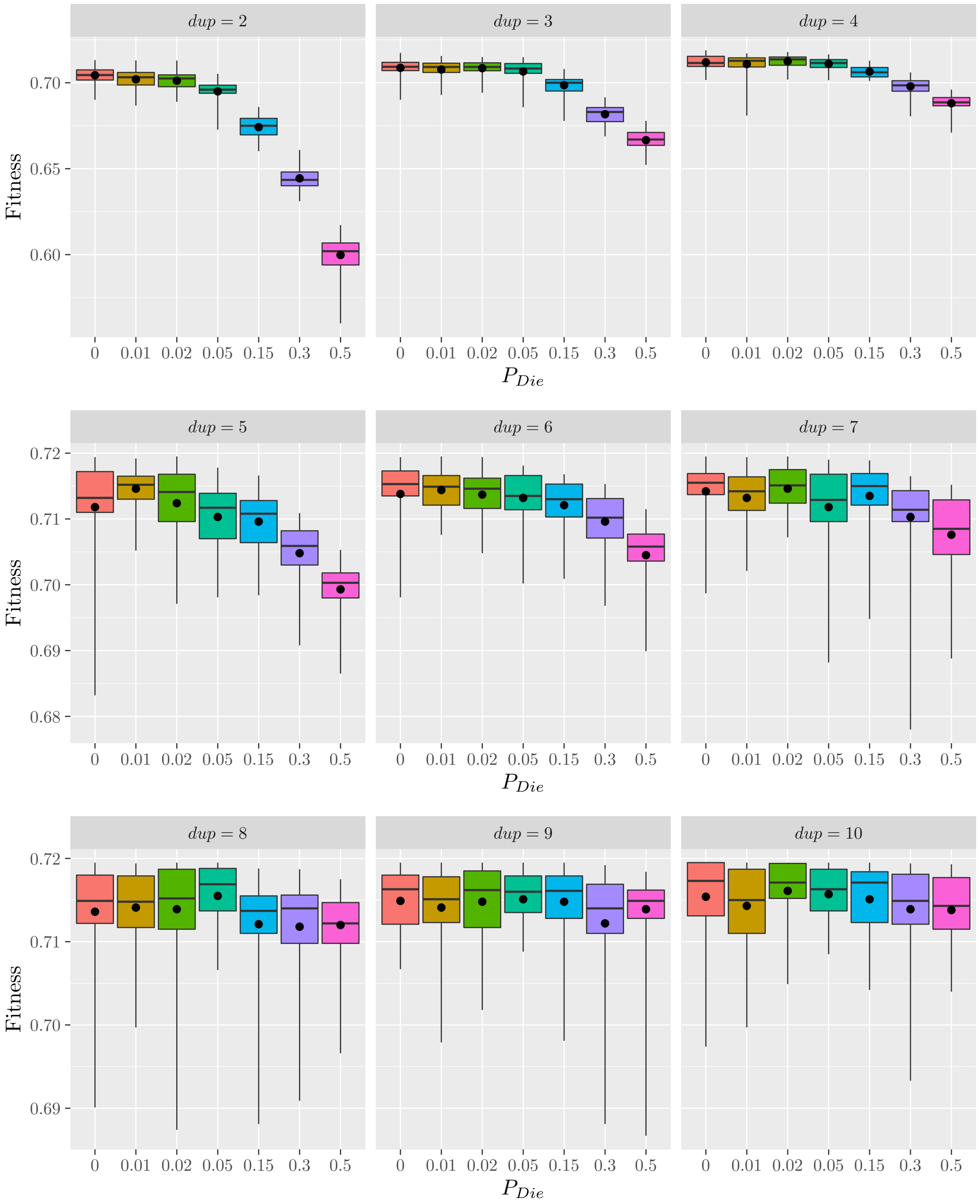


Figure 5.3: Results of tuning parameters with $M = 1$ on the *Cattle PPI* network.

network, we took into consideration both values, because their effect is also related to the network density (see results in Section 5.3).

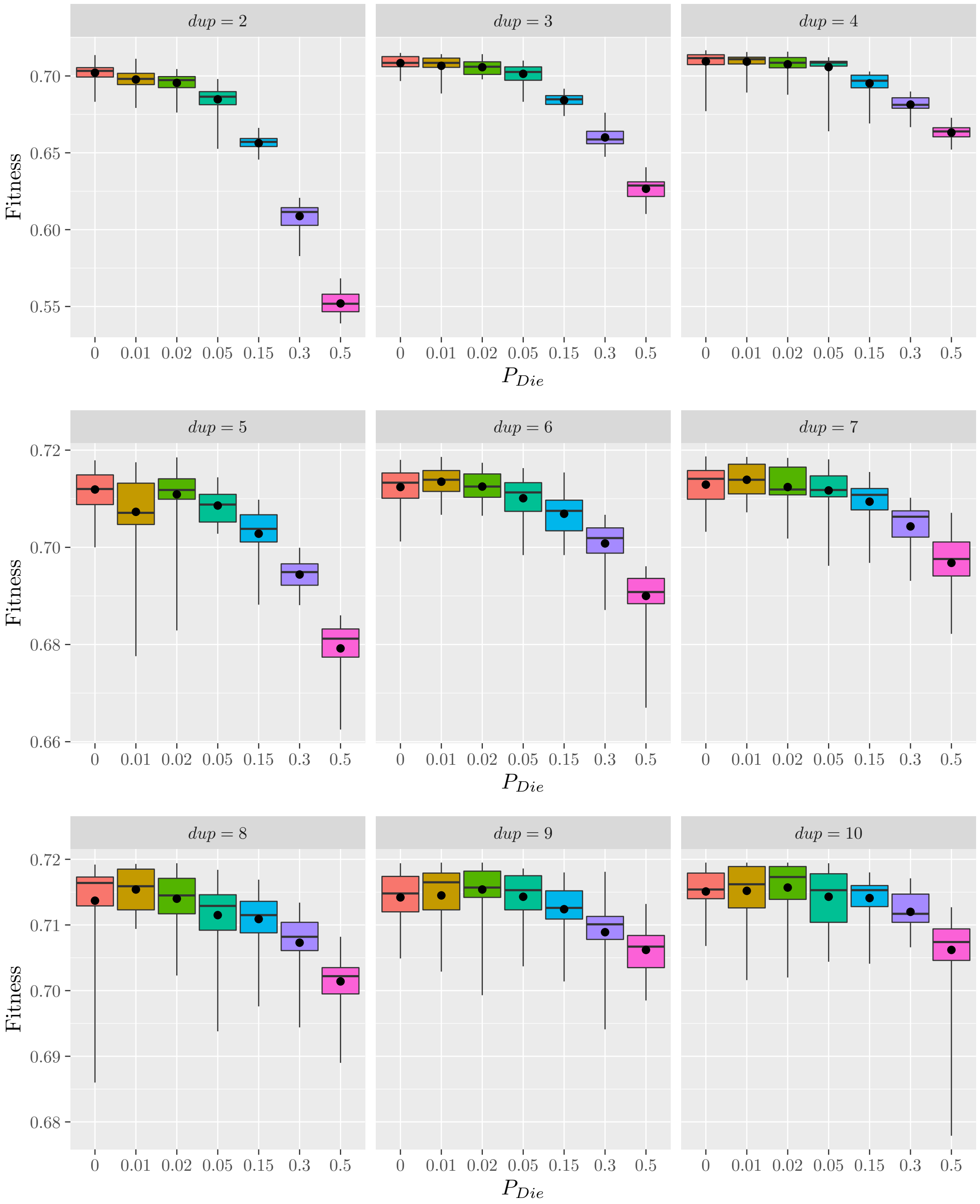
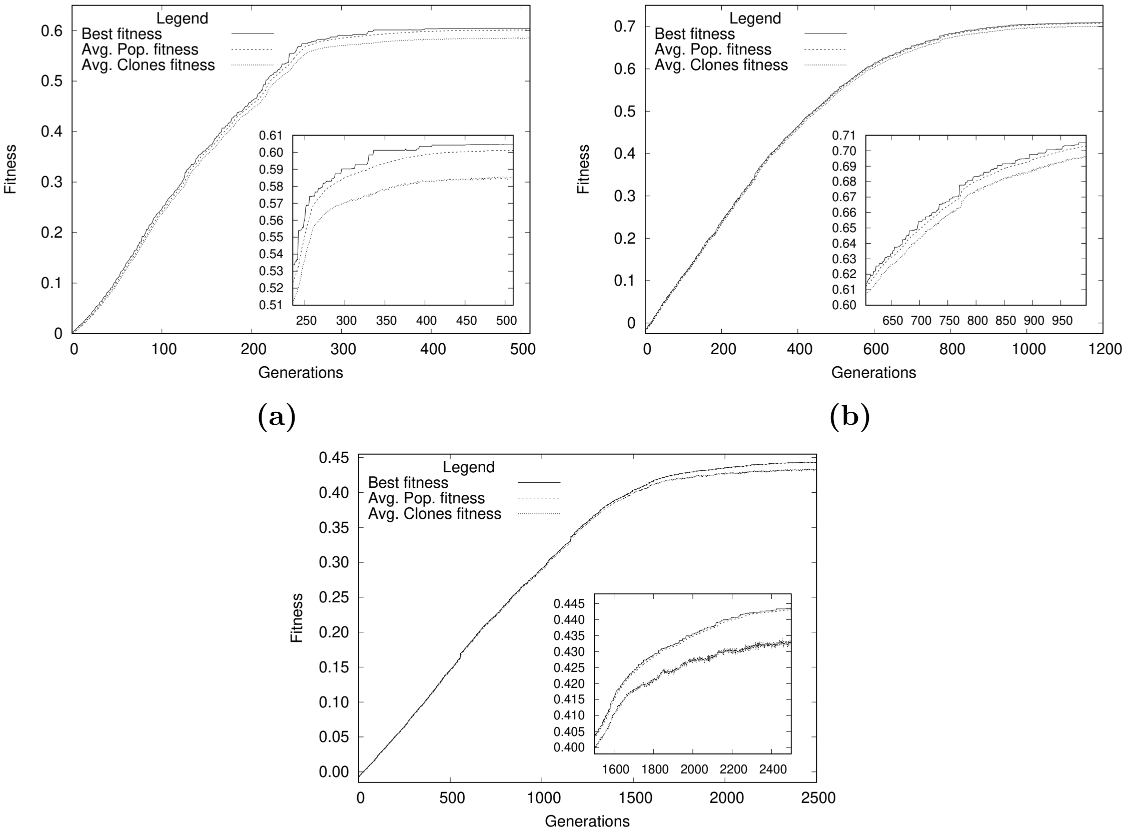


Figure 5.4: Results of tuning parameters with $M = 2$ on the *Cattle PPI* network.

5.2.3 Convergence Behaviour

A right convergence behaviour together with a good learning ability is the key factor for any successful stochastic search algorithm. Thus, we conducted a deep analysis of the dynamic behaviour of Opt-IA as reported in this section. We used the networks

American College Football, *Cattle PPI* and *C. elegans MRN* because, being different in types, sizes, density (Δ in Table 5.1), and mainly complexity, they allow a more robust analysis. As described above (Section 5.2.1), all these experiments were averaged over 30 independent runs.



(a)

(b)

(c)

Figure 5.5: Convergence behaviour of Opt-IA on (a) *American College Football*, (b) *Cattle PPI* and (c) *C. elegans MRN* networks. The figures show the average fitness function value and the best solution versus generations.

Figure 5.5 shows the convergence curves of Opt-IA, and, in particular, the best fitness, average fitness of the population, and average fitness of the hypermutated population. In particular, one can see how in all three plots Opt-IA shows a very good convergence towards the optimal solution. Indeed, the three curves grow slowly and improve step by step until they reach the best solution. It is important to note that initially the three curves are very close, and then begin to differentiate as they approach the optimal solution (see inset plots). This is due to the diversification of the solutions whose crucial impact happens mainly when the improvements are limited, and consequently when Opt-IA needs to get out of local optima.

Simply put, all three curves keep a right distance from each other, confirming the existence of a good degree of diversity among the solutions, which is useful for avoiding and/or escaping from local optima. Furthermore, it is also important to highlight that the curve of the best fitness does not increase monotonically: for some generations, the curve decreases slightly, and this corresponds to the discovery of better fitness values, right in the next generations.

Having analysed the convergence behaviour, we performed a study on the learning ability of Opt-IA. It is clearly very important to understand how much information the algorithm is able to discover during the evolutionary process since it plays a crucial role in the overall performance. To this end, we used the classical entropy function information gain, known also as Kullback–Leibler divergence, to measure the quantity of information the algorithm gains during the learning phase [89, 90]. Shannon's entropy [128] is the classical measure used in Information Theory and it represents a good measure of randomness or uncertainty, where the entropy of a random variable is defined in terms of its probability distribution. It is then used to measure the flatness of the information distribution provided by a set of solutions. The Kullback-Leibler divergence [89, 90] is the most frequently used information-theoretic distance measure and indicates how different two probability distributions are.

Let $B_m^{(t)}$ be the number B cells that at the timestep t have fitness value m. The candidate solutions distribution function $f_m^{(t)}$ is defined as the ratio of $B_m^{(t)}$ on the total solutions number d, that is:

$$f_m^{(t)} = \frac{B_m^{(t)}}{\sum_m B_m^{(t)}} = \frac{B_m^{(t)}}{d}. \tag{5.1}$$

Therefore, the information gain, labelled $K(t, t_0)$, and entropy, labelled $E(t)$, can be defined as:

$$K(t, t_0) = \sum_m f_m^{(t)} \log \left(\frac{f_m^{(t)}}{f_m^{(t_0)}} \right), \tag{5.2}$$

$$E(t) = -\sum_{m} f_m^{(t)} \log f_m^{(t)}, \tag{5.3}$$

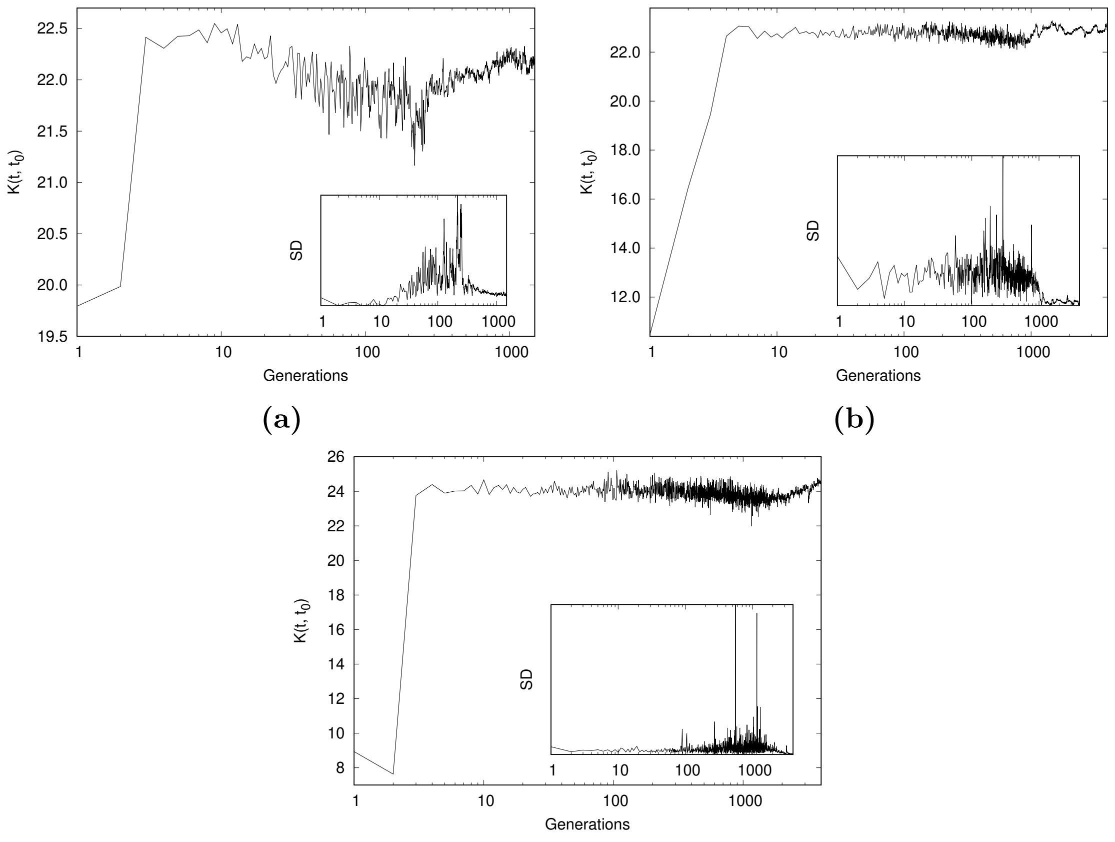
where, $K(t, t_0)$ is the quantity of information the system discovers during the convergence process. The gain is, then, the amount of information the system learned compared to the initial population $P^{(t=0)}$ which was randomly generated. Once the learning and search process begins, the information gain monotonically increases until it reaches a peak point after which it continues in a (roughly) steady state, consistently with the *maximum information-gain principle* [82]:

$$\frac{dK}{dt} \geq 0. \tag{5.4}$$

Overall, the information gain is a useful entropy function for understanding the behaviour of algorithms both *on-line* and at *run-time*, and for performing an accurate parameter tuning.

As for the convergence analysis, the same three networks were also used to evaluate the learning ability of Opt-IA. Figure 5.6 displays the information gain curves, respectively, on the *American College Football* in Figure 5.6a, *Cattle PPI* in Figure 5.6b and *C. elegans MRN* in Figure 5.6c. Inspecting all three plots, we can clearly see how Opt-IA quickly gains enough information during the first iterations, reaching regions of search space with a good average, which proves the efficiency of the mutation operators designed to explore the search space. After that, due to the fully random search process and without any guided search specific to the problem, the learning process alternates in gaining or losing information, until it reaches the highest peak that corresponds to having reached the best overall solution. The insert plot shows the relative standard deviations.

In Figure 5.6a it is possible to see the learning behaviour of Opt-IA on the *American College Football* network. In this plot, the algorithm shows a different learning behaviour compared to the other two considered networks, because this network is a little simpler in size and network density. Indeed, once the highest peak is reached (in the generations range $[20, 30]$), Opt-IA begins to lose information until around



(a)

(b)

(c)

Figure 5.6: Information gain curves of Opt-IA on (a) *American College Football*, (b) *Cattle PPI* and (c) *C. elegans MRN* networks. The inset plots show the relative standard deviations.

the 200th iteration when the curve begins to increase again, and therefore it starts to gain again information. In the inset plot, we show the relative deviation standard (σ) of Opt-IA, which measures the amount of dispersion (uncertainty) inside the population. It is interesting to note, indeed, that the iteration point where the algorithm reaches the maximum information gain corresponds exactly to the standard deviation lowest point. Correctly, then, the maximum information gain corresponds to minimum uncertainty. Similarly, at the point of the lowest information gain, reached before 200th generation, there is the highest standard deviation value. The information gain curves displayed in Figures 5.6b and 5.6c show, instead, a steadier state behaviour once the higher information value is reached. Interestingly, we can see in both plots that after 1000 generations Opt-IA begins to discover new information, and particularly in Figure 5.6b it reaches even the highest information gain value. This is consistent with the standard deviation curves, which reach their lowest values just after the 1000 generations.

In conclusion, both analyses (convergence and learning) prove the efficiency and robustness of Opt-IA in the community detection task. More importantly, they highlight and prove how Opt-IA needs more iterations to discover high-quality solutions due to the strong randomness present in the developed operators.

5.2.4 Computational Time Complexity

The running time of Opt-IA for reaching the best solution is another crucial measure to take into account for proving the efficiency of the proposed immune algorithm. We used the *time-to-target* (TTT) plots [1, 59] which are a standard graphical methodology for data analysis and for characterizing the running time of stochastic algorithms in order to solve a specific optimization problem. They measure the CPU times to find the target of the problem instance tackled. The basic idea behind TTT-plots is to compare the empirical and theoretical distributions, i.e. it displays the probability that an algorithm will find a solution as good as a target within a given running time. A Perl program has been proposed by Aiex et al. in [2] for automatically generating the TTT plots, which produces two different plots: a theoretical quantile-quantile (QQ) plot with superimposed variability information, and a superimposed empirical and theoretical distributions[2].

In order to perform such an analysis, the Opt-IA algorithm is run n times on a given instance using the achieving of a target value (i.e. achieve global optimum) as a stopping criterion. Obviously, for every single run, a different seed is considered for the random number generator to have independent runs. Note that the larger the number n considered, the closer the empirical distribution will be to the theoretical one. Therefore, following the suggestions given in [2], we set $n = 200$ because it has been proven that this value gives very good approximations of the theoretical

[2] The Perl program can be downloaded at http://mauricio.resende.info/tttplots/tttplots.zip.

distributions. This analysis has been conducted on six different networks in size and complexity: *Grevy's Zebras*, *Zachary's Karate Club*, *Bottlenose Dolphins*, *Books about US Politics*, and *C. elegans MRN* and *H. pylori PPI*. For a proper analysis, it is important to consider not easy instances, since the exponential distribution would degenerate to a step function, due to the very small CPU times in almost all runs, as asserted in [1]. Furthermore, these networks have been considered also because the new stopping criterion requires that in the tackled networks/instances the success rate is 100%.

In Figures 5.7-5.14 the TTT plots produced on the cited networks are shown. In each figure, the left plot shows the empirical versus theoretical distribution, whilst in right plots show the QQ plots with variability information.

Overall, by inspecting all plots for social networks in Figures 5.7-5.10, it emerges how both empirical curves perfectly fit the theoretical ones for the first three networks, whilst the empirical curves of Opt-IA slightly differ from the theoretical ones for the *Books about US Politics* network. It is important to point out that the TTT plots experiments on the *Books about US Politics* network were performed considering the best solution found (0.5272) as target value for the stopping criterion, and Opt-IA was able to find it in all 200 runs, although in Table 5.4 (see Section 5.3) the *best* and *mean* values are not the same. This confirms that with a larger number of iterations the developed search process is able to discover even better solutions until it reaches the optimal ones, of course, with higher computational complexity time. However, from the relative TTT plots in Figure 5.9, the empirical curve follows the same behaviour as the theoretical one, proving consequently the efficacy of Opt-IA on this network.

For the *C. elegans MRN* and *H. pylori PPI* networks - two of the larger networks in the dataset - two different target values were, instead, considered as stopping criteria, since Opt-IA found better modularity than the compared algorithms (see Table 5.8), either as *best*, *mean* and *worst* values. For *C. elegans MRN* the first experiment was then conducted considering 0.4185 as target value (Figure 5.11), which corresponds to the best modularity found among all compared algorithms, and specifically by HDSA.

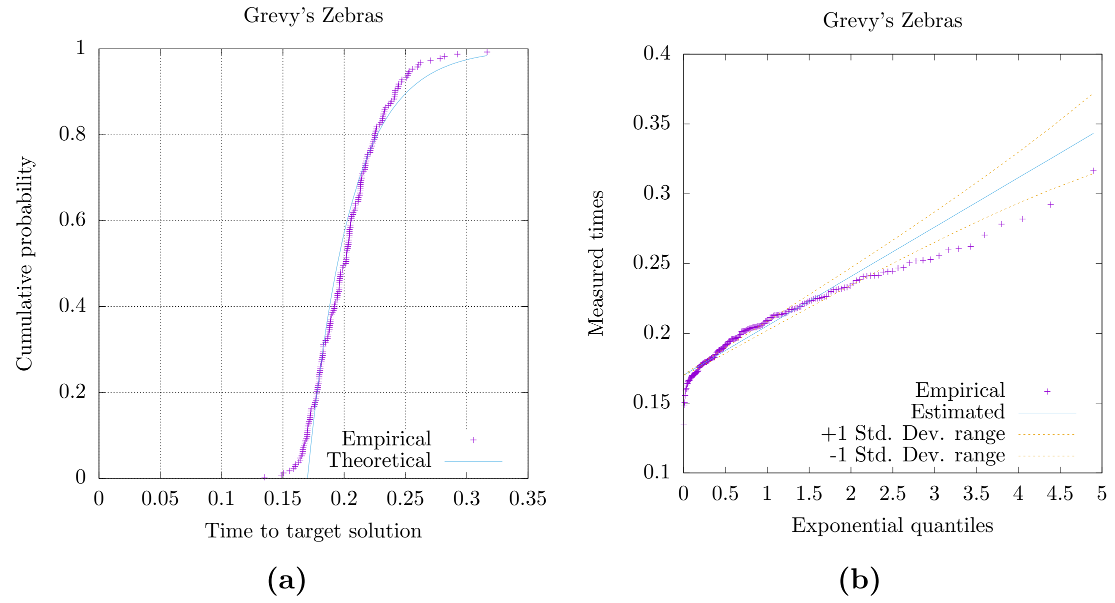


Figure 5.7: Time-to-target plots for *Grevy's Zebras* network with target value $t = 0.2768$. (a) Empirical versus theoretical distributions and (b) QQ plot with variability information.

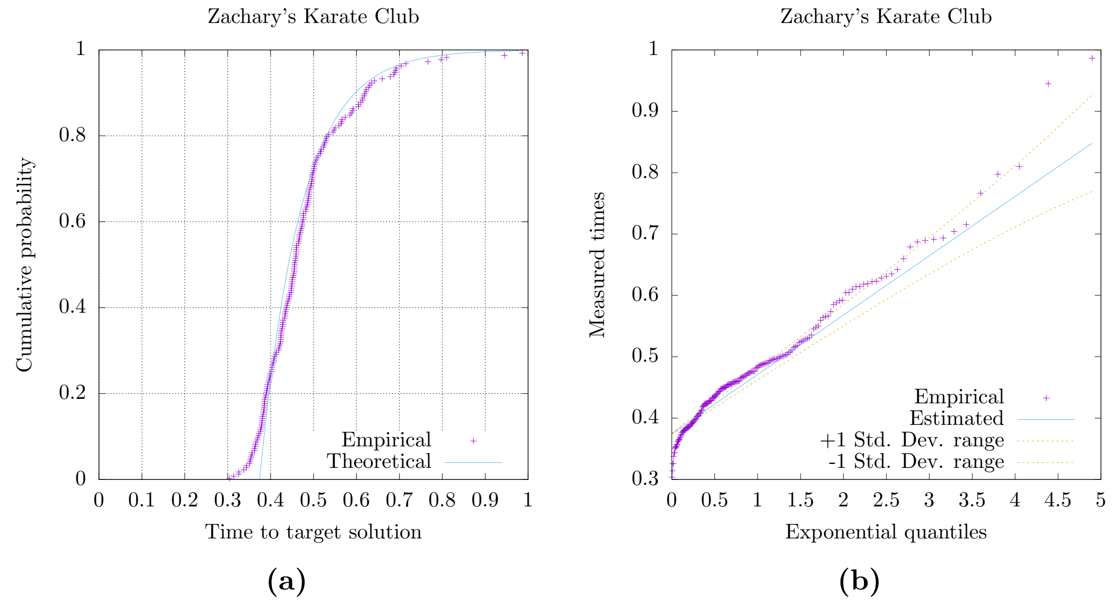
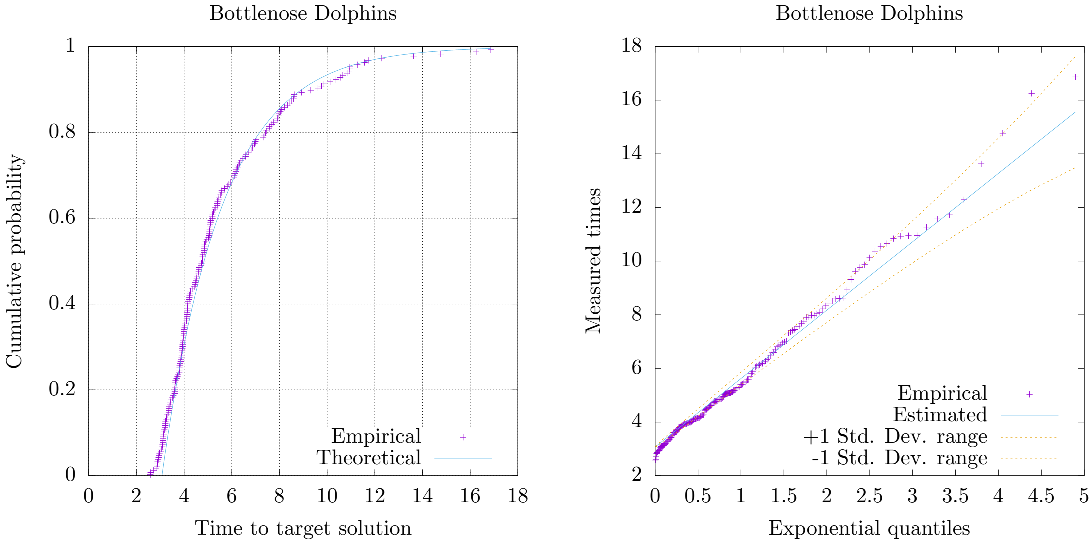
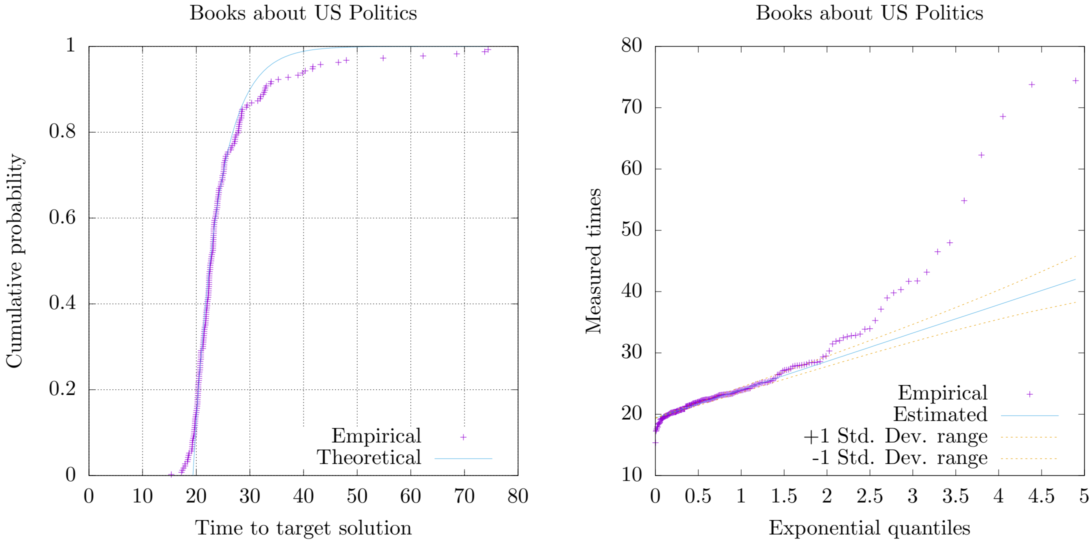


Figure 5.8: Time-to-target plots for *Zachary's Karate Club* network with target value $t = 0.4198$. (a) Empirical versus theoretical distributions and (b) QQ plot with variability information.



(a) **(b)**

Figure 5.9: Time-to-target plots for *Bottlenose Dolphins* network with target value $t = 0.5285$. (a) Empirical versus theoretical distributions and (b) QQ plot with variability information.



(a) **(b)**

Figure 5.10: Time-to-target plots for *Books about US Politics* network with target value $t = 0.5272$. (a) Empirical versus theoretical distributions and (b) QQ plot with variability information.

Moreover, because the worst modularity computed by Opt-IA is still better than the one found by HDSA, a second TTT plots experiment was performed setting the stopping target to 0.4221 (Figure 5.12), i.e. the worst solution of Opt-IA on such a network (see Table 5.8).

The same experimental protocol was used for the second biological network, *H. pylori PPI*. The first experiment was conducted considering 0.5086 as target value (Figure 5.13), which corresponds to the best modularity found by HDSA, while the second experiment was performed setting the target solution to 0.5116 (Figure 5.14).

Focusing the analysis only on the plots in Figures 5.11-5.14, that is the two different targets considered for the *C. elegans MRN* and *H. pylori PPI* networks, it appears clear how Opt-IA easily achieves the same maximum modularity of HDSA, whilst (obviously) needing more time to reach larger values of modularity. However, the empirical curve fits perfectly with the theoretical one.

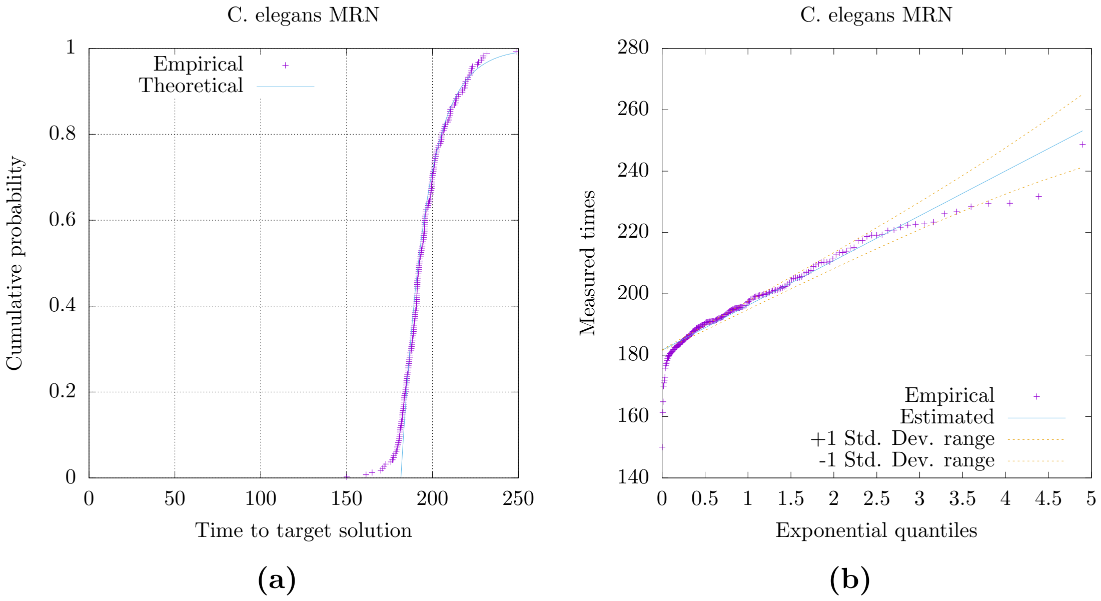


Figure 5.11: Time-to-target plots for *C. elegans MRN* network with target value $t = 0.4185$. (a) Empirical versus theoretical distributions and (b) QQ plot with variability information.

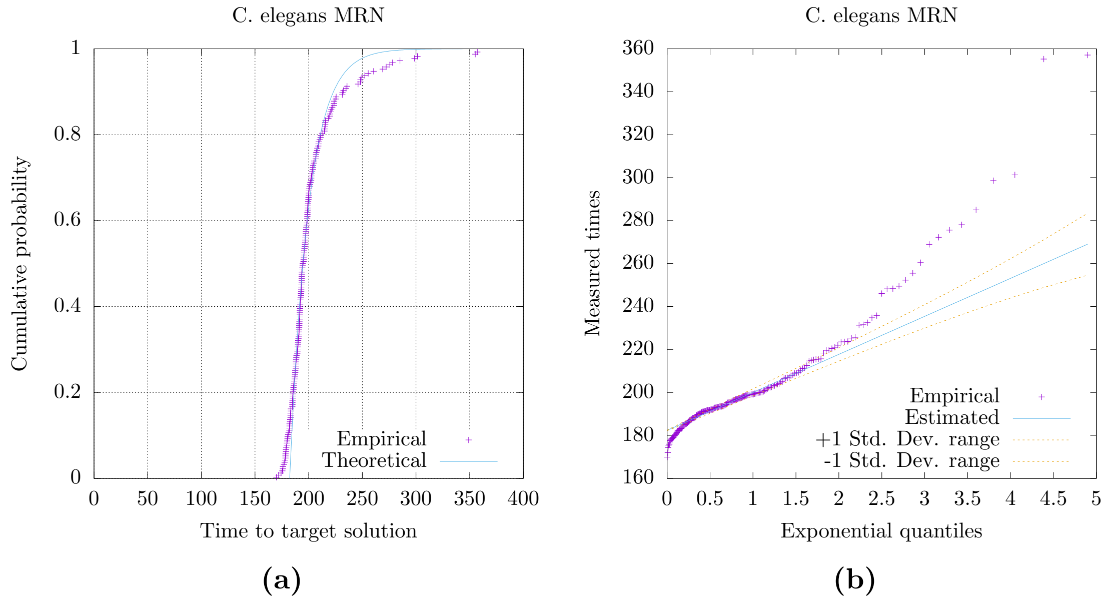


Figure 5.12: Time-to-target plots for *C. elegans MRN* network with target value $t = 0.4239$. (a) Empirical versus theoretical distributions and (b) QQ plot with variability information.

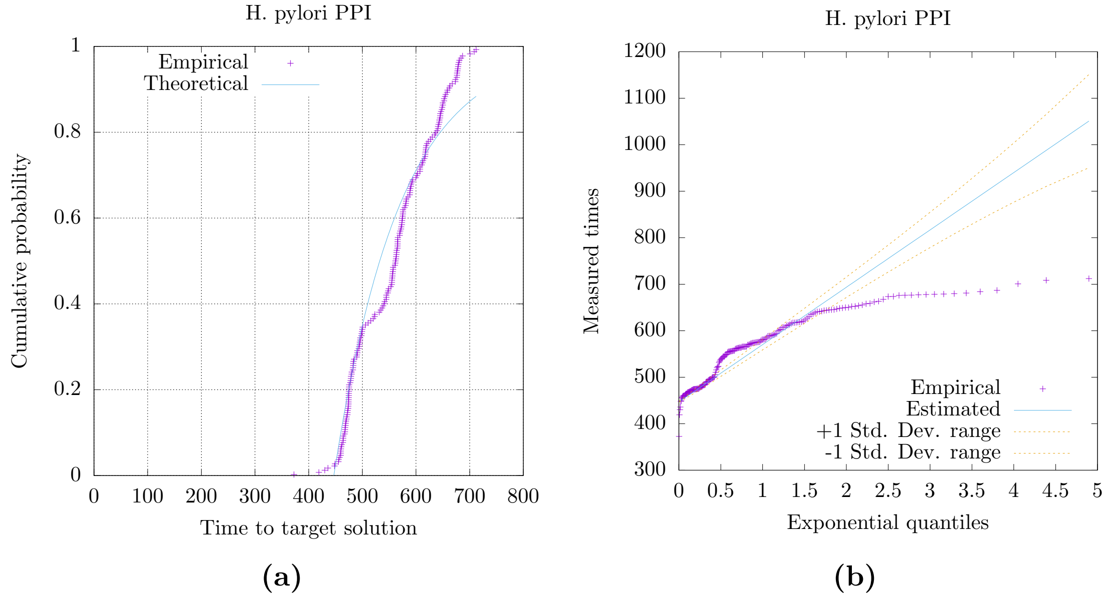
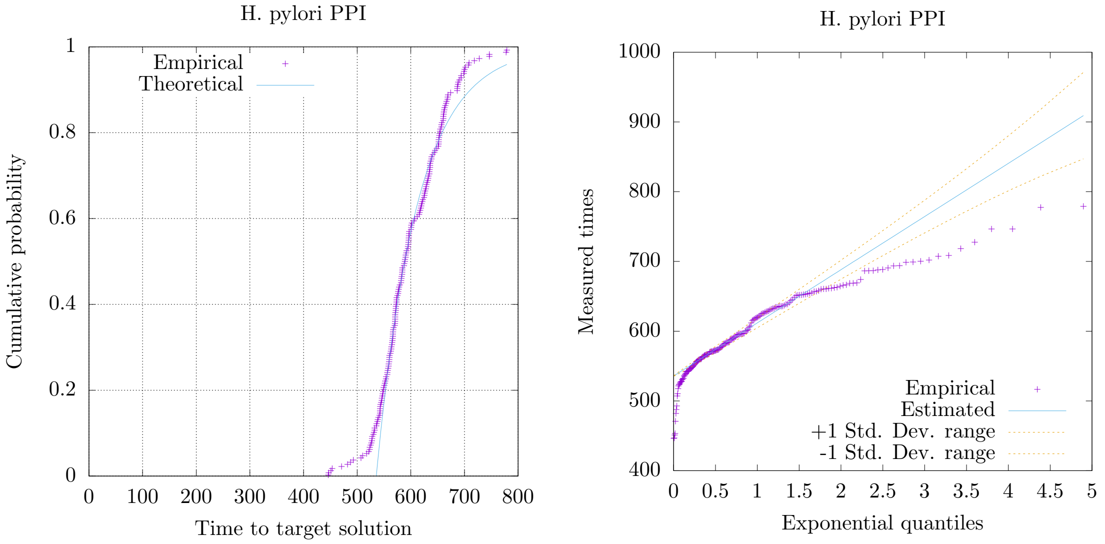


Figure 5.13: Time-to-target plots for *H. pylori PPI* network with target value $t = 0.5086$. (a) Empirical versus theoretical distributions and (b) QQ plot with variability information.



(a) **(b)**

Figure 5.14: Time-to-target plots for *H. pylori PPI* network with target value $t = 0.5116$. (a) Empirical versus theoretical distributions and (b) QQ plot with variability information.

5.2.5 Precompetition Operator Effectiveness

The precompetition operator, in addition to the aging operator, plays a key role in the performances of Opt-IA since it allows the algorithm to jump away from local optima by introducing heterogeneity in the population. This, of course, is a crucial characteristic, especially when addressing a hard and complex problem. Although the usefulness and efficiency of the aging operator are well known [80, 81], little instead is possible to assert on the efficacy of the precompetition operator, and how it affects the performance of Opt-IA. In light of this, in this section, an analysis of the overall effectiveness of the precompetition operator is presented, and it is shown in Table 5.2. For this analysis four biological networks were considered (*Cattle PPI*, *E. coli TRN*, *C. elegans MRN* and *H. pylori PPI*), and used for inspecting the convergence behaviour of Opt-IA, by enabling or disabling such an operator.

Looking at the outcomes reported in the table, the usefulness and efficacy of the precompetition operator are clearly evident: it allows Opt-IA to reach better modularity values not only with respect to the maximum value found but also with respect

Table 5.2: Experimental results of Opt-IA with and without the *precompetition* operator on biological networks.

		Opt-IA	
Network		*Precompetition*	$\neg$*Precompetition*
Cattle PPI	*Best*	**0.7195**	0.7195
	Mean	0.7156	0.7151
	Worst	0.7061	0.7053
	StD	0.0036	0.0044
	N_C	40	40
E. coli TRN	*Best*	**0.7796**	0.7776
	Mean	0.7711	0.7701
	Worst	0.7578	0.7525
	StD	0.0051	0.0055
	N_C	39	39
C. elegans MRN	*Best*	**0.4490**	0.4470
	Mean	0.4369	0.4340
	Worst	0.4239	0.4069
	StD	0.0053	0.0081
	N_C	8	9
H. pylori PPI	*Best*	**0.5416**	0.5329
	Mean	0.5249	0.5235
	Worst	0.5116	0.5133
	StD	0.0063	0.0055
	N_C	19	19

to the mean of the best found values in all independent runs, with the consequence of allowing the algorithm to obtain lower standard deviation values. It is important to highlight that, except for the Cattle PPI network where the best modularity is the same for both versions, the precompetition operator allows Opt-IA to produce considerably higher modularity values, proving the successful effect of this operator. The precompetition operator, in combination with the stochastic aging, compensates for the indirect elitism provided by the selection operator, therefore it helps to maintain the right balance of diversity in the population.

5.3 Experimental Results

We discuss now the overall experimental results and compare them with the results obtained by state-of-the-art algorithms. It is important to stress first that, in a preliminary work [132], Opt-IA was compared to Louvain algorithm on a set of different, simple and small networks, which for simplicity are reported in Table 5.3. By inspecting this table, it becomes clear how Opt-IA, based on a pure random-search, outperforms one of the best approaches on community detection with modularity optimization, which is Louvain algorithm[3]. However, given the low complexity of these tested networks, a deep and detailed analysis must be conducted in order to evaluate the real performance of Opt-IA. Therefore, for these new experiments, all networks in the data set in Table 5.1 were considered, and the experimental protocol described in Section 5.2.1 was used. The main goal of these experiments, as well as all comparisons made, is to prove the competitiveness and reliability of Opt-IA in terms of solution quality found, i.e. maximizing the modularity function (Equation 4.1).

Table 5.3: Comparative results of Opt-IA and Louvain algorithm.

		Louvain		Opt-IA	
Network	$\lvert V \rvert$	Q	N_C	Q	N_C
Zachary's Karate Club	34	0.4156	4	**0.4198**	4
Bottlenose Dolphins	62	0.5188	5	**0.5285**	5
UK Faculty	81	**0.4488**	4	**0.4488**	4
Huckleberry	69	**0.5346**	4	**0.5346**	4
Les Miserables	77	0.5583	6	**0.5600**	6
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

To this end, the proposed Opt-IA algorithm was compared to several different heuristics and metaheuristics (13 in the overall), each of them designed and developed

[3] The results of the Louvain algorithm reported in the following tables were obtained using the implementation of the *igraph* [40] library, version 1.2.8. In this version, the vertices are always processed in the same order resulting in the very same partition.

as a modularity optimization approach. Specifically, the first group of algorithms considered for the comparisons, in addition to Louvain, are:

- Bat Algorithm (BA) [4], a metaheuristic method based on the echolocation behaviour of bats [150];

- Gravitational Search Algorithm (GSA) [4], a metaheuristic algorithm based on the law of gravity and mass interactions [117];

- Big Bang–Big Crunch (BB-BC) algorithm [4], an algorithm inspired by the theories of the universe evolution in which, during the main phase, energy dissipation produces disorder and randomness, whilst in a second stage the randomly distributed particles are drawn into an order, i.e. the values in the vectors of the function to be optimized are determined [58];

- Bat Algorithm based on Differential Evolutionary (BADE) [4], an improved version based on the combination (hybridization) of Bat Algorithm and Differential Evolution (DE) algorithm [134, 133], where this latter is used in the population regeneration process. Both algorithms are used together for the selection of adjacent vertices;

- Scatter Search algorithm based on Genetic Algorithm (SSGA) [4], a Scatter Search (SS) approach [68, 100] of the best chromosomes provided by Genetic Algorithm (GA) [77, 70] and subjecting the population to the crossover and the mutation processes around the best solutions;

- Hyper-heuristic Differential Search Algorithm (HDSA) [4], a hyper-heuristic based on the migration of artificial super-organisms, where each of them in the population migrates between the maximum or minimum solution of the problem using the Differential Search Algorithm (DSA) [33] in the process of regeneration of individuals.

Moreover, the second group of algorithms considered for the comparisons are:

- MA-Net [104], a memetic algorithm based on the combination of a genetic algorithm with a local search;
- GACD [130], a genetic algorithm that takes advantage of the efficiency of the locus-based adjacency encoding scheme to represent a community partition;
- Clustering Coefficient-based Genetic Algorithm (CC-GA) [121], a genetic algorithm that uses the clustering coefficient (CC), which is a social networks analysis measure, to generate a better initial population;
- Multi-Start Iterated Greedy (MSIG) algorithm [122], which uses a new greedy procedure for generating the initial solutions and reconstructing the solutions, but has the disadvantage of being computationally expensive;
- Improved Discrete Particle Swarm Optimization with Redefined Operator (ID-PSO-RO) [27], based on particle swarm optimization, in which the update formulas of velocity and position are redefined according to the locus-based adjacency representation;
- Iterated Greedy (IG) algorithm [96] based on an iterative process that combines a destruction phase and a reconstruction phase: a complete candidate solution is partially destructed, and afterwards a new complete candidate solution is reconstructed via a greedy constructive heuristic.

The results reported in the following tables were taken from [4, 104, 96].

Following what was previously described, the parameters setting of Opt-IA, in all the performed experiments, are $d = 100$ as population size; $P_{die} = 0.02$ the probability of random aging operator; mutation rate M set to 1; and $T_{Max} = 4000$ as the maximum number of generations used as stopping criterion. The duplication parameter dup in according to the parameters tuning reported in Section 5.2.2, has been set to 4 ($dup = 4$) for all those instances with $|V| < 100$ (small social networks), whereas for the larger ones ($|V| \geq 100$) the experiments were performed with $dup = 9$ and $dup = 10$. Every experiment has been performed on 30 independent runs.

Table 5.4: Experimental results of Opt-IA on small social networks with $dup = 4$.

Network	*Best*	*Mean*	*Worst*	*StD*	N_C
Grevy's Zebras	0.2768	0.2768	0.2768	0.0000	4
Zachary's Karate Club	0.4198	0.4198	0.4198	0.0000	4
Bottlenose Dolphins	0.5285	0.5285	0.5285	0.0000	5

Table 5.5: Experimental results of Opt-IA on social and biological networks with $dup = \{9, 10\}$.

Network	*dup*	*Best*	*Mean*	*Worst*	*StD*	N_C
Books about US Politics	9	0.5272	**0.5270**	0.5208	0.0012	5
	10	0.5272	0.5268	0.5208	0.0016	5
American College Football	9	0.6046	**0.6011**	0.5848	0.0052	10
	10	0.6046	0.5999	0.5891	0.0050	10
Cattle PPI	9	0.7195	0.7148	0.7018	0.0044	40
	10	0.7195	**0.7161**	0.7049	0.0039	40
E. coli TRN	9	0.7734	0.7660	0.7486	0.0050	27
	10	**0.7795**	0.7670	0.7589	0.0049	32
C. elegans MRN	9	**0.4487**	0.4366	0.4221	0.0070	8
	10	0.4464	0.4377	0.4231	0.0061	10

The obtained outcomes are summarized in Tables 5.4 and 5.5. Table 5.4 reports the results of the proposed algorithm on small social networks, while in Table 5.5 we show the results on larger social networks and biological networks. In this last table, the best results obtained for $dup = 9$ or $dup = 10$ are also highlighted in boldface. The different setting of the duplication parameter is obviously due to the simplicity of the first networks ($dup = 4$) compared to the last ones ($dup = 9$ or $dup = 10$), which consequently require a more targeted search, and a less wide exploration of the solution space. Larger networks with a density $\Delta \geq 1\%$ (see Table 5.1), do not require a great variability in the population, and for this reason, $dup = 9$ seems to be the most appropriate value. Indeed, although in the social networks there is little difference in the results between the two *dup* values ($dup = 9$ vs. $dup = 10$), in *C. elegans MRN*, where $\Delta = 1.98\%$, a significant improvement is instead obtained in terms of best and average modularity found. On the other hand, for all networks with a low density ($\Delta < 1\%$) the parameter $dup = 10$ ensures good average values (*Mean*) in *Cattle PPI* instance and best modularity (*Best*) for *E. coli TRN* network. In these cases, a small increase in the *dup* parameter, i.e. having a larger number of duplicates, allows it to produce higher variability, and consequently enables it to work well on very sparse networks.

Tables 5.6-5.8 report the comparisons of Opt-IA with other heuristics and metaheuristics. The shown results are averaged on 30 independent runs for all algorithms. Note that, unlike other algorithms that use a maximum number of generations fixed, MA-Net stops running just only when 30 generations are performed without any improvement. For each table, the best modularity values (*Best*), average values (*Mean*), worst modularity (*Worst*), standard deviation (*StD*), and the number of communities discovered (N_C) are showed, respectively.

Table 5.6: Comparative results of Opt-IA and algorithms of the first group on social networks.

		Algorithms							
Network		Opt-IA	Louvain	HDSA	BADE	SSGA	BB-BC	BA	GSA
Grevy's Zebras	*Best*	0.2768	0.2768	0.2768	0.2768	0.2768	0.2768	0.2768	0.2768
	Mean	0.2768	-	0.2768	0.2768	0.2768	0.2766	0.2768	0.2768
	Worst	0.2768	-	0.2768	0.2768	0.2768	0.2761	0.2768	0.2768
	StD	0.0000	-	0.0000	0.0000	0.0000	0.0003	0.0000	0.0000
	N_C	4	4	4	4	4	4	4	4
Zachary's Karate Club	*Best*	0.4198	0.4189	0.4198	0.4198	0.4198	0.4198	0.4198	0.4198
	Mean	0.4198	-	0.4198	0.4188	0.4198	0.4196	0.4133	0.4170
	Worst	0.4198	-	0.4198	0.4156	0.4198	0.4188	0.3946	0.4107
	StD	0.0000	-	0.0000	0.0018	0.0000	0.0004	0.0105	0.0037
	N_C	4	4	4	4	4	4	4	4
Bottlenose Dolphins	*Best*	0.5285	0.5285	0.5285	0.5268	0.5257	0.5220	0.5157	0.4891
	Mean	0.5285	-	0.5282	0.5129	0.5200	0.5141	0.4919	0.4677
	Worst	0.5285	-	0.5276	0.4940	0.5156	0.5049	0.4427	0.4517
	StD	0.0000	-	0.0005	0.0120	0.0040	0.0068	0.0289	0.0155
	N_C	5	5	5	4	5	5	4	6
Books about US Politics	*Best*	0.5272	0.5205	0.5272	0.5239	0.5221	0.4992	0.5211	0.4775
	Mean	0.5270	-	0.5272	0.5178	0.5203	0.4914	0.5020	0.4661
	Worst	0.5208	-	0.5272	0.5137	0.5167	0.4799	0.4815	0.4558
	StD	0.0012	-	0.0000	0.0042	0.0024	0.0084	0.0149	0.0079
	N_C	5	4	5	4	5	9	3	5
American College Football	*Best*	0.6046	0.6046	0.6046	0.5646	0.5330	0.5171	0.5523	0.4175
	Mean	0.6011	-	0.6033	0.5513	0.5277	0.5061	0.5272	0.4032
	Worst	0.5848	-	0.6019	0.5430	0.5189	0.4986	0.4742	0.3905
	StD	0.0052	-	0.0009	0.0085	0.0057	0.0069	0.0325	0.0109
	N_C	10	10	10	11	6	10	7	5

By analysing Table 5.6, it is clear how the proposed Opt-IA considerably outperforms all compared algorithms, except for HDSA. Regarding this latter, however, it is possible to note how both algorithms (Opt-IA and HDSA) show identical performances on the first two networks in the table reaching the same values of *Best* and *Mean*; whilst on the last two, HDSA outperforms Opt-IA only with respect the average values (both reach the same *Best* values). On the network *Bottlenose Dolphins*, instead, Opt-IA strictly outperforms HDSA reaching a better mean value, and a standard deviation value equal to zero. It is important to highlight that HDSA uses an initial population generated with a Genetic Algorithm and a Scatter Search algorithm. It follows obviously that this method is potentially more robust from a *Mean* value perspective. However, the difference between the values obtained by both heuristics, as *Best* and *Mean*, is almost irrelevant, demonstrating that the two algorithms Opt-IA and HDSA can be considered comparable in the overall.

Table 5.7: Comparative results of Opt-IA and algorithms of the second group on social networks.

		Algorithms						
Network		Opt-IA	GACD	CC-GA	MSIG	IDPSO-RO	IG	MA-Net
Zachary's Karate Club	*Best*	0.4198	0.4198	0.4198	0.4198	0.4198	0.4198	0.420
	Mean	0.4198	0.4198	0.4198	0.4198	0.4198	0.4198	0.419
	Worst	0.4198	0.4198	0.4198	0.4198	0.4198	0.4198	-
	StD	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.002
	N_C	4	-	-	-	-	-	4
Bottlenose Dolphins	*Best*	0.5285	0.5285	0.5285	0.5201	0.5285	0.5285	0.529
	Mean	0.5285	0.5272	0.5275	0.5189	0.5271	0.5268	0.523
	Worst	0.5285	-	-	-	-	-	-
	StD	0.0000	0.0020	0.0019	0.0017	0.0010	0.0014	0.004
	N_C	5	-	-	-	-	-	5
Books about US Politics	*Best*	0.5272	0.5272	0.52729	0.5232	0.5272	0.5269	0.527
	Mean	0.5270	0.5257	0.5271	0.5149	0.5261	0.5269	0.526
	Worst	0.5208	-	-	-	-	-	-
	StD	0.0012	0.0002	0.0002	0.0070	0.0021	0.0000	0.002
	N_C	5	-	-	-	-	-	5
American College Football	*Best*	0.6046	0.5879	0.5787	0.6033	0.6044	0.6046	0.605
	Mean	0.6011	0.5777	0.5640	0.5954	0.5900	0.6017	0.601
	Worst	0.5848	-	-	-	-	-	-
	StD	0.0052	0.0069	0.0093	0.0084	0.0129	0.0033	0.003
	N_C	10	-	-	-	-	-	10
Jazz Musicians	*Best*	0.4451	-	-	-	-	-	0.445
	Mean	0.4449	-	-	-	-	-	0.445
	Worst	0.4449	-	-	-	-	-	-
	StD	0.0001	-	-	-	-	-	0.000
	N_C	4	-	-	-	-	-	4

In Table 5.7 Opt-IA is compared with the second group of more recent metaheuristics methods. Also on this comparison, the proposed algorithm outperforms the compared algorithms in all networks. Indeed, if the comparison is inspected from a ranking perspective with respect to the *Best* values, Opt-IA is always at the top, whilst if it is analysed with respect to the *Mean* values, it is easy, instead, to assert that it is always among the first two positions and very often in the first one. It is worth emphasizing once again that, whilst these compared algorithms include deterministic and sophisticated strategies, Opt-IA is fully random both in the generation of the initial population and in the solutions search process into the search space. Therefore, having shown better performances, it confirms the robustness and efficiency of all designed random operators.

In Table 5.8, Opt-IA is compared with the first group of algorithms on biological networks. Unfortunately, no results were found by the other considered algorithms on these networks. Thus, inspecting this table, it is possible to see how Opt-IA strictly outperforms all algorithms, including HDSA, on the *C. elegans MRN* and *H. pylori PPI* networks compared to all evaluation metrics (*Best*, *Mean*, *Worst* and *StD*), and detecting a smaller community value. However, on the other two networks, Opt-IA and HDSA are comparable in *Cattle PPI* with respect to the best value reached, but Opt-IA is outperformed by HDSA with respect to the mean values. Also, on the *E. coli TRN* instance HDSA outperforms Opt-IA in all assessment values. It is important to highlight that Opt-IA performs better than HDSA on larger networks.

Finally, focusing the inspection only on the comparison between Opt-IA and Louvain it is easy to assert that the first considerably outperforms the latter, except for the *H. pylori PPI* network. Overall, then, analyzing all outcomes and comparisons performed, it is possible to assert that the proposed algorithm Opt-IA outperforms all the compared metaheuristics, and shows comparable performances with respect to hyper-heuristic HDSA. It is important to highlight that HDSA uses an initial population generated with a Genetic Algorithm and improved with a Scatter Search algorithm, Opt-IA, instead, is entirely blind to the features of the problem, and it is based only on random-search without any deterministic guide. Therefore, taking into account these main differences and features, and, primarily, having found results

Table 5.8: Comparative results of Opt-IA and algorithms of the first group on social networks.

		Algorithms							
Network		Opt-IA	Louvain	HDSA	BADE	SSGA	BB-BC	BA	GSA
Cattle PPI	*Best*	0.7195	0.7195	0.7195	0.7183	0.7118	0.7095	0.7143	0.7053
	Mean	0.7161	-	0.7195	0.7138	0.7079	0.7084	0.7100	0.6983
	Worst	0.7049	-	0.7194	0.7059	0.7052	0.7079	0.7063	0.6949
	StD	0.0039	-	0.0001	0.0051	0.0025	0.0007	0.0035	0.0041
	N_C	40	40	40	41	40	48	42	43
E. coli TRN	*Best*	0.7795	0.7786	0.7822	0.7680	0.7507	0.7520	0.7629	0.7416
	Mean	0.7670	-	0.7815	0.7621	0.7457	0.7485	0.7599	0.7375
	Worst	0.7589	-	0.7808	0.7560	0.7412	0.7452	0.7542	0.7328
	StD	0.0049	-	0.0006	0.0043	0.0035	0.0026	0.0034	0.0034
	N_C	32	40	47	58	61	71	56	61
C. elegans MRN	*Best*	0.4487	0.4263	0.4185	0.3473	0.3336	0.3374	0.3514	0.3063
	Mean	0.4366	-	0.4074	0.3385	0.3220	0.3266	0.3438	0.3039
	Worst	0.4221	-	0.3962	0.3335	0.3124	0.3194	0.3356	0.2974
	StD	0.0070	-	0.0010	0.0054	0.0077	0.0074	0.0073	0.0037
	N_C	8	9	13	25	22	21	22	24
H. pylori PPI	*Best*	0.5386	0.5450	0.5086	0.4926	0.4726	0.4681	0.4900	0.4600
	Mean	0.5204	-	0.5078	0.4854	0.4695	0.4660	0.4814	0.4567
	Worst	0.5065	-	0.5048	0.4809	0.4659	0.4642	0.4738	0.4549
	StD	0.0067	-	0.0017	0.0047	0.0021	0.0018	0.0073	0.0020
	N_C	17	23	52	69	70	75	62	77

comparable with those of HDSA, it is possible to confirm the efficiency and reliability of the proposed random-search algorithm Opt-IA.

In order to study Opt-IA on large networks a further set of networks was considered and tested, and the results are reported in Table 5.9. Of course, being Opt-IA fully based on random-search, for these experiments, a larger number of iterations was needed. During these experiments, increasing the network size, we saw that the combination of the developed operators guided the algorithm towards useless search, disregarding a proper and deep exploration of specific neighbourhoods. However, such behaviour did not happen on all previously tested networks. In light of this, to indirectly guide the search to explore promising regions in the search space more intensively, a simple modification in Opt-IA was made: allow the selection operator to also choose elements having the same fitness. This modified version, reported in Table 5.9, is labelled as Opt-IA$_{\text{FR}}$ (Opt-IA with Fitness Repetition), whilst the original one is called Opt-IA. Both versions are compared with the well-known Louvain algorithm. By

Table 5.9: Comparative results of two variant of Opt-IA and Louvain algorithm on larger biological networks.

Network	*Algorithm*	*Best*	*Mean*	*Worst*	*StD*	N_C
H. pylori PPI	Opt-IA	0.5386	0.5204	0.5065	0.0067	17
	Opt-IA$_{\text{FR}}$	0.5416	0.5249	0.5116	0.0063	19
	Louvain	0.5450	-	-	-	23
E. coli MRN	Opt-IA	0.3287	0.3141	0.3026	0.0074	31
	Opt-IA$_{\text{FR}}$	0.3629	0.3437	0.3282	0.0064	9
	Louvain	0.3569	-	-	-	9
S. cerevisiae PPI (1)	Opt-IA	0.6387	0.6145	0.5955	0.0100	257
	Opt-IA$_{\text{FR}}$	0.6516	0.6344	0.6178	0.0089	386
	Louvain	0.7638	-	-	-	216
S. cerevisiae PPI (2)	Opt-IA	0.4753	0.4606	0.4405	0.0082	288
	Opt-IA$_{\text{FR}}$	0.4879	0.4746	0.4473	0.0085	317
	Louvain	0.5905	-	-	-	46

comparing the two versions, it appears clear how such a simple change allows Opt-IA to improve the modularity values in the overall. At any rate, the results obtained by the best version of Opt-IA still remain a bit far from the results obtained by Louvain. This is explainable with the features of Opt-IA to be fully based on random search and without any simple deterministic approach. It is very likely that by further increasing the number of generations the gap with Louvain's results will be substantially narrowed. As expected, this is the main limitation of our proposed random-search algorithm.

In Figure 5.15, finally, are displayed the communities detected by Opt-IA on the *Books about US Politics* (Figure 5.15a), *American College Football* (Figure 5.15b) and *C. elegans MRN* (Figure 5.15c) networks, respectively.

5.3.1 Functional Sensitivity Analysis

Although modularity is the commonly used evaluation metric, it tells very little about how similar the detected communities are when compared to the original/target ones. Furthermore, an important limitation in modularity optimization is that it can fail in identifying smaller communities, due to the degree of interconnectivity of the com-

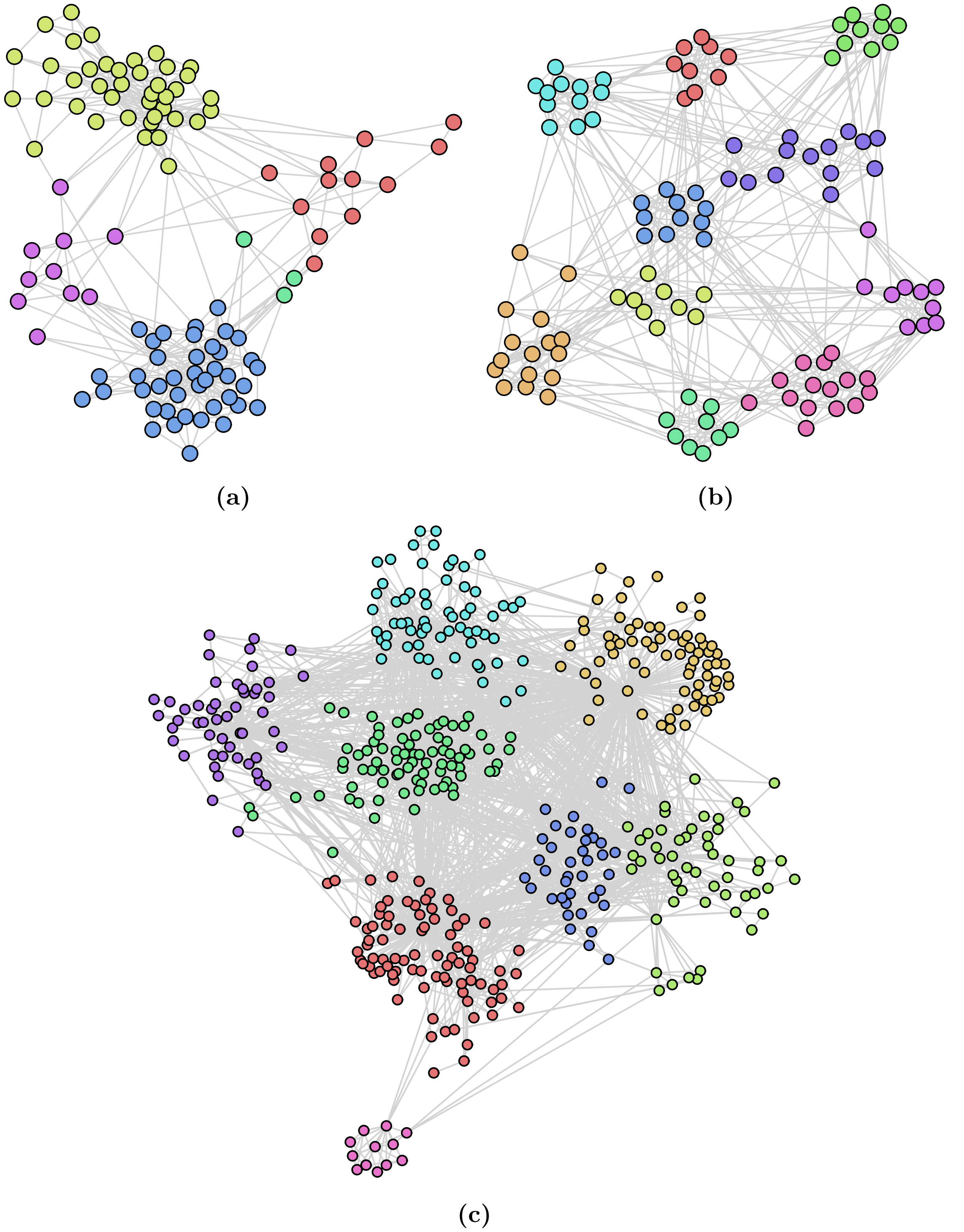


Figure 5.15: Community structures obtained by Opt-IA for (a) *Books about US Politics*, (b) *American College Football* and (c) *C. elegans MRN* networks.

munities [63]. To this end, we conducted a second experimental step, using synthetic networks generated by the LFR algorithm proposed in [92, 91]. The aims of this second experiment are to analyse the convergence behaviour of Opt-IA in different complexity scenarios, thanks to the diverse network features which can be generated, and, most importantly, by inspecting how good and similar are the communities uncovered by Opt-IA with respect to the target ones. Obviously, since all networks are artificially generated, their community structures are known. It is important to stress how this benchmark faithfully reproduces the key features of real graphs communities, affirming therefore its validation.

The networks generated for this experiment were respectively created with 300, 500, 1000, 2000, 3000 and 5000 vertices, each of them with average degree 15 and 20 or 20 and 25, and the maximum degree equal to 50. Furthermore, for each instance, we set: $\tau_1 = 2$ as the exponent of the degree distribution; $\tau_2 = 1$ as the distribution of community sizes; $min_c = 10$ and $max_c = 50$, respectively, as minimum and maximum of the communities' size. All experiments were conducted at the varying of the mixing parameter μ_t, which identifies the relationship between the vertex's external and internal degree with respect to its community: the greater the value of μ_t, the greater is the number of edges that a vertex shares with vertices outside of its communities. In order to analyse the performances of Opt-IA on several scenarios, the mixing parameter was made to vary in the range $\{0.1, 0.2, \cdots, 0.8\}$.

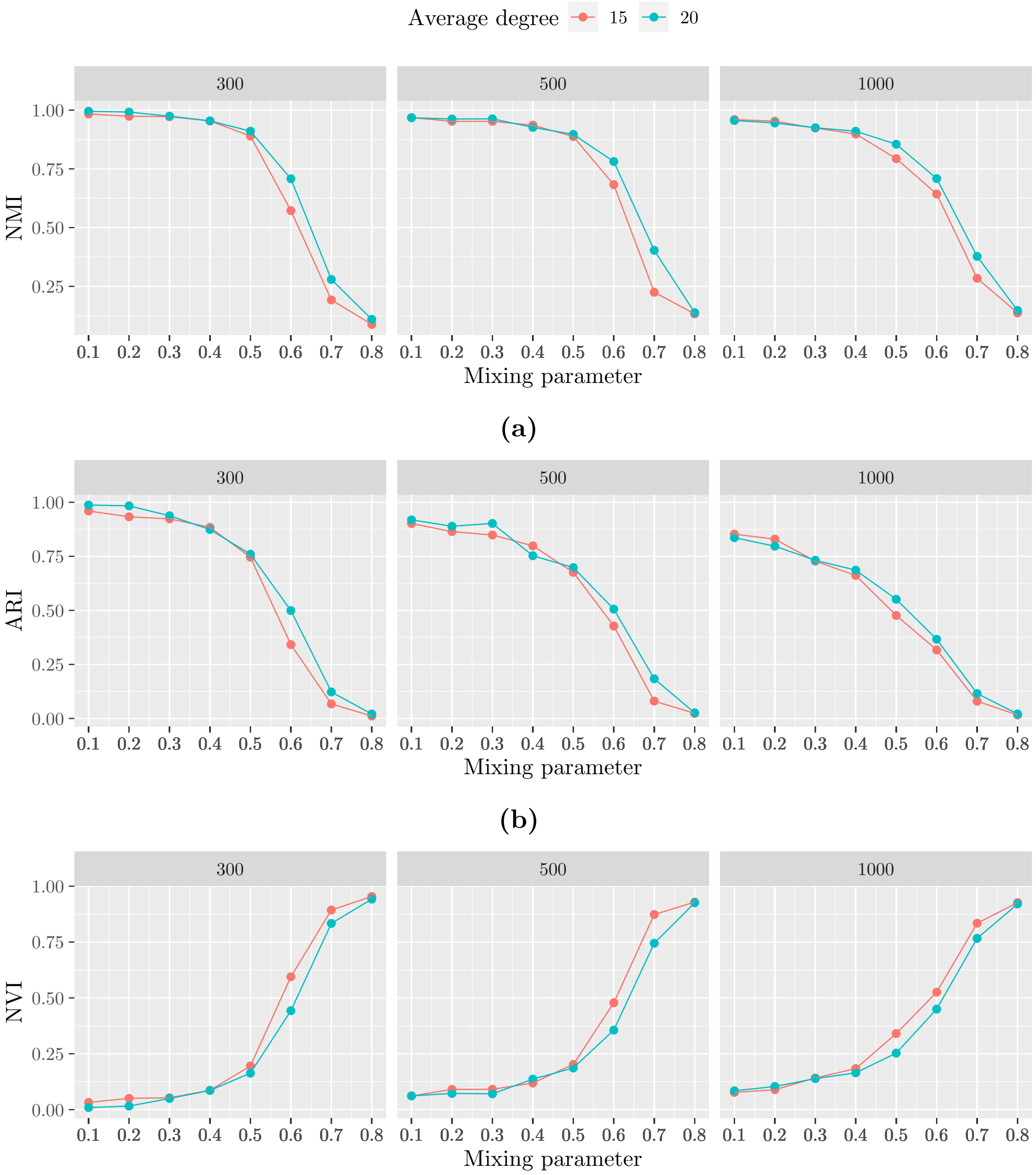
Once the synthetic networks were generated, each with different features, a functional sensitivity analysis was conducted using the well-known community structure similarity metrics, such as (1) *Normalized Mutual Information* (NMI) [48] that measures the amount of information correctly extracted, and allows for assessing how similar the detected communities are to real ones; (2) *Adjusted Rand Index* (ARI) [79], which focuses on the pairwise agreement, that is for each possible pair of elements it evaluates how similarly the two partitions treat them; and, finally, (3) *Normalized Variation of Information* (NVI) [101], 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. Note that NMI is the most used in community detection tasks. It is important also to point out that the closer to 1 the NMI and ARI values are (closer to 0 for the NVI value, instead), the more similar the uncovered communities are to the target ones.

In Figure 5.16 we can see the graphics of NMI, ARI and NVI indexes for the LFR benchmarks with 300, 500 and 1000 vertices. By analysing each plot, it is possible to note how the NMI and ARI curves remain on high values (> 0.70) for $\mu_t \leq 0.6$ and $\mu_t \leq 0.5$, respectively, whilst the NVI curve remains on low values for $\mu_t \leq 0.5$. This proves that Opt-IA is able to uncover communities roughly closer to the original ones. The two NMI and ARI curves instead begin to decrease, and the NVI curve increases, as the graph begins to get denser ($\mu_t > 0.6$); in this case, Opt-IA detects community structures not well-defined.

In Figure 5.17, instead, it is displayed the functional sensitivity analysis conducted on the synthetic networks with 2000, 3000 and 5000 vertices. By inspecting these plots, it is possible to assert that, for instances with 2000 vertices, the NMI curves in Figure 5.17a still continue to remain high for $\mu_t \leq 0.5$, whilst decrease at the increasing of the mixed parameter, corresponding then to more complex community structure. The ARI curves in Figure 5.17b, remain acceptable for all $\mu_t \leq 0.4$ while decreasing at higher values of μ_t. As we have repeatedly said, this is obviously caused by the fully random search at the basis of the algorithm that requires a longer time to converge towards good solutions. The same analysis can be done also for the NVI curves in Figure 5.17c. This is confirmed by looking at the convergence behaviours shown in Section 5.2.3 (Figure 5.5), wherein each of them the relative convergence is represented by a monotonically increasing curve with respect to the number of generations.

For the instance with 3000 and 5000 vertices, it is important to note that the behaviour of the NMI curve on the plot in Figure 5.17a, where the NMI curve values are on average high (≥ 0.55), highlights the limit of Opt-IA due to its randomness, and, consequently, pointing out the need to have longer iterations for solving larger



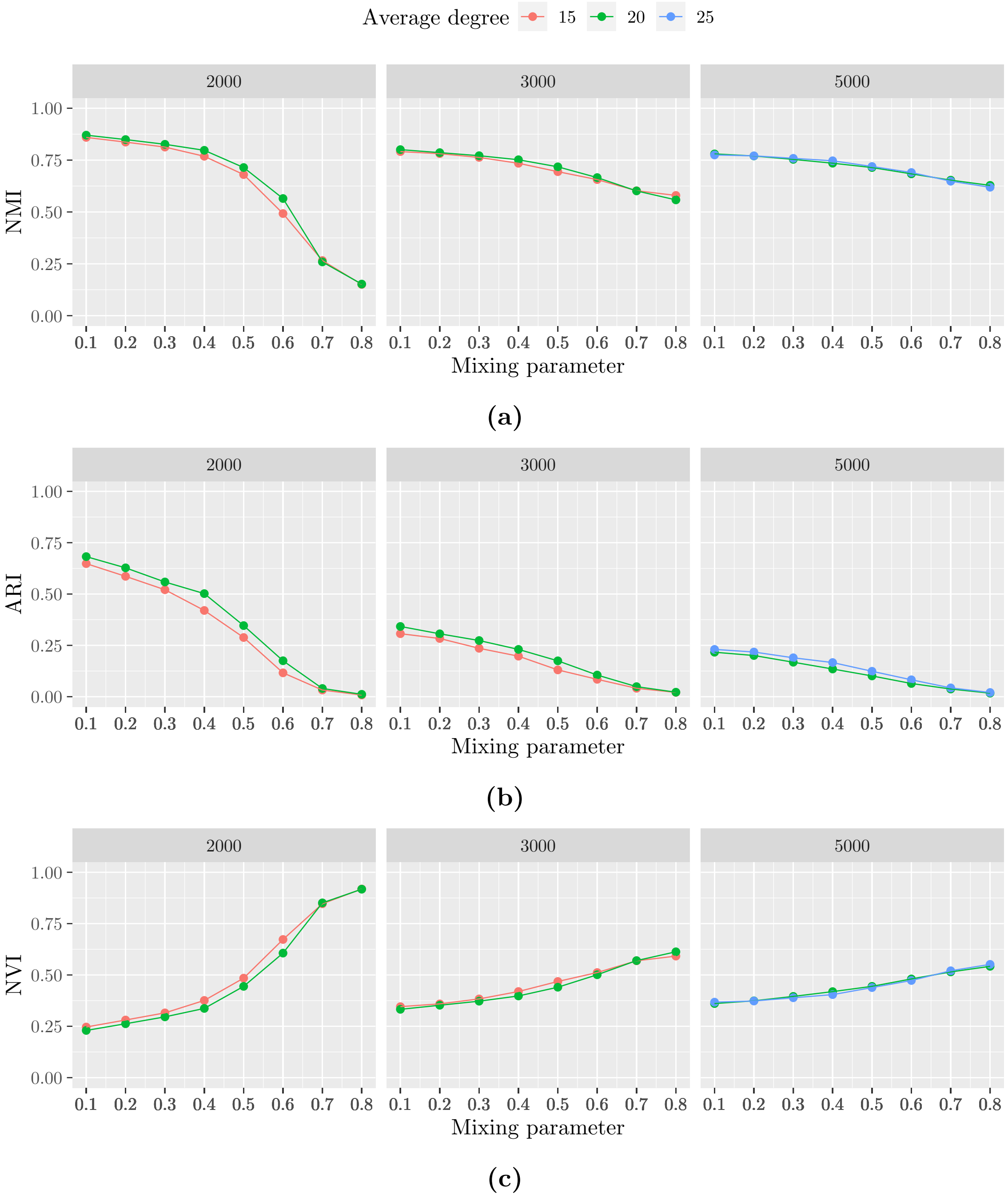
(a)

(b)

(c)

Figure 5.16: Functional sensitivity analysis of Opt-IA performed on LFR benchmark instances with 300, 500 and 1000 vertices. (a) Normalized Mutual Information, (b) Adjusted Rand Index and (c) Normalized Variation of Information.

networks. The same statement can be also made for plots in Figures 5.17b and 5.17c. On the other hand, however, these high NMI curve values obtained by Opt-IA prove the ability of the algorithm to detect communities as similar to the target ones as possible.



(a)

(b)

(c)

Figure 5.17: Functional sensitivity analysis of Opt-IA performed on LFR benchmark instances with 2000, 3000 and 5000 vertices. (a) Normalized Mutual Information, (b) Adjusted Rand Index and (c) Normalized Variation of Information.

5.4 Conclusions

A novel immune algorithm was designed and developed for community detection, which represents one of the most influential problems in many research areas. The proposed algorithm, called Opt-IA, is inspired by the clonal selection principle, and consequently is based on three main immune operators, such as cloning, hypermutation and stochastic aging, whose combination allows the algorithm to perform in a proper way the exploration and exploitation of the search space. The presented algorithm is entirely blind to the features of the problem because it is mainly based on a pure random search of the solutions combined with stochastic operators. In this way, the algorithm can easily jump out from local optimal and perform an extensive exploration thanks to the high diversity in the population produced by the several stochastic strategies developed.

The reliability and efficiency of Opt-IA in community detection has been tested on several social and biological networks, each of them showing different complexity and dimensions. By inspecting the results of all the performed experiments, it clearly emerges the efficiency and reliability of Opt-IA, as well as its robustness as proven in the analysis of the convergence quality and learning capability. Having included a random-search strategy in Opt-IA along with several stochastic operators, it allows the algorithm to carry out a careful and at the same time vast exploration of the search space. An analysis of the computational time complexity has been also conducted by making use of the *time-to-target* (TTT) plots, which confirm that Opt-IA albeit it needs more iterations compared to other algorithms (due to its pure randomness), it reaches however the best solutions in acceptable times.

In order to assess Opt-IA with respect to the state-of-the-art in community detection, the algorithm was compared against about twenty different heuristics and metaheuristics. From these comparisons, it appears very clear how the proposed algorithm strictly outperforms most of the compared algorithms, except for the hyper-Heuristic where instead the performances can be considered comparable in the overall. In particular, the main difference in the performances between the hyper-Heuristic and Opt-IA is given on the values of the average of the best solutions found on 30 independent runs. However, this is reasonably foreseeable since the main feature of hyper-Heuristic methods is the combination of several heuristics, efficient on the problem to be tackled, in order to exploit the strength of one to overcome the weaknesses of the others, while OPT-IA is an algorithm entirely based on random-search combined with pure stochastic operators.

In conclusion, all the outcomes and the analysis conducted to prove the reliability of the proposed random search, making OPT-IA comparable with sophisticated algorithms, especially on networks that are not too dense, such as biological networks for instance. Obviously, the limit of the random search, and therefore of OPT-IA, is the need to have a large number of generations to converge to acceptable solutions when tackling wide networks (e.g. $|V| \geq 5000$). However, since the solution search process is entirely guided by randomness and stochastic operators, and therefore without any deterministic approach or any information on the features of the network (OPT-IA is a fully blind algorithm), it allows, the other hand, to be easily adapted and applied in dynamic network scenarios and situations of high uncertainty.

6

Hybrid Immunological Algorithm

6.1 The Proposed Method

Immunological 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 [64]. Hybrid-IA [41] uses a deterministic local search, based on rational choices that refine and improve the solutions found so far and belongs to the special class *Clonal Selection Algorithms* (CSA) [111, 47], whose efficiency is due to the three main immune operators: (i) cloning, (ii) hypermutation, and (iii) aging. Furthermore, this algorithm is based on two main concepts: antigen (Ag), which represents the problem to tackle, and B cell, or antibody (Ab) which represents a candidate solution, i.e. a point in the solution space.

At each time step t, the algorithm maintains a population of d candidate solutions:

each solution is a subdivision of the vertices of the graph $G = (V, E)$ in communities. Let $N = |V|$, a B cell $\vec{x}$ is a sequence of N integers belonging to the range $[1, N]$, where $x_i = j$ indicates that the vertex i has been added to the cluster j. The population is initialized at the time step $t = 0$ randomly assigning each vertex i to a group j, with $j \in [1, N]$. Just after the initialization step, the algorithm evaluates the fitness function of each generated element ($\vec{x} \in P^{(t)}$), i.e. Equation 4.1, using the procedure ComputeFitness($P^{(t)}$). Hybrid-IA ends its evolution once the halting criterion is reached, which was fixed to a maximum number of generations (T_{max}). The pseudo-code of Hybrid-IA is described in Algorithm 6.1.

Algorithm 6.1: Pseudo-code of the hybrid immunological algorithm Hybrid-IA.

```
procedure Hybrid-IA($d$, $dup$, $\rho$, $\tau_B$)
    $t \leftarrow 0$
    $P^{(t)} \leftarrow$ InitializePopulation($d$)
    ComputeFitness($P^{(t)}$)
    repeat
        $P^{(clo)} \leftarrow$ Cloning($P^{(t)}, dup$)
        $P^{(mut)} \leftarrow$ Hypermutation($P^{(clo)}, \rho$)
        ComputeFitness($P^{(mut)}$)
        $(P_a^{(t)}, P_a^{(mut)}) \leftarrow$ Aging($P^{(t)}, P^{(mut)}, \tau_B$)
        $P^{(sel)} \leftarrow (\mu + \lambda)$−Selection($P_a^{(t)}, P_a^{(mut)}$)
        $P^{(t+1)} \leftarrow$ LocalSearch($P^{(sel)}$)
        $t \leftarrow t + 1$
    until (termination criterion is satisfied)
end procedure
```

Cloning is the first immune operator to be carried out, which simply copies *dup* times each B cell producing an intermediate population $P^{(clo)}$ of size $d \times dup$. A static version was considered for avoiding premature convergences, which can instead occur using the proportional one. Indeed, if a number of clones proportional to the fitness value are produced, preferring the cloning of the best through a higher number of clones, already in the first iterations is very likely that a population of B cells very similar to each other is obtained, with the outcome to cannot perform a proper exploration of the search space, and thus getting easily trapped in local optima. Once a clone is created, Hybrid-IA assigns an age to it that determines how long the clone/solution can live inside the population: from such assigned age until it reaches the maximum age allowed τ_B (user-defined parameter). Specifically, a random age chosen in the range $[0 : \frac{2}{3}\tau_B]$ is assigned to each clone. 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 (described below) play a crucial role in Hybrid-IA performances, and any evolutionary algorithm in general, because they are able to keep the right amount of diversity among the solutions, helping thus the algorithm to avoid premature convergences [54].

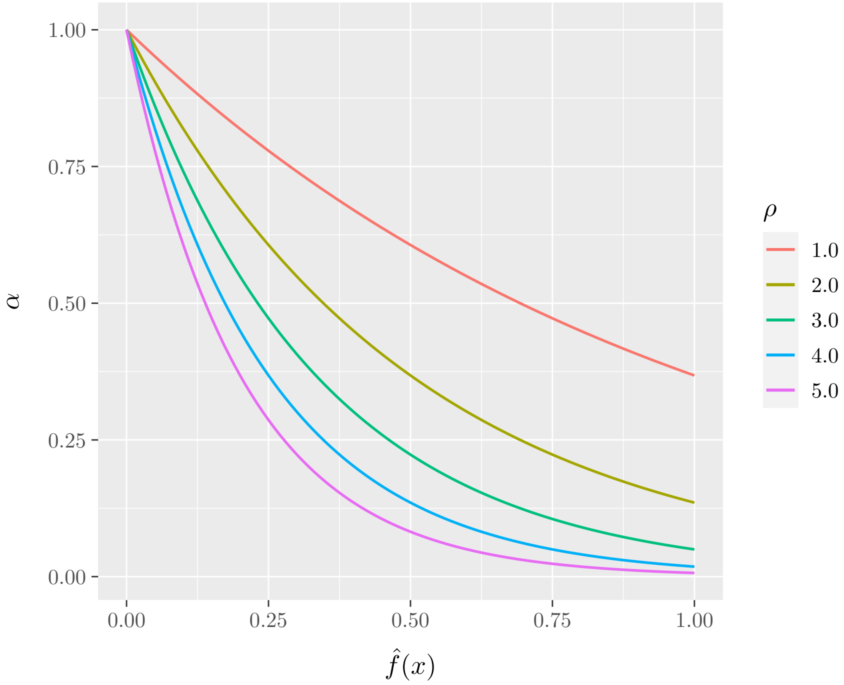


Figure 6.1: Impact of the mutation shape ρ on the probability α of mutation operator, with respect to the normalized modularity value.

The aim of the *hypermutation* operator is to generate new elements, acting on each clone in $P^{(clo)}$, with the main purpose of efficiently and carefully exploring the search space. Just as happens in the natural immune system, the number of changes on each clone, called *mutation rate*, is determined through an *inversely proportional* law to the fitness function value of the B cell considered: better the fitness value of the solution, smaller the relative mutation rate will be. In particular, let $\vec{x}$ be a cloned B cell, the mutation rate $\alpha = e^{-\rho\hat{f}(\vec{x})}$ is defined as the probability to move a vertex from one community to another one, where ρ is a user-defined parameter that determines the shape of the mutation rate, and $\hat{f}(\vec{x})$ is the fitness function normalized in the range $[0, 1]$. In Figure 6.1 are displayed the curves of the mutation rate behaviour at the varying of the ρ parameter. It is therefore shown how the ρ mutation shape affects the probability α, at different fitness function values. Indeed, it is possible to observe how low fitness values (which represent not good solutions) correspond to high α values, which means that a high number of vertices will be moved from one community to another; vice versa, at high fitness values (i.e. the best solutions), correspond low α values, and that is only a low number of vertices will be moved.

Formally, it works as follows: for each B cell, two integers c_i and c_j ($c_i \neq c_j$) are randomly chosen, which represent respectively two communities (see description of solution representation above). The first one is chosen among the existing communities, whilst the latter is randomly chosen in the range $[1, N]$. Then, all vertices in c_i are moved in c_j with a probability given by α. Note that, however, c_j might not match with any currently existing community; in this case, a new community will be created and added to the existing ones. Depending on the c_j value, two different mutation approaches can occur: *merging* and *splitting*. In the first, a subset of vertices of community c_i will be moved and therefore merged with another existing community c_j; in the second, instead, a community c_i will be divided into two communities: c_i itself, and a new one c_j. In figure 6.2 is reported a simple example of how these two approaches work (merging in Figure 6.2a and splitting in Figure 6.2b) for better comprehension. The main idea behind the hypermutation operator is to create and discover new communities by moving a variable percentage of vertices from existing communities. This search method balances the effects of local search (described below), allowing the algorithm to avoid premature convergences towards local optima.

The *static aging operator* is the one that plays a central role in the efficiency and reliability of Hybrid-IA, particularly when it is applied to complex and large problems. It simply acts on each mutated B cell by removing older ones from the two populations $P^{(t)}$ and $P^{(mut)}$. Let τ_B be the maximum number of generations allowed for every B cell

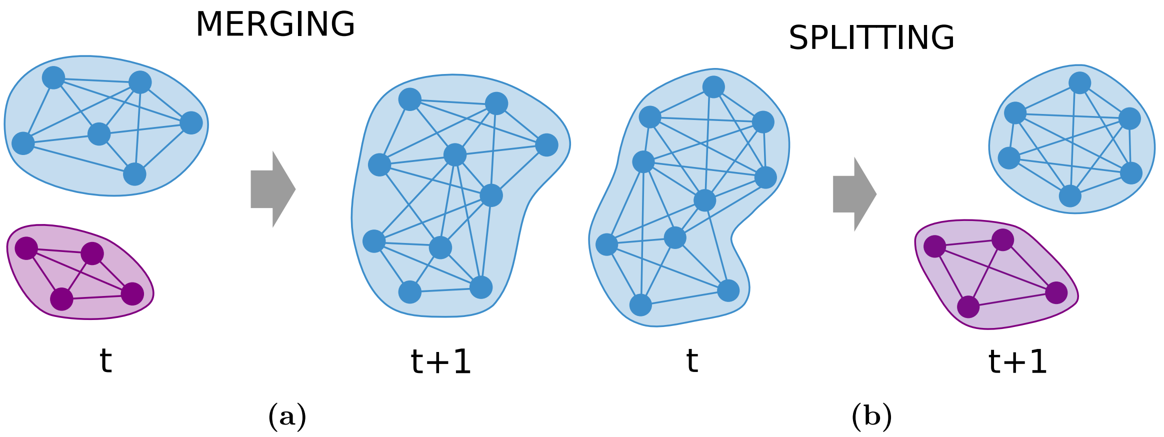


Figure 6.2: Result of mutation operator in terms of community structure changes of the current partition. (a) A subset of vertices from community c_i will be merged to an existing community c_j. (b) A subset of vertices from community c_i will be divided to create a new community c_j.

to stay in the population; once the age of a B cell exceeds τ_B, it will be removed from the relative population, independently from its fitness value. However, an exception may be done for the best current solution, which is kept alive even if its age is older than τ_B. Such variant is called *elitist aging operator*. The purpose of this operator is, then, 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.

The last operator to be performed within the evolutionary cycle is the $(\mu + \lambda)$-*Selection operator*, with $\mu = d$ and $\lambda = (d \times dup)$, which has the aim to select the best d survivors from both populations $P_a^{(t)}$ and $P_a^{(mut)}$, producing a temporary population $P^{(sel)}$, on which the local search will be performed later. Basically, it identifies the best d elements among the set of offspring and the parent B cells (those that survived the aging step), ensuring monotonicity in the evolutionary dynamics.

The local search designed and introduced is the key operator to properly speed up the convergence of the algorithm, and, in a way, drive it towards more promising regions. Furthermore, it intensifies the search and explores the neighbourhood of each solution using the well-known *Move Vertex* (MV) approach [86]. The basic idea of the proposed LS is to assess deterministically if it is possible to move a vertex from its community to another one within its neighbours. The MV approach takes into account the *move gain* that can be defined as the variation in modularity produced when a vertex is moved from one community to another. Before formally defining the move gain, it is important to point out that the modularity Q, defined in Equation (4.1), can be rewritten as:

$$Q(c) = \sum_{i=1}^{k} \left[\frac{\ell_i}{M} - \left(\frac{d_i}{2M} \right)^2 \right], \tag{6.1}$$

where k is the number of the found communities; $c = \{c_1, \ldots, c_i, \ldots, c_k\}$ is the set of communities that is the partitioning of the set of vertices 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. Thus, the *move gain* of a vertex $u \in c_i$ is the modularity variation produced by moving u from c_i to c_j, that is:

$$\Delta Q_u(c_i, c_j) = \frac{l_{c_j}(u) - l_{c_i}(u)}{M} + d_V(u) \left[\frac{d_{c_i} - d_V(u) - d_{c_j}}{2M^2} \right], \tag{6.2}$$

where $l_{c_i}(u)$ and $l_{c_j}(u)$ are the number of links from u to vertices in c_i and c_j respectively, and $d_V(u)$ is the degree of u when considering all the vertices V. If $\Delta Q_u(c_i, c_j) > 0$, then moving vertex u from c_i to c_j produces an increment in modularity, and therefore a possible improvement. Consequently, the goal of MV is to find a vertex u to move to an adjacent community in order to maximize ΔQ_u:

$$\underset{v \in Adj(u)}{\operatorname{argmax}} \Delta Q_u(i, j), \tag{6.3}$$

where $u \in c_i$, $v \in c_j$ and $Adj(u)$ is the adjacency list of vertex u.

For each solution in $P^{(select)}$, the Local Search begins by sorting the communities in increasing order with respect to the ratio between the sum of inside links and the sum of the vertex degrees in the community. In this way, poorly formed communities are identified. After that, MV acts on each community of the solution, starting from vertices that lie on the border of the community, that is, those that have at least an outgoing link. In addition, for communities, the vertices are sorted with respect to the ratio between the links inside and the vertex degree. The key idea behind LS is to deterministically repair the solutions which were produced by the hypermutation operator, by discovering new partitions with higher modularity values. Equation 6.2 can be calculated efficiently because M and $d_V(u)$ are constants, the terms l_{c_i} and d_{c_i} can be stored and updated using appropriate data structures, while the terms $l_{c_i}(u)$ can be calculated during the exploration of all adjacent vertices of u. Therefore, the complexity of the move vertex operator is linear on the dimension of the neighbourhood of vertex u.

6.2 Networks Data Set

In this section, the different social and biological networks used during the tests are summarized, and for which the communities were identified. They are grouped into five types, where three of them, described below, refer to biological interactions and main molecular networks.

6.2.1 Social Networks

Social networks are a classical example of networks with a community structure, as people tend to form groups within their work environment, family, and friends. The instances that have been considered in this work are well-known networks used for the community detection problem. *Grevy's Zebra* [135] is a network created by Sundaresan et al. in which a link between vertices indicates that a pair of zebras appeared together at least once during the study. In *Zachary's Karate Club* [154] network, collected by Zachary in 1977, a vertex represents a member of the club and an edge represents a tie between two members of the club. *Bottlenose Dolphins* [98] is an undirected social network of dolphins where an edge represents a frequent association. *Books about US Politics* [88] is a network of books sold, compiled by Krebs, where edges represent frequent co-purchasing of books by the same buyers. Another network considered is

American College Football [66], a network of football games between colleges. *Jazz Musicians* [67] is the collaboration network between Jazz musicians. Each vertex is a Jazz musician and an edge denotes that two musicians have played together in a band.

6.2.2 Protein-Protein Interaction Networks

The physical interaction between the proteins has always been an important consideration for gene function. Proteins are the main participants in a variety of biological processes inside cells, including signal transduction, homeostasis control, maintenance of internal balance and developmental processes [155]. They rarely function independently but form protein complexes [73]. The mathematical representation of the physical contacts between proteins inside the cell can be obtained through a non-direct binary physical PPI network [97], in which vertices represent proteins and whose edges connect pairs of interacting proteins. By considering the spatial and temporal aspects of interactions, networks can help understand the general organization of protein-protein connections and discover the principles of their organization within the cell. These have a fundamental role in all biological processes and in all organisms [142], therefore a complete knowledge of PPIs and their protein interconnections, would allow the understanding of cell physiology in pathogenic (and normal) states. This would have a great impact on disease diagnosis, disease genes often interact with other disease genes [69], as well as for drug discovery and disease treatment [103, 140, 56]. In this work, two small *Cattle PPI* [31] and *Helicobacter pylori PPI* Protein-Protein interactions [149, 116] and two large networks (with a number of vertices > 2000), related to Yeast PPI instances [153, 25] have been considered. All networks in question are related to the data of interactions between proteins in the three different organisms mentioned before (cattle, helicobacter pylori and yeast) where each vertex represents a protein and they are linked if they interact physically within the cell.

6.2.3 Metabolic Networks

With the technological advancement and the sequencing of whole genomes, as well as it has been possible to reconstruct the protein-protein interaction networks described above, it has also been possible to obtain the networks of biochemical reactions in many organisms. Metabolic networks are powerful tools to represent and study a complete set of relationships between metabolites, small chemical compounds, and proteins/enzymes. They describe the set of processes and reactions that determine the biochemical and physiological properties of a cell, including the chemical reactions of metabolism, the metabolic pathways and regulatory interactions that drive these reactions. Metabolic networks make it possible to detect diseases given an enzymatic defect in a reaction that can affect flows in subsequent reactions. These defects often cause cascading effects responsible for associated metabolic diseases [93]. Therefore, this type of network can be used to understand if metabolic disorders are linked due to their related reactions [119]. In order to investigate this functional information, it is necessary to identify the functional modules in it [152]. Identifying the communities in the metabolic networks will help in understanding the pathways and cycles in metabolic networks [62]. In the two considered real networks, the metabolic network of *Caenorhabditis elegans* [57] and *E. coli* bacteria [123], each vertex represents a metabolite, and each direct link a reaction between them that binds the metabolite with the reaction product.

6.2.4 Transcriptional Regulatory Networks

Understanding the mechanisms underlying the regulation of gene expression is the main goal of contemporary biology. Important cellular processes, such as cell differentiation, cell cycle and metabolism are controlled by the complex biological mechanism of gene regulation. However, the relationship between structure and regulatory function is not easy to observe experimentally. Therefore, a Systems Biology-based approach is needed. In this regard, network theory is useful for understanding the activity behind these complex transcriptional regulatory mechanisms [7]. The transcriptional network can be represented as a direct graph, composed of transcription factors (TFs) and target genes (TGs) which are regulated in a tightly coordinated way. Within the network, each vertex represents a gene (or operon, in the case of prokaryotic organisms) and the edges represent direct transcriptional regulation. Each edge is directed from a gene (or operon) that encodes a transcription factor to a gene (or operon) that is regulated by that transcription factor. Transcription factors are modular proteins that regulate the gene expression of other proteins by binding to specific sites in the DNA (promoter sites) and allowing (or preventing) the synthesis of mRNA. The relationships between TFs and their targets (TGs) determine a given phenotype [118]. Moreover, in transcription regulatory networks, modules (or communities) correspond to sets of co-regulated genes [26, 99, 148, 159]. For this reason, the problem of community detection plays a relevant role [139]. Escherichia coli and Saccharomyces cerevisiae are two well-known organisms often used as a model for studying gene regulation. In this work, two transcriptional regulatory networks, *E. coli TRN* [129] and *Yeast TRN* [102] have been considered, constituted by transcription factors and target genes, where each edge in the network is directed from an operon that encodes a TF to an operon that it directly regulates.

6.2.5 Synthetic Networks

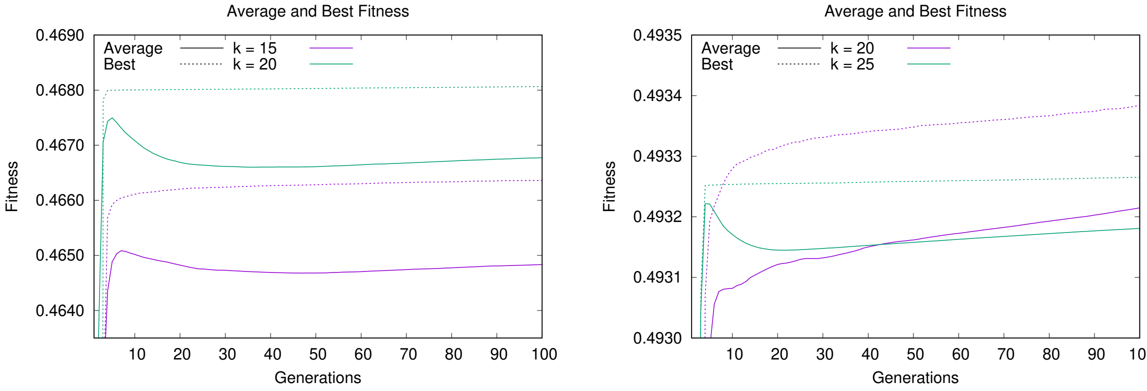
In addition to real biological networks, artificial instances were also taken into account in the experimental phase. These synthetic networks can be generated with different characteristics and with a known community structure. Using these kinds of networks allows for testing of the algorithms on different scenarios and gives the possibility of evaluating the goodness of the detected communities. The algorithm used to generate these synthetic networks is the LFR benchmarks, proposed in [92, 91]. The algorithm assumes that both the distributions of degree and community size are power laws, with exponents τ_1 and τ_2, respectively. The mixing parameter μ_t, identifies the relationship between the vertex's external and internal degree. In particular, each vertex of the networks shares a fraction $1-\mu_t$ of its edges with the other vertices of its community and a fraction μ_t with other vertices outside of its community. Also, the LFR benchmarks can be used to generate directed and weighted synthetic networks with overlapping communities. More details on the LFR algorithm about key parameters and how to generate benchmark instances can be found in [92, 91].

6.3 Experimental Results

For evaluating the efficiency and reliability of Hybrid-IA, many experiments have been performed on all biological networks described above (see Section 6.2). In each experiment Hybrid-IA maintains a population of $d = 100$ B cells; uses a duplication parameter $dup = 2$; keeps a solution for at most $\tau_B = 5$ generations within the population, and uses $\rho = 1.0$ as mutation shape. This parameter setting comes out from preliminary experimental results performed, and from previous knowledge learned.

6.3.1 Convergence Behaviour

In the initial part of the experimental phase, the analysis has been focused on the convergence behaviour and learning rate in order to inspect the efficiency of Hybrid-IA. For this study, artificial networks have been taken into account as benchmark instances, which have been generated by the LFR algorithm [92, 91] and described in Section 6.2.5. In particular, networks with 1000 vertices and average degree 15 and 20, and networks with $|V| = 5000$ and average degree 20 and 25 have been generated. For each of these networks generated, the maximum degree was set to 50, while the exponents of the power laws, which control the degree and community sizes distribution (τ_1 and τ_2), have been set to 2 and 1, respectively. A minimum of 10 vertices to a maximum of 50 have been set as sizes of the communities. The mixing parameter μ_t was fixed to 0.5. Finally, for these experiments, a maximum number of generations $T_{max} = 100$ was set, and 5 random instances were generated for each network parameters configuration.

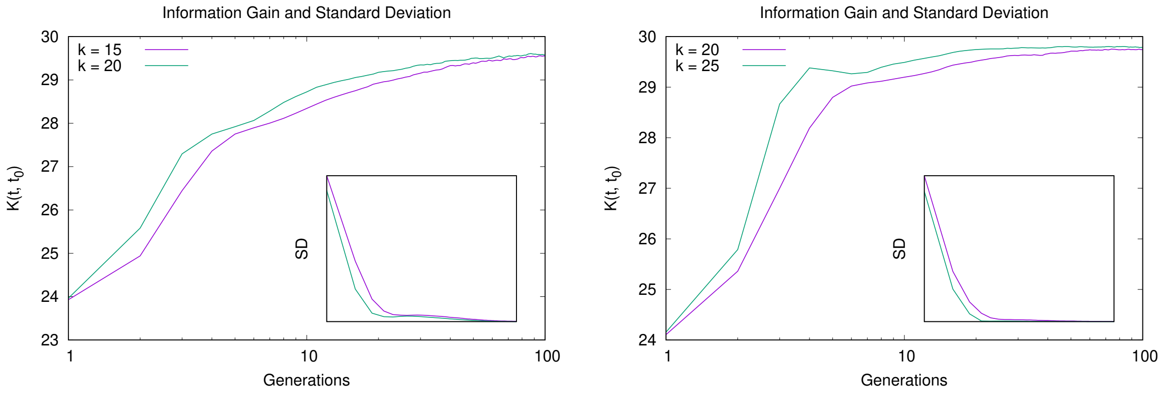


(a) **(b)**

Figure 6.3: Convergence behaviour of Hybrid-IA on LFR benchmark instances with 1000 and 5000 vertices. Average and best fitness value versus generations on (a) `LFR(1000,15,0.5)` and `LFR(1000,20,0.5)`, and on (b) `LFR(5000,20,0.5)` and `LFR(5000,25,0.5)`.

In Figure 6.3a is shown the convergence plot on the LFR instances with 1000 vertices and average degree k of 15 and 20. The two curves represent the best and average fitness of the population and both are averaged over 100 independent runs. From this plot can be noted how the two curves of the best fitness have the same trend for both values of k: reach a high value of modularity in the early generations and then improves slowly. The improvement for $k = 20$ compared to the first generations is minimal, while for $k = 15$ the increase in modularity is slightly more significant. Instead, the average fitness curves have a similar trend in the first generations, but subsequently decrease and then gradually increase. From these two curves can be seen how the population maintains a good degree of diversity within the population, favouring thus a better exploration of the search space.

A similar situation can be also observed on the LFR instances with 5000 vertices. The plots in Figure 6.3b show the best and average fitness of the population for $k = 20$ and $k = 25$. For $k = 25$, Hybrid-IA obtains a high value of modularity in a few generations, and after that it stays in a steady state for the rest of the execution, reaching a high-modularity plateau [71]. On the other hand, for $k = 20$ the algorithm has a growth much more constant and linear, both in terms of the best solution and the average of the population. Also, in this case, the two curves of best and average fitness are well separated, indicating that the algorithm maintains a good diversity of solutions within the population.



(a) **(b)**

Figure 6.4: Learning ability of Hybrid-IA on LFR benchmark instances with 1000 and 5000 vertices. Information gain and standard deviation versus generations on (a) `LFR(1000,15,0.5)` and `LFR(1000,20,0.5)`, and on (b) `LFR(5000,20,0.5)` and `LFR(5000,25,0.5)`.

Once analyzed the convergence behaviour, an investigation on the learning ability of Hybrid-IA has been performed as well, using the information gain that measures the quantity of information the algorithm gains during the evolutionary process [89, 90], that is the amount of information learned compared to the randomly generated initial population. At each generation t, let $B_m^{(t)}$ be the number of the B cells that have the fitness function value to m; the candidate solutions distribution function $f_m^{(t)}$ can be defined as the ratio between the number $B_m^{(t)}$ and the total number of candidate solutions:

$$f_m^{(t)} = \frac{B_m^{(t)}}{\sum_m B_m^{(t)}} = \frac{B_m^{(t)}}{d}. \tag{6.4}$$

It follows that the information gain $K(t, t_0)$ can be calculated as:

$$K(t, t_0) = \sum_m f_m^{(t)} \log \left(\frac{f_m^{(t)}}{f_m^{(t_0)}} \right). \tag{6.5}$$

The plots in Figures 6.4a and 6.4b show the information gain obtained by the algorithm during its running in different scenarios. For both values of $|V|$, Hybrid-IA

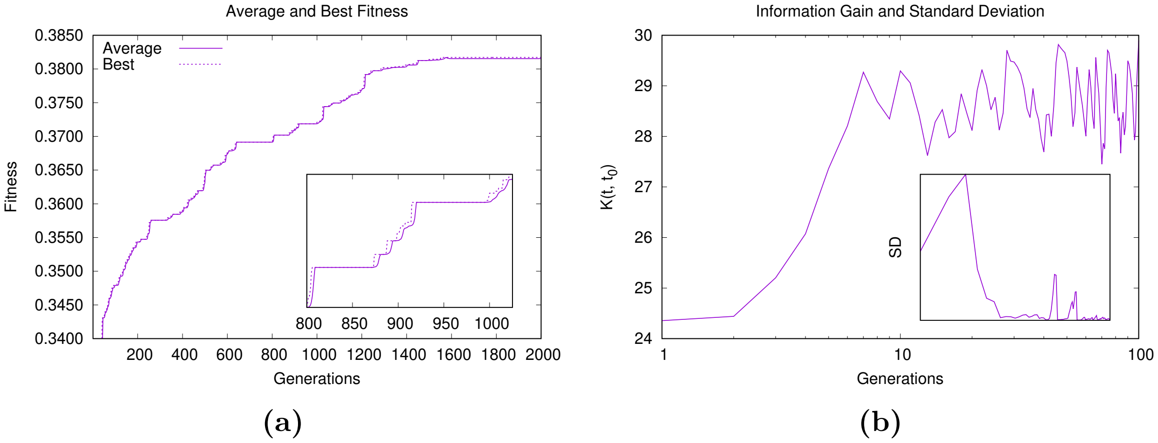


Figure 6.5: Convergence behaviour and learning ability of Hybrid-IA on the *E. coli MRN* network. (a) Average and best fitness value of the population versus generations. (b) Information gain and standard deviation versus generations.

is able to learn information step by step, showing thus an increasing curve until it reaches a steady state, which is exactly when the modularity of all solutions begins to become similar. The monotonically increasing of the information gain curve until reaching a steady state is consistent with the *maximum information-gain principle*: $\frac{dK}{dt} \geq 0$. In the overall, the convergence behaviour and learning process analyzed (Figures 6.3 and 6.4), suggest that Hybrid-IA finds very quickly good solutions in networks with medium/high density (i.e. $k = 20$ for 1000 vertices and $k = 25$ for 5000), as the community structure is well-defined. On the other hand, on sparse networks, with an unclear community structure, the algorithm converges more slowly.

In addition, a convergence analysis on the network *E. coli MRN* [123] was carried out, which presents a very low density (less than 1%). In Figures 6.5a and 6.5b are shown the plots relative to the run in which Hybrid-IA has reached its best solution. Again, after the initial climb, the algorithm begins its exploration around the solutions found, gradually improving. In some places (inset plot of Figure 6.5a) the algorithm seems to stagnate in some local optima but, thanks to the *aging* operator, manages to escape, finding better solutions. During these phases, the population tends to reduce its diversity, being almost entirely composed of solutions of equal quality.

Finally, the experimental analysis has been focused on the inspection of the efficiency of Hybrid-IA with respect to Opt-IA, in terms of convergence and solution quality found. In Figure 6.6, the convergence behaviour of both Opt-IA and Hybrid-IA on the *Books about US Politics* network is shown. In this plot, the curves represent the evolution of the best and average fitness of the population; the standard deviation of the fitness values of the population is superimposed onto the average fitness and gives an idea about how heterogeneous the elements in the population are.

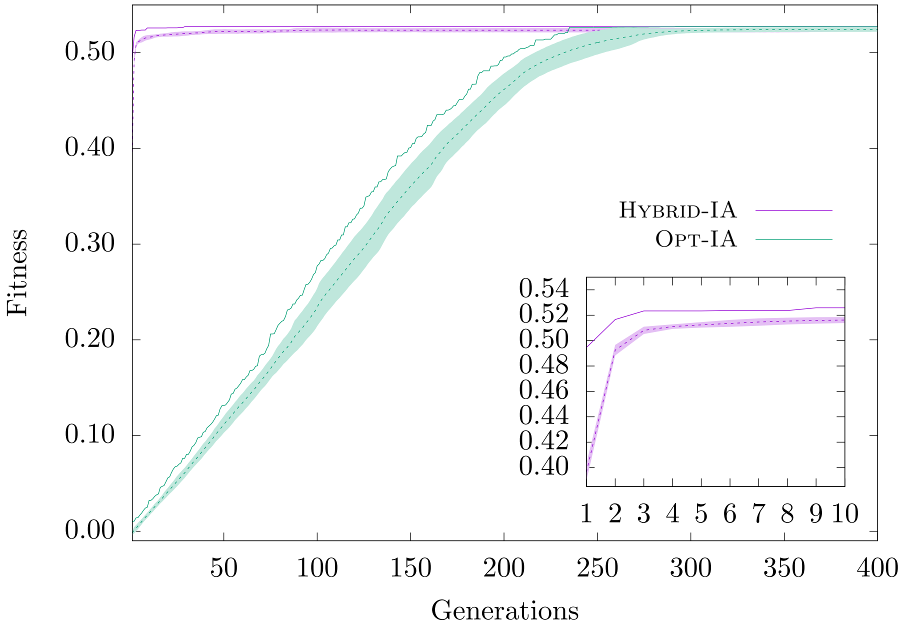


Figure 6.6: Comparative convergence behaviour of Opt-IA and Hybrid-IA on the *Books about US Politics* network.

From Figure 6.6, one can note how Opt-IA converges more slowly towards the best solution, as expected, always keeping a certain variability within the population. This allows the algorithm to better explore the search space. When the population is composed of very different elements, i.e., when the standard deviation is high, the algorithm discovers new solutions, significantly improving the current best solution. However, after about 250 generations, Opt-IA reaches the optimal solution and the curves (best and average fitness) tend to overlap. Moreover, the achievement of the optimal solution helps the creation of better clones, reducing the variability of the population. Unlike Opt-IA, Hybrid-IA converges easily thanks to the local search applied to the elements after the selection phase. As can be noted from the inset plot in Figure 6.6, Hybrid-IA reaches the optimal solution after a few generations. Even in this case, once the best solution is reached, the population follows the same trend of the curve of the best fitness, and both curves continue (almost) as a single line. If on one hand, the local search helps to quickly discover good solutions, on the other, it reduces the diversity inside the population, reducing then the exploration of the search space. In particular, as demonstrated by the worst value found in the *Jazz Musicians* network, reported in Table 6.2, Hybrid-IA prematurely converges towards local optima, from which it will hardly be able to get out. At the end of the analysis of Figure 6.6, it is possible to conclude that the stochastic operators designed in Opt-IA guarantee an excellent and large exploration of the search space, but with the disadvantage of requiring a longer evolution time; however, the local search developed in Hybrid-IA, and relative sorting criteria, allow for quickly discovering good solutions to exploit during the evolutionary process.

6.3.2 Results

In this section, the outcomes obtained by Hybrid-IA on the social and biological networks, described above and summarize in Table 6.1, are presented and analyzed. For proving the competitiveness and reliability of Hybrid-IA with respect to the state of the art and assessing its performance in general, the algorithm was compared to other well-known metaheuristics, each based on a modularity optimization approach.

In particular, it was compared with an effective Hyper-Heuristics Differential Search Algorithm (HDSA) [4, 33] based on the migration of artificial superorganisms; an improved Bat Algorithm (BADE) [4, 150, 133] based on Differential Evolution algorithm; a Scatter Search (SSGA) [4, 68, 100] algorithm based on the Genetic Algorithm; a modified Big Bang–Big Crunch (BB-BC) [4, 58] algorithm; the original Bat Algorithm (BA) [4, 150] based on echolocation behaviour of bats adapted for community detection; and the original Gravitational Search Algorithm (GSA) [4, 117], re-designed for solving the community detection problem. Further, the Louvain algorithm [12], a greedy optimization method that attempts to optimize the modularity, was also con-

Table 6.1: Social and biological network instances used in the experiments.

Name	*Reference*	$\|V\|$	$\|E\|$
Grevy's Zebras	[135]	28	111
Zachary's Karate Club	[154]	34	78
Bottlenose Dolphins	[98]	62	159
Books about US Politics	[88]	105	441
American College Football	[66]	115	613
Jazz Musicians Collaborations	[67]	198	2742
Cattle PPI	[31]	268	303
E. coli TRN	[129]	418	519
C. elegans MRN	[57]	453	2025
Yeast TRN	[102]	688	1078
H. pylori PPI	[149, 116]	724	1403
E. coli MRN	[123]	1039	4741
Yeast PPI (1)	[153]	2018	2705
Yeast PPI (2)	[25]	2284	6646

sidered for the comparison[1].

The parameter configuration used by Hybrid-IA is the same as described above, whilst the number of generations (T_{max}) considered depends on the size of the network tested: for social instances, T_{max} was set to 100, for biological instances with less than 1000 vertices, T_{max} was set to 1000, while for the ones with more than 1000 vertices, T_{max} is 2000. It is important to highlight that in all compared algorithms the results have been taken from [4].

The comparison performed on all social networks is reported in Table 6.2, where we show for each algorithm (where possible) the best, mean, and worst values of the Q modularity; standard deviation, and, finally, the created communities number (k). Furthermore, whilst the experiments for Hybrid-IA, Opt-IA and Louvain were performed on 100 independent runs, for all other compared algorithms, only 30 independent runs have been considered. Obviously, all algorithms optimize the same fitness function reported in Equation 4.1 and rewritten in a simpler way in Equation 6.1.

From Table 6.2, it is possible to note that all algorithms reach the optimal solution in the first two networks *Grevy's Zebras* and *Zachary's Karate Club*; on all the

[1] For these experiments we used the C++ source code of Louvain algorithm that can be downloaded from this link: https://perso.uclouvain.be/vincent.blondel/research/louvain.html.

Table 6.2: Comparative results of Hybrid-IA and other algorithms on social networks. The results are calculated over 100 independent runs for Hybrid-IA, Opt-IA and Louvain, while over 30 runs for the rest.

Name		Hybrid-IA	Opt-IA	Louvain	HDSA	BADE	SSGA	BB-BC	BA	GSA
Grevy's Zebras	*k*	4	4	4	4	4	4	4	4	4
	Best	0.2768	0.2768	0.2768	0.2768	0.2768	0.2768	0.2768	0.2768	0.2768
	Worst	0.2768	0.2768	0.2768	0.2768	0.2768	0.2768	0.2761	0.2768	0.2768
	Mean	0.2768	0.2768	0.2768	0.2768	0.2768	0.2768	0.2766	0.2768	0.2768
	StD	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0003	0.0000	0.0000
Zachary's Karate Club	*k*	4	4	4	4	4	4	4	4	4
	Best	0.4198	0.4198	0.4188	0.4198	0.4198	0.4198	0.4198	0.4198	0.4198
	Worst	0.4198	0.4198	0.3854	0.4198	0.4156	0.4198	0.4188	0.3946	0.4107
	Mean	0.4198	0.4198	0.4156	0.4198	0.4188	0.4198	0.4196	0.4133	0.4170
	StD	0.0000	0.0000	0.0064	0.0000	0.0018	0.0000	0.0004	0.0105	0.0037
Bottlenose Dolphins	*k*	5	5	5	5	4	5	5	4	6
	Best	0.5285	0.5285	0.5185	0.5285	0.5268	0.5257	0.5220	0.5157	0.4891
	Worst	0.5220	0.5268	0.5176	0.5276	0.4940	0.5156	0.5049	0.4427	0.4517
	Mean	0.5273	0.5285	0.5203	0.5282	0.5129	0.5200	0.5141	0.4919	0.4677
	StD	0.0009	0.0003	0.0032	0.0005	0.0120	0.0040	0.0068	0.0289	0.0155
Books about US Politics	*k*	5	5	4	5	4	5	9	3	5
	Best	0.5272	0.5272	0.5205	0.5272	0.5239	0.5221	0.4992	0.5211	0.4775
	Worst	0.5246	0.5063	0.5102	0.5272	0.5137	0.5167	0.4799	0.4815	0.4558
	Mean	0.5270	0.5267	0.5261	0.5272	0.5178	0.5203	0.4914	0.5020	0.4661
	StD	0.0005	0.0028	0.0027	0.0000	0.0042	0.0024	0.0084	0.0149	0.0079
American College Football	*k*	10	10	10	10	11	6	10	7	5
	Best	0.6046	0.6046	0.6046	0.6046	0.5646	0.5330	0.5171	0.5523	0.4175
	Worst	0.6031	0.5736	0.5963	0.6019	0.5430	0.5189	0.4986	0.4742	0.3905
	Mean	0.6039	0.5989	0.6038	0.6033	0.5513	0.5277	0.5061	0.5272	0.4032
	StD	0.0007	0.0078	0.0018	0.0009	0.0085	0.0057	0.0069	0.0325	0.0109
Jazz Musicians	*k*	4	4	4	-	-	-	-	-	-
	Best	0.4451	0.4451	0.4451	-	-	-	-	-	-
	Worst	0.4446	0.4449	0.4346	-	-	-	-	-	-
	Mean	0.4450	0.4449	0.4422	-	-	-	-	-	-
	StD	0.0002	0.0001	0.0027	-	-	-	-	-	-

other network instances, both Opt-IA and Hybrid-IA outperform all other compared algorithms, matching their best values only with HDSA. It is important to point out, which proves, even more, the efficiency of the proposed immunological algorithm, how the mean values obtained by Opt-IA and Hybrid-IA, on all tested networks, are better than the best modularity found by the other algorithms, such as BADE, SSGA, BB-BC, BA, and GSA; even on the *Bottlenose Dolphins* and *American College Football* networks, the worst modularity value obtained by Opt-IA is equal to or greater than the best one obtained by the same algorithms. On the *Bottlenose Dolphins* network, Opt-IA reaches a better mean value than Hybrid-IA and HDSA, since, because of its random/blind exploration of the search space, it jumps out from local optima more easily than the other three. The opposite behaviour of Opt-IA occurs when the size and complexity of the networks increase. In such a case, it obviously needs more generations to converge towards the optimal solutions and this is highlighted by the mean value and the standard deviation obtained for *Books about US Politics* and *American College Football*. Note that, with longer generations, Opt-IA finds roughly the same mean values as Hybrid-IA.

Hybrid-IA shows more stable results on all tested networks than Opt-IA, obtaining lower standard deviation values in all instances. On *Bottlenose Dolphins* network, Hybrid-IA has a mean value slightly lower than Opt-IA and HDSA, while in *Books about US Politics* lower only than HDSA. As described above, this is due to the local search that leads the algorithm to a premature convergence towards local optima, obtaining the lowest worst value. Furthermore, HDSA is a hyper-heuristic which uses a genetic algorithm and scatter search to create the initial population for the differential search algorithm, speeding up the convergence of the algorithm, and reducing the spread of results. In *Jazz Musicians* network, both Opt-IA and Hybrid-IA algorithms obtain similar results, better than those obtained by Louvain. Finally, if we focus on the comparison with only the Louvain algorithm, both immunological algorithms outperform it in almost all networks (5 out of 6).

Table 6.3 displays the detailed results of Hybrid-IA in comparisons to the others, and presents, for each algorithm, the best values of the Q modularity (*Best*) found, the average of the values (*Mean*), the worst modularity (*Worst*), the standard deviation (*StD*) and the number of community structures (k) detected by the best solution. Noticeably, the proposed Hybrid-IA algorithm outperformed all metaheuristics in terms of both the value of modularity obtained and mean value, except HDSA in the *E. coli TRN* biological network, although it still provides an upper limit very close to that obtained. It is important to highlight that Hybrid-IA results underline the efficiency of the proposed algorithm, also proved by the fact that the average values obtained on *Cattle PPI*, *E. coli TRN*, *C. elegans MRN* and *H. pylori PPI* networks are better than the *Best* modularity values obtained by the other algorithms, with the exception of the Hyper-heuristic Differential Search Algorithm (HDSA).

Furthermore, from the analysis of the results obtained by the Louvain algorithm,

Table 6.3: Comparative results of Hybrid-IA and other algorithms on biological networks. The results are calculated over 100 independent runs for Hybrid-IA, Opt-IA and Louvain, while over 30 runs for the rest.

Name		Hybrid-IA	Opt-IA	Louvain	HDSA	BADE	SSGA	BB-BC	BA	GSA
Cattle PPI	*k*	40	40	40	40	41	40	48	42	43
	Best	0.7195	0.7195	0.7195	0.7195	0.7183	0.7118	0.7095	0.7143	0.7053
	Worst	0.7011	0.7049	0.7181	0.7194	0.7059	0.7052	0.7079	0.7063	0.6949
	Mean	0.7154	0.7161	0.7193	0.7195	0.7138	0.7079	0.7084	0.7100	0.6983
	StD	0.0037	0.0039	0.0005	0.0001	0.0051	0.0025	0.0007	0.0035	0.0041
E. coli TRN	*k*	43	32	41	47	58	61	71	56	61
	Best	0.7785	0.7795	0.7793	0.7822	0.7680	0.7507	0.7520	0.7629	0.7416
	Worst	0.7563	0.7589	0.7747	0.7808	0.7560	0.7412	0.7452	0.7542	0.7328
	Mean	0.7701	0.7670	0.7779	0.7815	0.7621	0.7457	0.7485	0.7599	0.7375
	StD	0.0049	0.0049	0.0011	0.0006	0.0043	0.0035	0.0026	0.0034	0.0034
C. elegans MRN	*k*	10	8	10	13	25	22	21	22	24
	Best	0.4506	0.4487	0.4490	0.4185	0.3473	0.3336	0.3374	0.3514	0.3063
	Worst	0.4321	0.4221	0.4216	0.3962	0.3335	0.3124	0.3194	0.3356	0.2974
	Mean	0.4437	0.4366	0.4365	0.4074	0.3385	0.3220	0.3266	0.3438	0.3039
	StD	0.0040	0.0070	0.0049	0.0010	0.0054	0.0077	0.0074	0.0073	0.0037
Yeast TRN	*k*	33	-	26	-	-	-	-	-	-
	Best	0.7668	-	0.7683	-	-	-	-	-	-
	Worst	0.7363	-	0.7489	-	-	-	-	-	-
	Mean	0.7569	-	0.7607	-	-	-	-	-	-
	StD	0.0050	-	0.0033	-	-	-	-	-	-
H. pylori PPI	*k*	51	19	24	52	69	70	75	62	77
	Best	0.5359	0.5416	0.5462	0.5086	0.4926	0.4726	0.4681	0.4900	0.4600
	Worst	0.5104	0.5116	0.5356	0.5048	0.4809	0.4659	0.4642	0.4738	0.4549
	Mean	0.5240	0.5249	0.5410	0.5078	0.4854	0.4695	0.4660	0.4814	0.4567
	StD	0.0056	0.0063	0.0025	0.0017	0.0047	0.0021	0.0018	0.0073	0.0020
E. coli MRN	*k*	13	9	8	-	-	-	-	-	-
	Best	0.3817	0.3629	0.3734	-	-	-	-	-	-
	Worst	0.3598	0.3282	0.3450	-	-	-	-	-	-
	Mean	0.3695	0.3437	0.3583	-	-	-	-	-	-
	StD	0.0042	0.0064	0.0058	-	-	-	-	-	-
Yeast PPI (1)	*k*	353	386	213	-	-	-	-	-	-
	Best	0.7002	0.6516	0.7648	-	-	-	-	-	-
	Worst	0.6602	0.6178	0.7519	-	-	-	-	-	-
	Mean	0.6798	0.6344	0.7609	-	-	-	-	-	-
	StD	0.0078	0.0089	0.0022	-	-	-	-	-	-
Yeast PPI (2)	*k*	159	317	46	-	-	-	-	-	-
	Best	0.5796	0.4879	0.5961	-	-	-	-	-	-
	Worst	0.5524	0.4473	0.5870	-	-	-	-	-	-
	Mean	0.5652	0.4746	0.5925	-	-	-	-	-	-
	StD	0.0052	0.0085	0.0019	-	-	-	-	-	-

the only deterministic algorithm included in the comparison, it is clear how Hybrid-IA performs well equating the modularity value in the *Cattle PPI* dataset, and exceeding it in the *C. elegans MRN* and *E. coli MRN* networks. For these datasets, the Figures 6.7a, 6.8a, 6.8b show the detected community structures by Hybrid-IA. Figure 6.7b shows the communities generated for *E. coli TRN* network. For the other instances considered, the modularity is however close to the optimal one. Finally, as will be explained in detail in the next section, although Louvain manages to achieve a better maximization of the modularity value than the ones achieved by Hybrid-IA, the latter reveals a higher number of communities. This is due to the different nature of the two algorithms, where Louvain algorithm tends to aggregate communities.

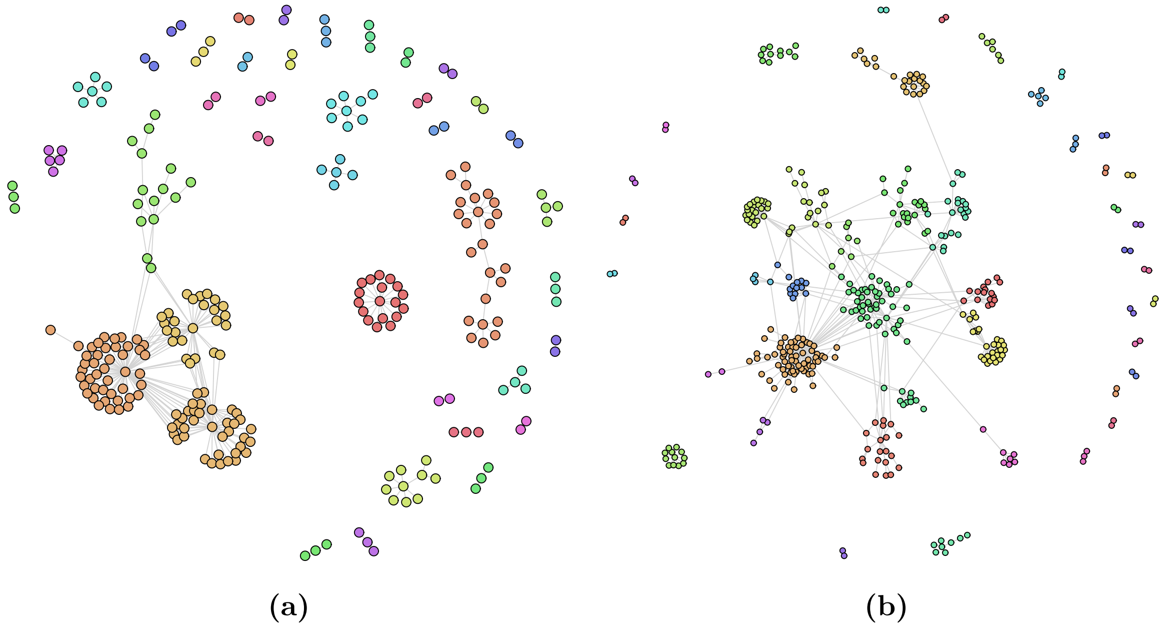


Figure 6.7: Community structures identified by Hybrid-IA on (a) *Cattle PPI* and (b) *E. coli TRN* networks.

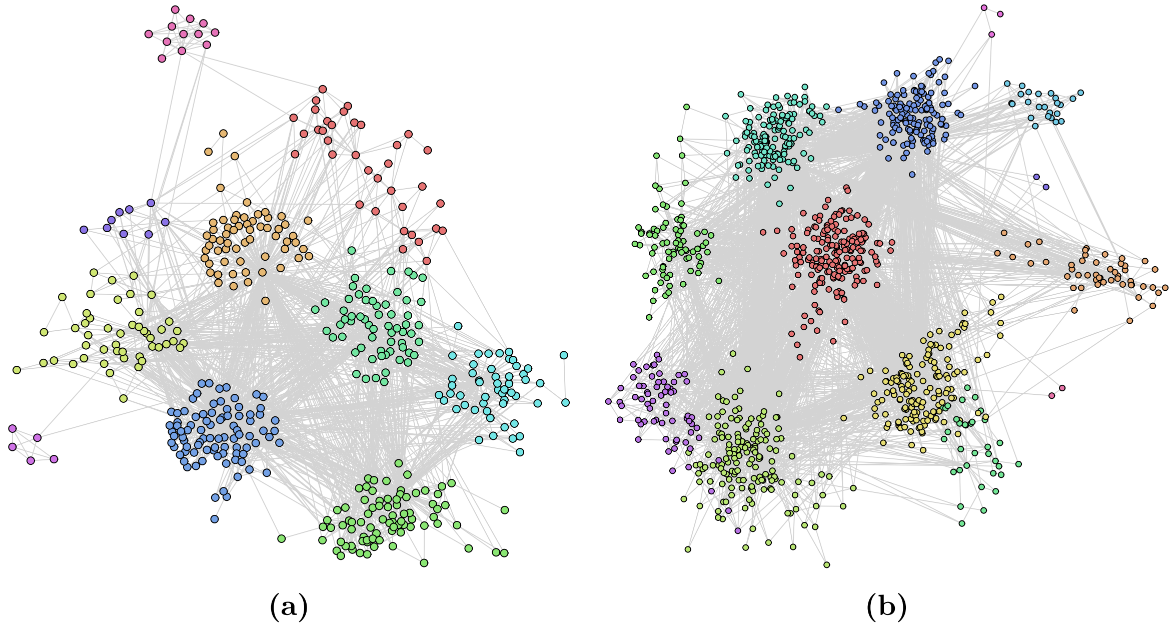


Figure 6.8: Community structures identified by Hybrid-IA on (a) *C. elegans MRN* and (b) *E. coli MRN* networks.

6.3.3 Functional Sensitivity of Community Detection

In order to uphold the efficiency and reliability of Hybrid-IA in detecting strong communities, a new evaluation metric has been considered. Thanks to the advantages offered by the synthetic networks (see Section 6.2.5), the *Normalized Mutual Information* (NMI) [48] has been taken into account, which is a widely used measure to compare community detection methods as it discloses the similarity between the genuine community (target) and the detected community structures. While the modularity allows for getting the measure of how cohesive the detected communities are, the NMI allows for assessing how similar they are concerning the real ones. Moreover, a more in-depth functional sensitivity analysis was conducted based on two other community structure similarity metrics: *Adjusted Rand Index* (ARI) [79], which is based on pairs counting for measuring the similarity, and *Normalized Variation of Information* (NVI) [101], which is based on the Shannon entropy and measures the lost and the gained information in changing from one clustering to another one. It is worth recalling that whilst in NMI and ARI the values as close as 1 indicate a strong similarity between the community detected and the real one (1 means identical communities), the NVI values, on the other hand, tend to 0 as the similarity between the compared communities increases (0 means identical communities). For this analysis, a new dataset of LFR instances has been generated, with the mixing parameter μ_t that ranges from 0.1 to 0.8. The three measures of functional sensitivity were computed on synthetic networks with 1000, 5000 and 10000 vertices.

In Tables 6.4-6.6, the Hybrid-IA outcomes on these new synthetic datasets are reported and compared to the ones obtained by Louvain. The features of the LFR networks tested are shown in the first column; for each of these parameters, 5 random instances have been generated. The values of modularity Q and clustering measures have been computed over 100 independent runs for both algorithms.

Analyzing the comparison, it is possible to see how Louvain outperforms Hybrid-IA in almost all networks with 1000 vertices with respect to the Q modularity metric,

Table 6.4: Functional sensitivity analysis of Hybrid-IA and Louvain on synthetic networks with 1000 vertices. NMI, ARI and NVI were considered as community structure similarity metrics.

	Hybrid-IA				Louvain			
$(\|V\|, k, \mu_t)$	Q	NMI	ARI	NVI	Q	NMI	ARI	NVI
$(1000, 15, 0.1)$	0.8608	**0.9951**	**0.9873**	**0.0098**	0.8608	0.9918	0.9785	0.0161
$(1000, 15, 0.2)$	0.7621	**0.9894**	**0.9716**	**0.0209**	0.7623	0.9807	0.9483	0.0377
$(1000, 15, 0.3)$	0.6646	**0.9862**	**0.9567**	**0.0271**	0.6651	0.9716	0.9175	0.0550
$(1000, 15, 0.4)$	0.5654	**0.9836**	**0.9490**	**0.0322**	0.5660	0.9691	0.9104	0.0598
$(1000, 15, 0.5)$	0.4670	**0.9847**	**0.9467**	**0.0301**	0.4688	0.9462	0.8274	0.1019
$(1000, 15, 0.6)$	0.3688	**0.9612**	**0.8664**	**0.0742**	0.3718	0.9113	0.7368	0.1627
$(1000, 15, 0.7)$	0.2712	**0.5467**	**0.2379**	**0.6220**	0.2675	0.4977	0.2168	0.6664
$(1000, 15, 0.8)$	0.2415	**0.1600**	**0.0219**	**0.9130**	0.2354	0.1536	0.0218	0.9167
$(1000, 20, 0.1)$	0.8606	**0.9980**	**0.9948**	**0.0040**	0.8607	0.9931	0.9842	0.0135
$(1000, 20, 0.2)$	0.7622	**0.9964**	**0.9918**	**0.0071**	0.7622	0.9914	0.9782	0.0170
$(1000, 20, 0.3)$	0.6656	**0.9921**	**0.9771**	**0.0156**	0.6658	0.9830	0.9525	0.0332
$(1000, 20, 0.4)$	0.5668	**0.9910**	**0.9707**	**0.0179**	0.5676	0.9656	0.8961	0.0664
$(1000, 20, 0.5)$	0.4685	**0.9829**	**0.9426**	**0.0337**	0.4700	0.9491	0.8363	0.0968
$(1000, 20, 0.6)$	0.3688	**0.9748**	**0.9038**	**0.0491**	0.3712	0.9263	0.7641	0.1370
$(1000, 20, 0.7)$	0.2714	**0.9244**	**0.7561**	**0.1399**	0.2737	0.8230	0.5403	0.3002
$(1000, 20, 0.8)$	0.2169	0.1708	0.0288	0.9064	0.2069	**0.1793**	**0.0300**	**0.9012**

and in all instances with 5000 and 10000 vertices. On the other hand, though, Hybrid-IA outperforms Louvain in all networks with respect to the NMI index, except just for one $(1000, 20, 0.8)$.

This gap is due to the combination between the random search and local search that, together with the diversity produced by the immune operators, requires a longer convergence time than Louvain. Indeed, Louvain is a multilevel algorithm that although obtains good modularity values, aggregates too much the communities bypassing, then, the real community structures of the networks. These results prove, therefore, a better ability of the hybrid immune algorithm proposed in detecting communities closer to the true ones, than the greedy optimization algorithm. Importantly, although modularity assesses the cohesion of the communities detected, maximizing Q might not correspond to detecting true communities. Indeed, as also asserted in [63], in maximizing modularity is possible to fail to identify smaller communities due to the degree of interconnectedness of the communities.

Table 6.5: Functional sensitivity analysis of Hybrid-IA and Louvain on synthetic networks with 5000 vertices. NMI, ARI and NVI were considered as community structure similarity metrics.

	Hybrid-IA				Louvain			
$(\|V\|, k, \mu_t)$	Q	NMI	ARI	NVI	Q	NMI	ARI	NVI
$(5000, 20, 0.1)$	0.8923	**0.9988**	**0.9940**	**0.0024**	0.8934	0.9586	0.8194	0.0794
$(5000, 20, 0.2)$	0.7927	**0.9965**	**0.9816**	**0.0070**	0.7949	0.9394	0.7302	0.1142
$(5000, 20, 0.3)$	0.6929	**0.9965**	**0.9808**	**0.0070**	0.6960	0.9252	0.6678	0.1392
$(5000, 20, 0.4)$	0.5931	**0.9948**	**0.9711**	**0.0104**	0.5976	0.9065	0.5941	0.1709
$(5000, 20, 0.5)$	0.4936	**0.9951**	**0.9728**	**0.0098**	0.5003	0.8779	0.4959	0.2177
$(5000, 20, 0.6)$	0.3939	**0.9966**	**0.9797**	**0.0068**	0.4030	0.8474	0.4048	0.2648
$(5000, 20, 0.7)$	0.2932	**0.9927**	**0.9539**	**0.0145**	0.3056	0.8145	0.3327	0.3130
$(5000, 20, 0.8)$	0.2084	**0.3285**	0.0176	**0.8027**	0.2102	0.2634	**0.0195**	0.8481
$(5000, 25, 0.1)$	0.8922	**0.9993**	**0.9966**	**0.0013**	0.8925	0.9770	0.8928	0.0449
$(5000, 25, 0.2)$	0.7925	**0.9988**	**0.9935**	**0.0024**	0.7936	0.9527	0.7821	0.0902
$(5000, 25, 0.3)$	0.6929	**0.9986**	**0.9921**	**0.0029**	0.6948	0.9348	0.7026	0.1225
$(5000, 25, 0.4)$	0.5931	**0.9987**	**0.9915**	**0.0027**	0.5966	0.9125	0.6158	0.1609
$(5000, 25, 0.5)$	0.4934	**0.9955**	**0.9747**	**0.0089**	0.4983	0.8907	0.5366	0.1970
$(5000, 25, 0.6)$	0.3939	**0.9950**	**0.9666**	**0.0100**	0.4008	0.8621	0.4473	0.2424
$(5000, 25, 0.7)$	0.2940	**0.9951**	**0.9653**	**0.0097**	0.3037	0.8285	0.3604	0.2927
$(5000, 25, 0.8)$	0.1872	**0.6067**	0.0667	**0.5607**	0.1942	0.5654	**0.1065**	0.6058

By analyzing these tables it is possible to assert the same statement made for NMI for the other two evaluation metrics, as well. Indeed, appears clear as Hybrid-IA achieves better metric values than Louvain in all three sensitivity measures. It follows therefore that, albeit Louvain achieves slightly better modularity values in almost all networks considered (mainly on the larger ones), Hybrid-IA instead is able to detect communities strongly similar to the real ones outperforming Louvain in all three sensitivity metrics. It is worth pointing out how Hybrid-IA obtains NMI and ARI values close to 1 in almost all tested networks, with particular reference to the largest one ($|V| = 10000$).

In Figures 6.9a are shown the curves of the NMI, ARI and NVI indices for all LFR benchmarks with 1000 vertices. From these plots can be observed that the two curves have the same trend: for low values of μ_t (≤ 0.3), both algorithms obtain similar results, and the NMI curves grow as a common line, while as the μ_t parameter increases, the gap between Hybrid-IA and Louvain begins to be more consistent. However note

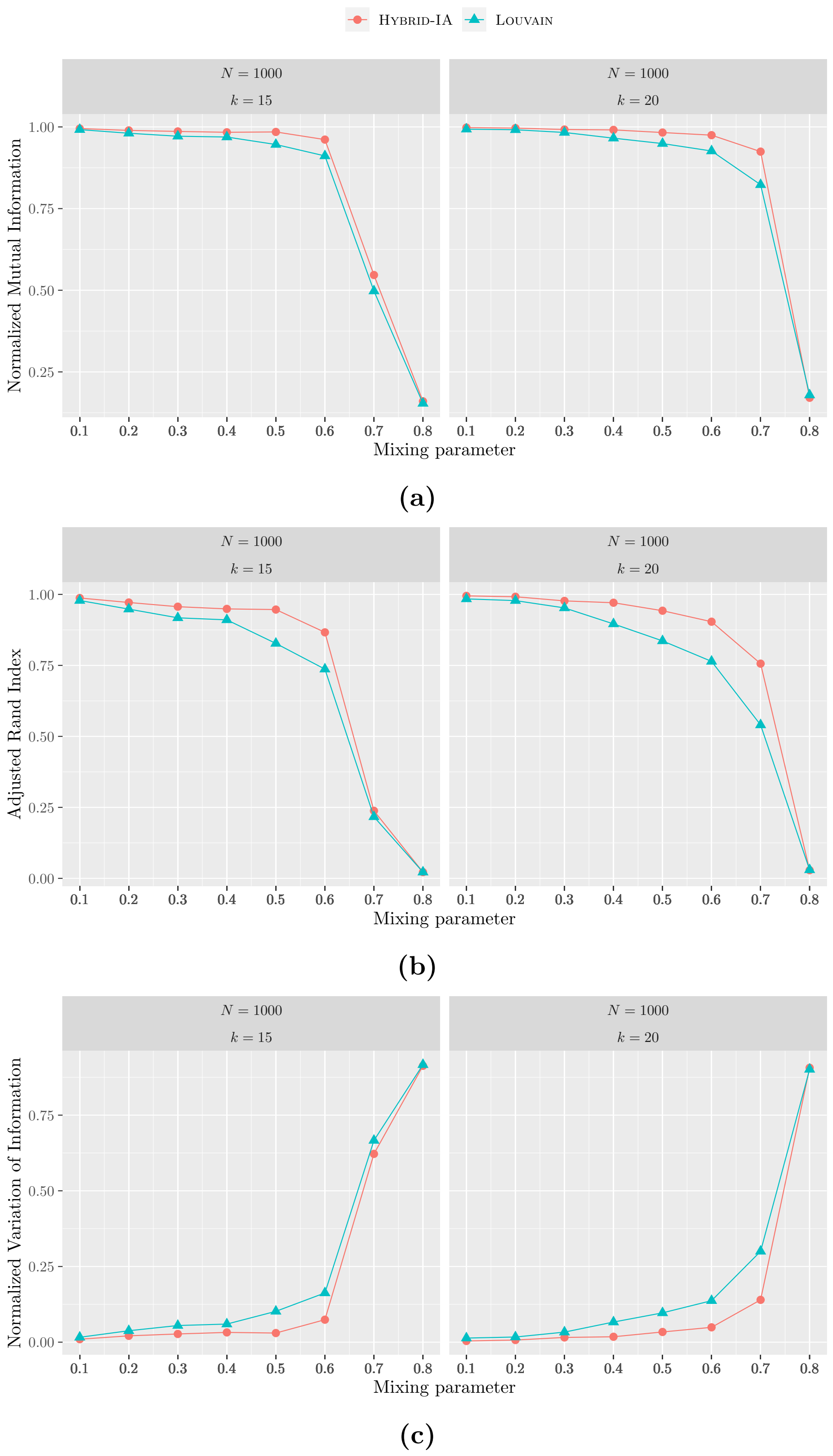
Table 6.6: Functional sensitivity analysis of Hybrid-IA and Louvain on synthetic networks with 10000 vertices. NMI, ARI and NVI were considered as community structure similarity metrics.

	Hybrid-IA				Louvain			
$(\|V\|, k, \mu_t)$	Q	NMI	ARI	NVI	Q	NMI	ARI	NVI
$(10000, 20, 0.1)$	0.8938	**0.9995**	**0.9982**	**0.0010**	0.8945	0.9686	0.8874	0.0609
$(10000, 20, 0.2)$	0.7938	**0.9981**	**0.9925**	**0.0037**	0.7951	0.9538	0.8198	0.0883
$(10000, 20, 0.3)$	0.6940	**0.9980**	**0.9925**	**0.0040**	0.6960	0.9407	0.7563	0.1119
$(10000, 20, 0.4)$	0.5941	**0.9977**	**0.9900**	**0.0045**	0.5972	0.9202	0.6682	0.1478
$(10000, 20, 0.5)$	0.4942	**0.9988**	**0.9945**	**0.0024**	0.4986	0.8982	0.5793	0.1847
$(10000, 20, 0.6)$	0.3943	**0.9996**	**0.9974**	**0.0008**	0.4004	0.8720	0.4872	0.2269
$(10000, 20, 0.7)$	0.2909	**0.9847**	**0.8556**	**0.0301**	0.3013	0.8287	0.3737	0.2925
$(10000, 20, 0.8)$	0.2064	**0.2621**	**0.0085**	**0.8491**	0.2094	0.1704	0.0079	0.9068
$(10000, 25, 0.1)$	0.8937	**0.9995**	**0.9984**	**0.0010**	0.8940	0.9792	0.9253	0.0407
$(10000, 25, 0.2)$	0.7940	**0.9993**	**0.9968**	**0.0014**	0.7947	0.9627	0.8524	0.0719
$(10000, 25, 0.3)$	0.6941	**0.9990**	**0.9955**	**0.0020**	0.6955	0.9497	0.7946	0.0958
$(10000, 25, 0.4)$	0.5942	**0.9992**	**0.9969**	**0.0015**	0.5964	0.9320	0.7179	0.1273
$(10000, 25, 0.5)$	0.4944	**0.9982**	**0.9917**	**0.0036**	0.4978	0.9083	0.6199	0.1679
$(10000, 25, 0.6)$	0.3943	**0.9984**	**0.9897**	**0.0032**	0.3989	0.8841	0.5313	0.2077
$(10000, 25, 0.7)$	0.2941	**0.9981**	**0.9860**	**0.0038**	0.3008	0.8516	0.4299	0.2584
$(10000, 25, 0.8)$	0.1849	**0.4316**	0.0227	**0.7245**	0.1867	0.3352	**0.0247**	0.7985

that, when the mixing parameter assumes higher values, the generated LFR networks have community structures not well-defined, resulting, then, in low NMI values for both algorithms. In Figures 6.10a and 6.11a, instead, are shown the NMI curves for the LFR networks with 5000 and 10000 vertices. For these instances, on the other hand, the difference between Louvain and Hybrid-IA is much more substantial, and it is evident even at low values of μ_t.

6.4 Local Search Position Analysis

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 to new and unknown problems, and, in general, to all those problems whose knowledge about their features and structures is very limited. Among the evolutionary computation methodologies, the immune-inspired

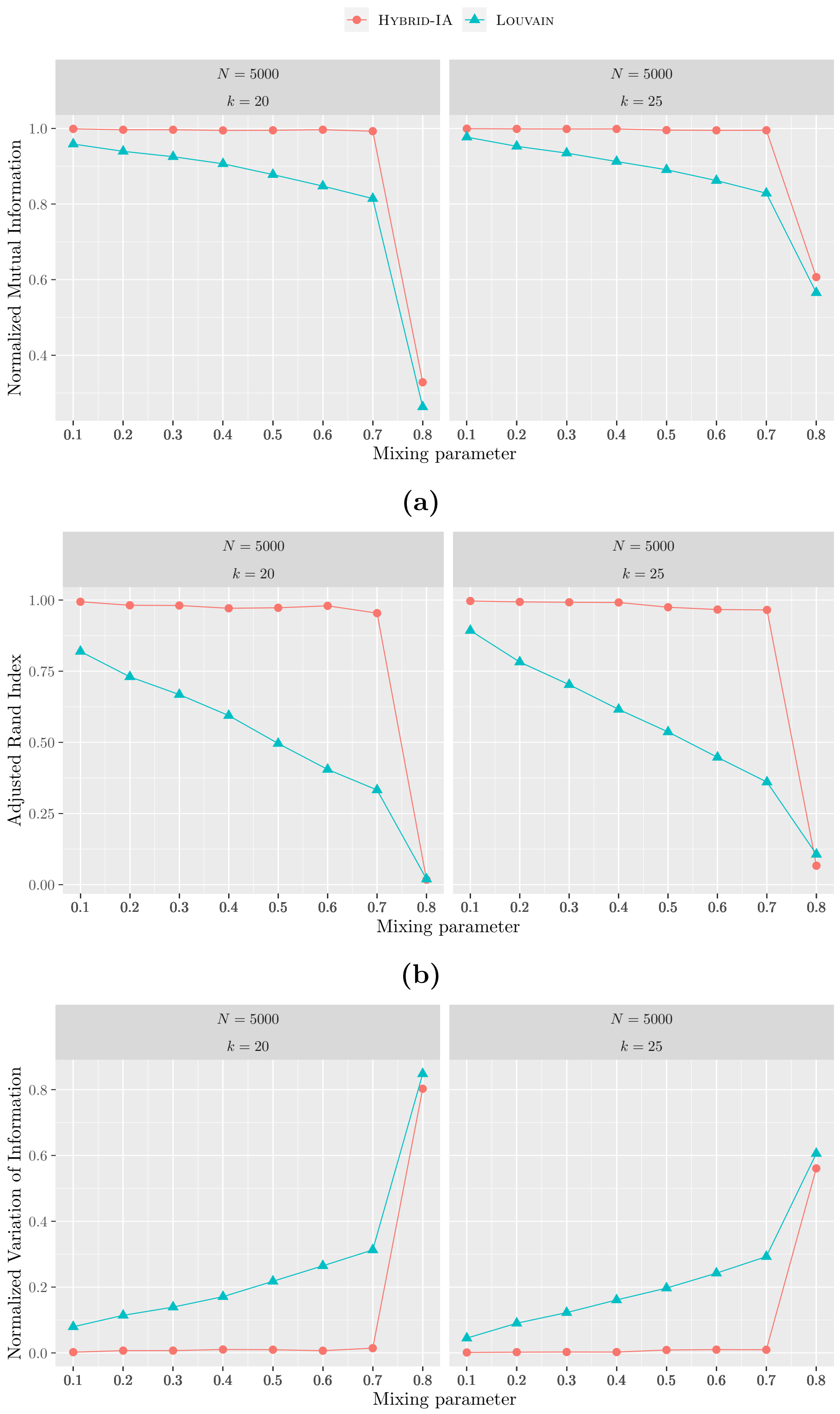


(a)

(b)

(c)

Figure 6.9: Comparative evaluation of the performances of Hybrid-IA and Louvain on the LFR instances with 1000 vertices and average degree 15 and 20. The plots show (a) the Normalized Mutual Information, (b) the Adjusted Rand Index and (c) the Normalized Variation of Information as function of the mixing parameter. Each point corresponds to an average over 5 graph realizations and 100 runs.

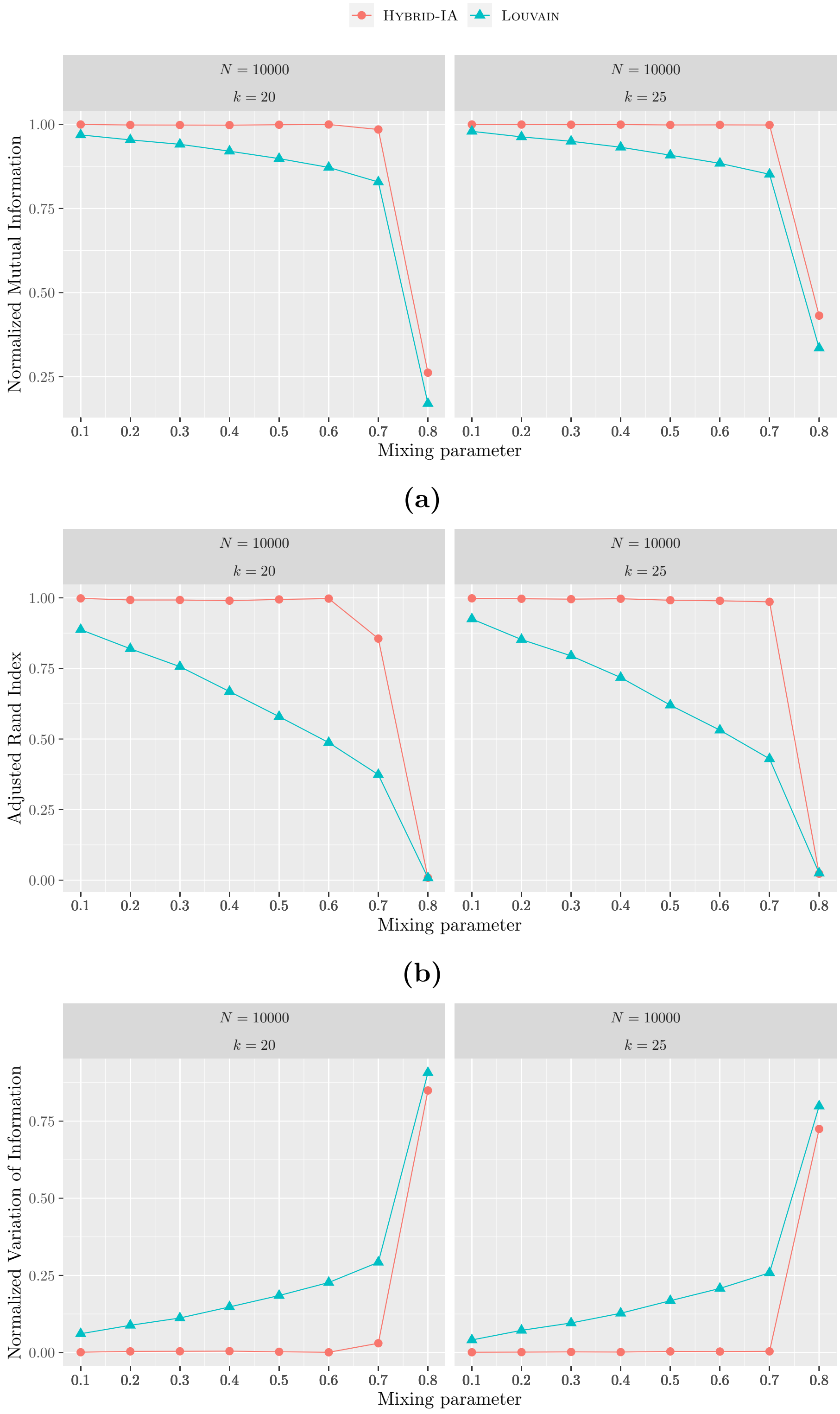


(a)

(b)

(c)

Figure 6.10: Comparative evaluation of the performances of Hybrid-IA and Louvain on the LFR instances with 5000 vertices and average degree 20 and 25. The plots show (a) the Normalized Mutual Information, (b) the Adjusted Rand Index and (c) the Normalized Variation of Information as function of the mixing parameter. Each point corresponds to an average over 5 graph realizations and 100 runs.



(a)

(b)

(c)

Figure 6.11: Comparative evaluation of the performances of Hybrid-IA and Louvain on the LFR instances with 10000 vertices and average degree 20 and 25. The plots show (a) the Normalized Mutual Information, (b) the Adjusted Rand Index and (c) the Normalized Variation of Information as function of the mixing parameter. Each point corresponds to an average over 5 graph realizations and 100 runs.

algorithms represent a powerful algorithmic class, which takes inspiration from the principles and dynamics of the biological immune system (IS). What makes the IS very interesting and a source of inspiration from a computational perspective is its ability in learning, detecting, and recognizing foreign and dangerous entities [64].

However, although many methodologies inspired by biology and nature have been developed, and 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 [13]. 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 the 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 to 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 section, we want to investigate when is better to perform the Local Search (LS), and if, in the overall, replacing the original solution with the revised one by LS is the best choice. In light of this Hybrid-IA (described in Section 6.1) has been taken into account in an attempt to answer these questions. Hybrid-IA has been considered as it was successfully applied in several and various combinatorial optimization problems [43, 46, 47, 41]. Thus, the effect of the local search on the performances of Hybrid-IA has been investigated, considering three different positions in the evolutionary cycle in which 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 6.1); 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.

The main idea behind the Local Search operator is to refine and improve in a deterministic way the solutions produced by the stochastic mutation operator. In this study, the effect and impact of the position where to run the local search within Hybrid-IA are inspected (see Section 6.1 and Algorithm 6.1). 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.

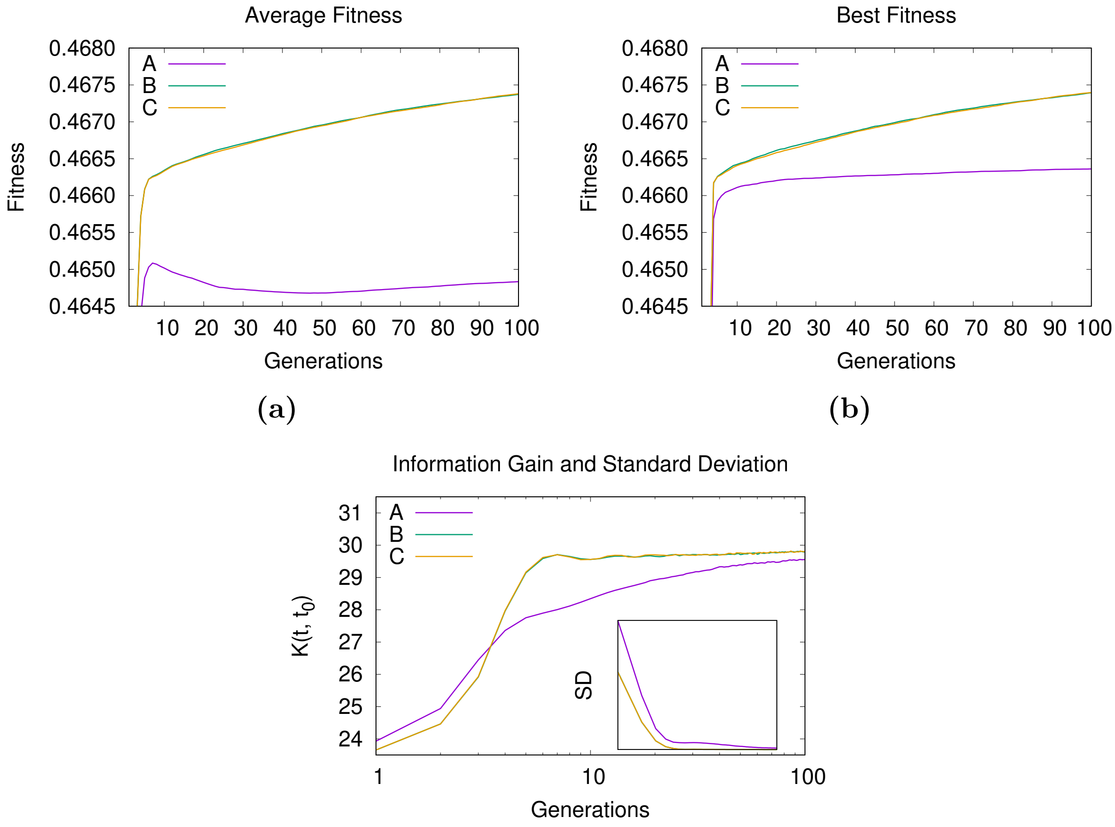
- **Method B**: applying LS to the population generated by the hypermutation operator, where each revised individual replaces the hypermutated one, maintaining the same population. In this way, it is applied to a wider set of solutions generated from the current ones through mutation allowing a better 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 the 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.

6.4.1 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 [125], 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 the LFR algorithm, proposed in [92, 91], and have been used because they allow us to perform our study on different complexity scenarios thanks to their diverse features. Note that the validity of this benchmark is given by faithfully reproducing the key features of real graph communities. In particular, networks with number of vertices 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 $\tau_2 = 1$. The 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 vertex'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 the number of edges that a vertex shares with vertices 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 $d = 100$; the 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 [111, 47, 41], and from preliminary experiments carried out. The 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 methods 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



(a) (b)

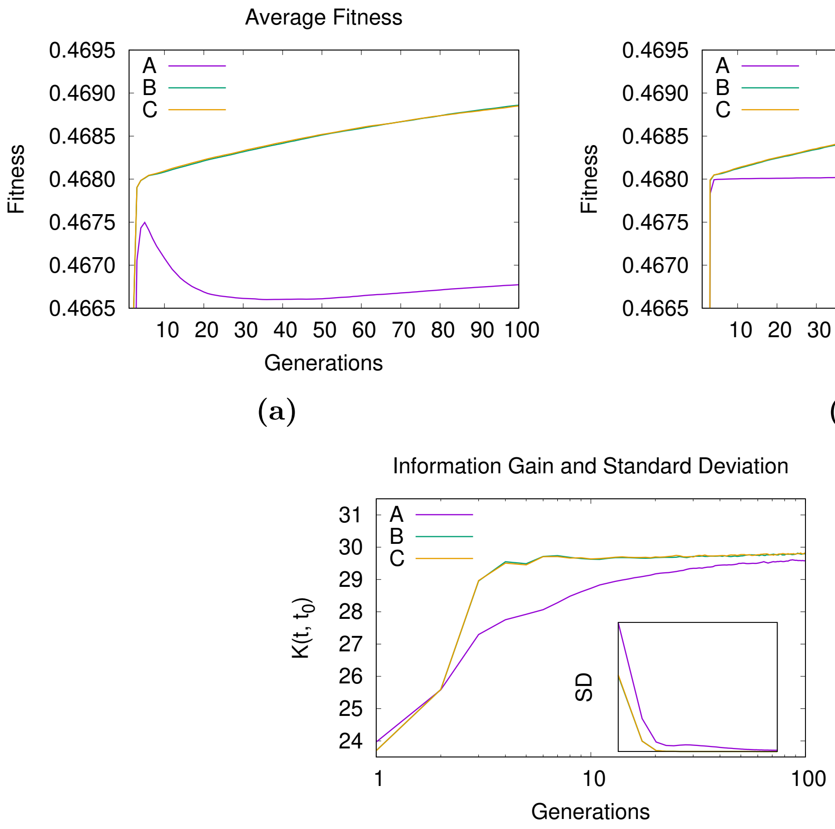
(c)

Figure 6.12: Convergence behaviour of the three methods on the `LFR(1000,15,0.5)` instance. (a) Average and (b) best fitness value of the population versus generations. (c) Information gain and standard deviation.

been considered as evaluation metric. This entropic function measures the quantity of information the system discovers during the learning phase (see [89, 90, 43]).

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 6.12 and 6.13 are shown the convergence plots for the LFR instances with 1000 vertices 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 improving 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 the average of the population. The average fitness curve is very close



(a)

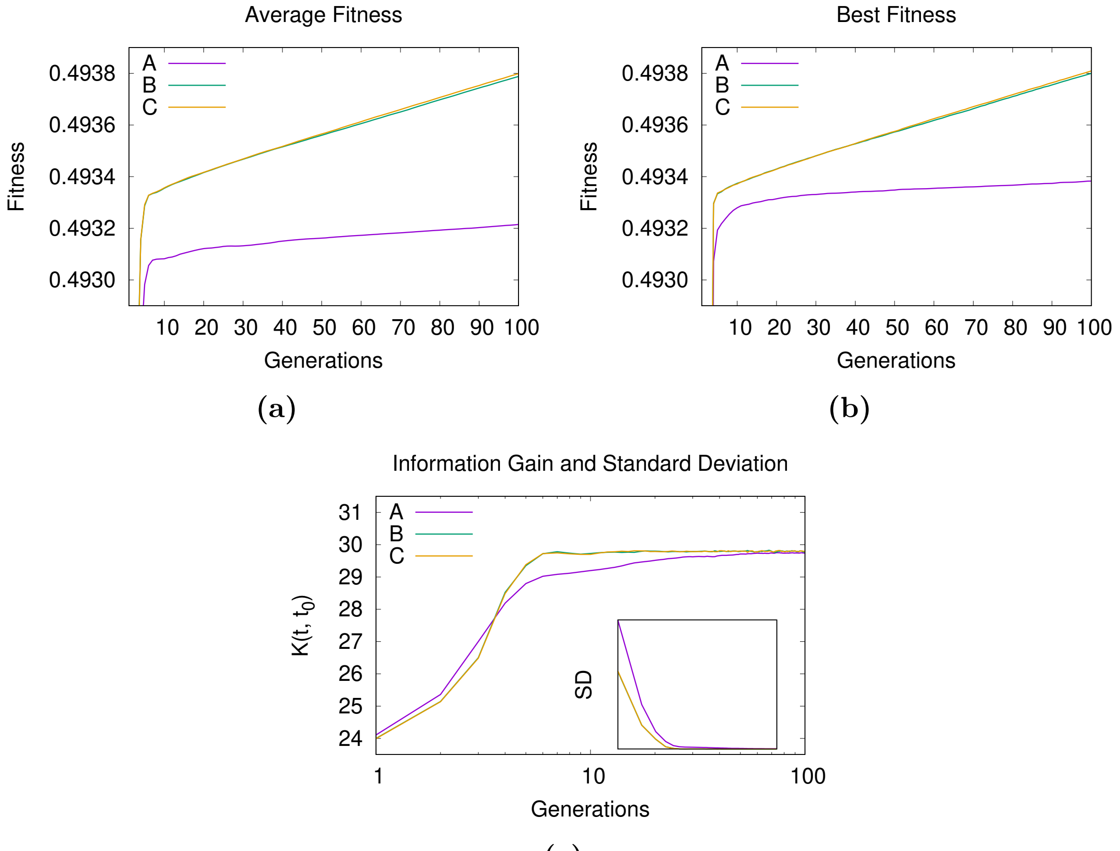
(b)

(c)

Figure 6.13: Convergence behaviour of the three methods on `LFR(1000,20,0.5)` instance. (a) Average and (b) best fitness value of the population versus generations. (c) Information gain and standard deviation.

to the curve of the best solution, indicating 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 6.12c and 6.13c), while method A needs more generations to converge to the same value reached by the other two methods.

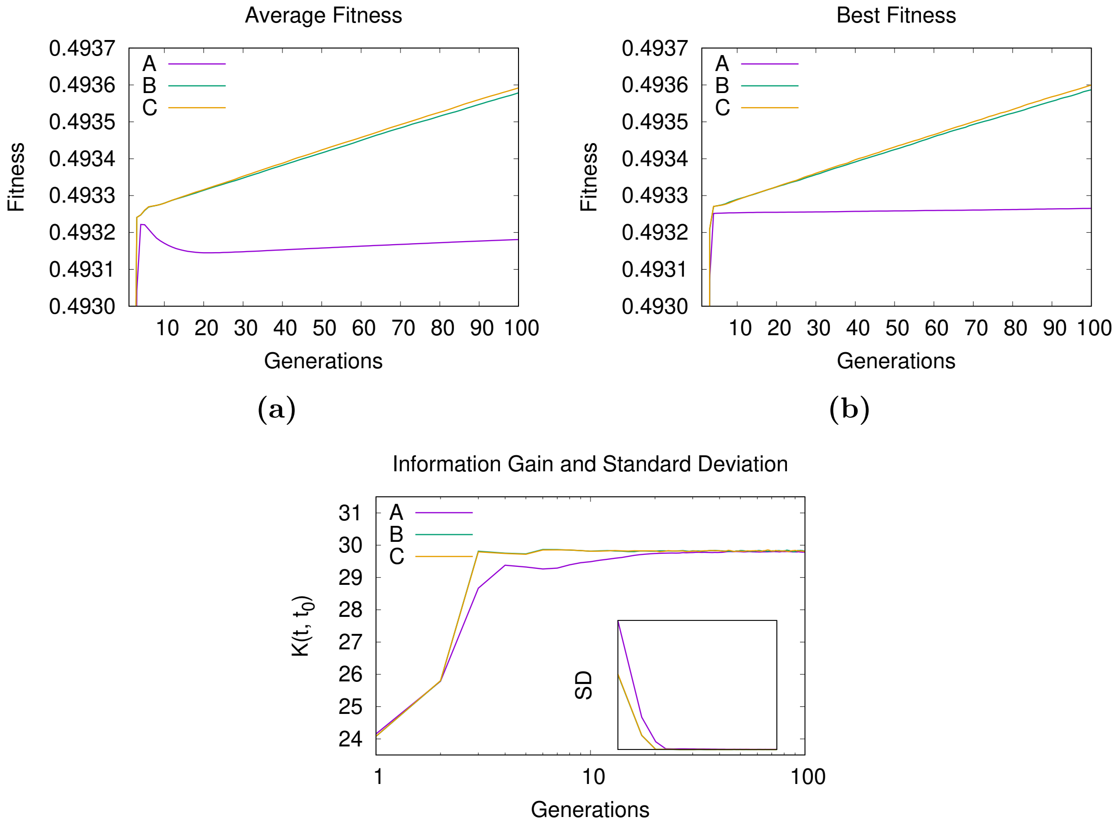
The same situation is obtained in the networks with 5000 vertices and average degree k equal to 20 and 25 (Figures 6.14 and 6.15). Also in these plots, method A has a much slower convergence than the other two methods and maintains a certain degree of diversity within the population, demonstrated by 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)**

Figure 6.14: Convergence behavior of the three methods on `LFR(5000,20,0.5)` instance. (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. Table 6.7 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), the 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 vertices, all three methods reach what is most likely the maximum modularity value, detecting the same number of communities. On the other hand, for the instances with 500 vertices, 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



(a) **(b)**

(c)

Figure 6.15: Convergence behavior of the three methods on `LFR(5000,25,0.5)` instance. (a) Average and (b) best fitness value of the population versus generations. (c) Information gain and standard deviation.

16.8 for both method B and C). The difference in modularity becomes greater as the number of vertices increases. For the instances with 1000 vertices, method A reaches a lower modularity value than the other two methods (about 10^{-3} on average for both instances), as observed in the respective 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 same results can be observed for the network with 5000 vertices, where method A is behind the other two methods in terms of modularity, although with a lower gap with respect to the instances with 1000 vertices (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 vertices), on the networks with 1000 and 5000 vertices, the number of communities found by methods B and C is different. Finally,

Table 6.7: Comparative results of the three methods on LFR benchmark instances. The results are averaged on 5 random instances and calculated over 100 independent runs.

	A			B			C		
Instance	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

from the computational time point of view, methods B and C, as expected, take about 90% more time than method A, but they allow a better exploration of the search space 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 6.8 and 6.9. As described above, the mixing parameter μ_t identifies the relationships between the communities, that is the ratio between the vertex's degree internal to the community, and the external one. In this way, it is possible to carry out a comparative analysis in different scenarios, each of which was designed as realistic as possible ($\mu_t = \{0.1, 0.2, \ldots, 0.8\}$). Table 6.8 reports the experimental results obtained by the three methods on networks with 300 and 500 vertices. Focusing on the first one, that is the network with 300 vertices, it is possible to note how the three methods are equivalent in 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 terms of the modularity values and number of communities discovered. A similar behaviour can be observed also on the network with 500 vertices, although the threshold, where the three methods are equivalent, decreases to $\mu_t \leq 0.4$ when the average degree k of the vertices is 15. What is more interesting to note is that the method C considerably outperforms not only the method A, which is to be

Table 6.8: Comparative results of the three methods on synthetic networks with 300 and 500 vertices.

	A			B			C		
Instance	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

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 vertices average degree and, primarily, the number of external links increases, 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.

Table 6.9: Comparative results of the three methods on synthetic networks with 1000 and 5000 vertices.

	A			B			C		
Instance	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

From Table 6.9, where are showed the experimental results on the networks with

1000 and 5000 vertices, 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 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 for solving this kind of task, dues to their feature of producing higher diversity in the population.

6.4.2 Functional Sensitivity Analysis

As the last step of this work, in this subsection, the investigation of the sensitivity of the three community detection methods is reported from a functional perspective. The main aim of this analysis is to measure the similarity between the detected communities and the original ones. For doing this, commonly used community structure similarity metrics have been considered: (1) *Normalized Mutual Information* (NMI) [48], 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) [79], which focuses on the pairwise agreement: for each possible pair of elements it evaluates how similarly the two partitions treat them; and (3) *Normalized Variation of Information* (NVI) [101], 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 6.10 (only for 1000 and 5000 vertices). From this investigation, clearly appears that the method A outperforms the other two in almost all tests performed, uncovering, consequently, more similar

Table 6.10: Comparative evaluation of the performances of the three methods on the LFR instances.

	NMI			ARI			NVI		
Instance	*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

communities to the target/real ones, in opposite to the outcomes obtained with respect to 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 [63].

6.5 Conclusions

Being able to efficiently analyze complex networks is one of the most crucial and central issue in many areas, including systems biology, since through them is possible to understand and identify the dynamics and structures of molecular interactions. In general, disease phenotypes are generally caused by the failure of modules of genes that often have similar biological roles. In light of this, being able to detect elements of a network that have characteristics in common, or similar functions, plays a key and useful role in providing insights into the biological functionality of these elements. Therefore, developing efficient and robust algorithmic methods able to uncover such elements in biological networks may help in detecting those groups of genes that are the cause of disease, and, consequently, useful in the development of specific and targeted drugs. The problem to identify modules in a network is known as *community detection*.

A Hybrid Immune Algorithm, called Hybrid-IA, was designed for the community detection problem and was tested on several large biological networks. The strength of Hybrid-IA is given not only by the immune operators (cloning, hypermutation and aging) but also by a specially designed Local Search, which aims to speed up the convergence of Hybrid-IA towards promising regions. Basically, it attempts deterministically to move a vertex from the belonging community to another within its neighbours with the purpose to refine and improve the solutions discovered.

For assessing the robustness of Hybrid-IA, a comparison with other metaheuristics, hyper-heuristics and the well-known algorithm Louvain has been performed. Such a comparison has been conducted based on modularity function maximization. However, due to the limitation of the modularity optimization in detecting communities that are comparatively small, the Hybrid-IA performances have been also evaluated with respect to the *Normalized Mutual Information* (NMI), *Adjusted Rand Index* (ARI) and *Normalized Variation of Information* (NVI), which are evaluation metrics commonly used in community detection that simply assess how similar the communities discovered are with respect to the real ones. Inspecting, in the overall, all outcomes obtained and all comparisons performed clearly emerges how Hybrid-IA outperforms all metaheuristics and hyper-heuristics compared in terms of best and mean modularity values. Focusing only on the comparison with Louvain is possible to assert that although Hybrid-IA finds slightly lower modularity values, it is still able to detect more similar communities to the real ones with respect to those discovered by Louvain.

Moreover, three different positions where run the local search within Hybrid-IA, have been investigated in order to ascertain which of the three acts best on the algorithm's performance. A comparison between the three methods has been conducted with respect to the solution quality found and the learning process quality. Several artificial networks were generated ($|V| \in \{300, 500, 1000, 5000, 10000\}$) through which was possible to inspect the three methods in various complex 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 helps the algorithm, especially on larger and complex networks.

7

Multi-level Optimization

The multi-level approach is an optimization technique used to improve a community detection algorithm, both in terms of the objective function and computational cost. This approach consists in creating a new graph in which the vertices are the communities of the partial solution found by the base algorithm, while edges between communities are merged together with a weight given by the sum of the edges between vertices in the corresponding two communities. Edges between vertices in the same community are translated in self-loops in the new graph. In this way, the modularity value for the partition does not change in the new graph [3]. In Figure 7.1 is shown the creation of a new level starting from the partition found on the current graph. The reduced graph is then passed as input to the base algorithm to compute the next solution. These steps are repeated until no further improvement can be achieved or for a certain amount of time. At each iteration, the size of the reduced graph decreases and consequently the efficiency of the base algorithm is greatly improved.

In a metaheuristic algorithm, the implementation of multi-level optimization could lead to wrong solutions, because some parts of the solution would be *locked* and any fur-

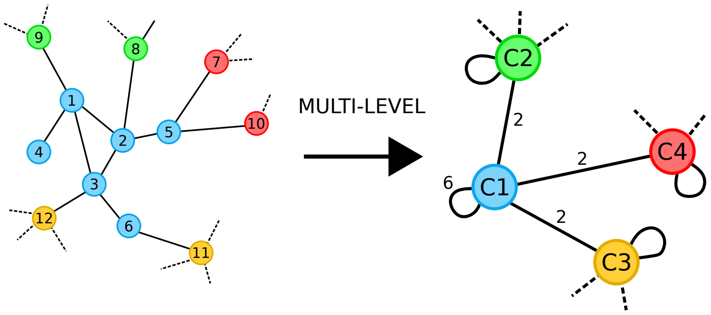


Figure 7.1: Creation of the community network by the multi-level optimization. Communities will be translated in vertices in the next level, while edges are merged together with a weight that is the number of edges that those communities share. Self-loops identify internal edges.

ther improvement to the solution would be done only by fusing together the remaining vertices. After a few iterations, the graph will be reduced to a small number of vertices where any combination between them would not result in a modularity improvement, but the solution would remain of low quality. In light of this, in the following sections we propose two multi-level approaches: the first one uses a backtracking mechanism to give the underlying base metaheuristic algorithm a chance to improve specific parts of the global solution; the second one uses a heuristic to merge vertices together.

7.1 Random and Smart Explosion

The first approach proposed consists of the classical multi-level optimization with a backtracking mechanism that brings the algorithm back to a previous level when there is no improvement of the modularity value for a certain number of levels. Then to the base algorithm, we provide the graph of the level with the best partition found in which some vertices, that represent communities, are disaggregated from the original graph. This allows freeing vertices that had been blocked in an earlier wrong solution trying to repair communities not well-formed. The communities which *explode* are randomly selected from those in the best partition and the number is given by a user-defined parameter N_e. Usually, the number of communities to disaggregate is kept low in order to avoid degrading too much the current solution and letting the underlying base algorithm focus mainly on those vertices that are now free. After the roll-back to a previous level, the multi-level approach continues in a classical way until a new stagnation of the modularity value occurs. Then the back-tracking mechanism is applied again and this process is repeated a certain number of times. Finally, the algorithm stops and returns the best solution found.

A complete disaggregation of one or more communities affects the performance of the base algorithm, disrupting correct parts of the current solution and increasing the number of vertices to evaluate. A further improvement of this approach consists in a *smart explosion* of the communities, in which only a subset of vertices is disaggregated. In this way, the method disaggregates only critical vertices, that is the vertices that lie on the boundary of the community. Critical vertices are identified using the internal-total degree ratio:

$$\frac{k_i^{int}(C)}{k_i} < T_e, \tag{7.1}$$

where $k_i^{int}(C)$ is the sum of the weights of edges that vertex i shares with other vertices belonging to the same community C, k_i is the sum of weights of all incident edges of vertex i and T_e is a user-defined threshold. In Figure 7.2 is shown the application of the *smart explosion* approach.

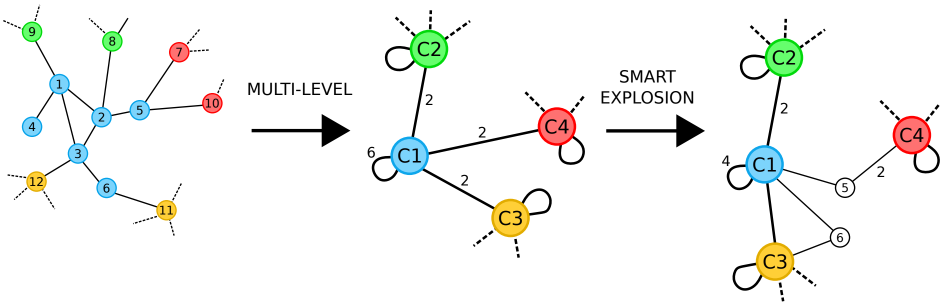


Figure 7.2: Multi-level with smart explosion approach. In this case, vertices 5 and 6 have an internal-total degree ratio less or equal than 0.5 and the method disaggregates them from their own community in order to let the base algorithm relocate them to a better community.

7.2 Smart Merge

A naive multi-level approach, that blindly merges all vertices in their respective supposed communities, could lead to wrong associations vertex-community, as described before. The second proposed approach modifies the multi-level optimization introducing a *quality-based aggregation*. In particular, during the aggregation phase, only those vertices belonging to the same community and with a high internal-total degree ratio (Equation 7.1) will be merged together. In this way, the vertices that are supposed to be already associated with the correct community and that will not change in subsequent iterations will be merged together, reducing the size of the graph and the complexity of the base algorithm. On the other hand, critical vertices are kept free and can be moved to the correct community by the underlying base algorithm. In Figure 7.3 is shown how the multi-level with smart merge mechanism works. This approach is useful and efficient with a base algorithm that finds good partitions in a relatively small time. Algorithms that tend to converge slowly starting from low-quality solutions, do not receive a significant improvement by this approach because the aggregation heuristic used in the smart merge (that depends on the threshold T_m) decreases the graph size slowly, affecting the overall computational time.

However, although this approach allows reaching high values of modularity, by in-

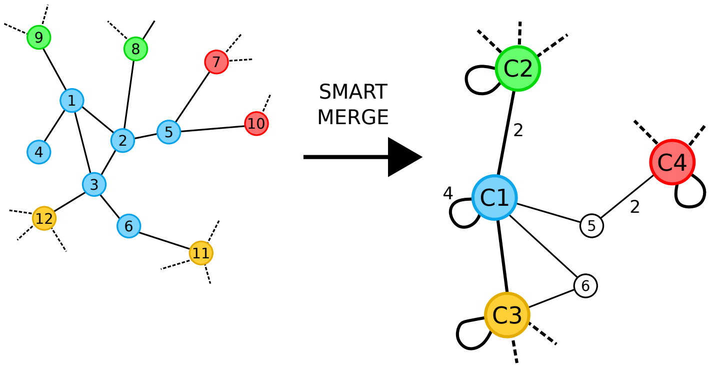


Figure 7.3: Creation of the network of the next level by the multi-level optimization with smart merge mechanism. In this case vertices 5 and 6 are kept free because they share a number of links with other communities greater than or equal to those they share with their own community.

specting the graphical representation of the detected communities (Figure 7.4a), it is possible to note how a single community is composed by elements disconnected from each other (see inset plot in Figure 7.4a). This happens because these disconnections are disregarded by *smart merge* approach, as it asserts the goodness of a vertex by checking only if its links are inside or outside. In light of this, to overcome this limitation, it was enough to add a control on the communities detected by the basic version of the algorithm (Hybrid-IA), which divides the clusters into their connected components. Through this simple check, the detected communities appear to be more compact graphically (Figure 7.4b), as well as reaching higher modularity values (see Table 7.3, Section 7.3). This variant is called *smart merge + check connect*.

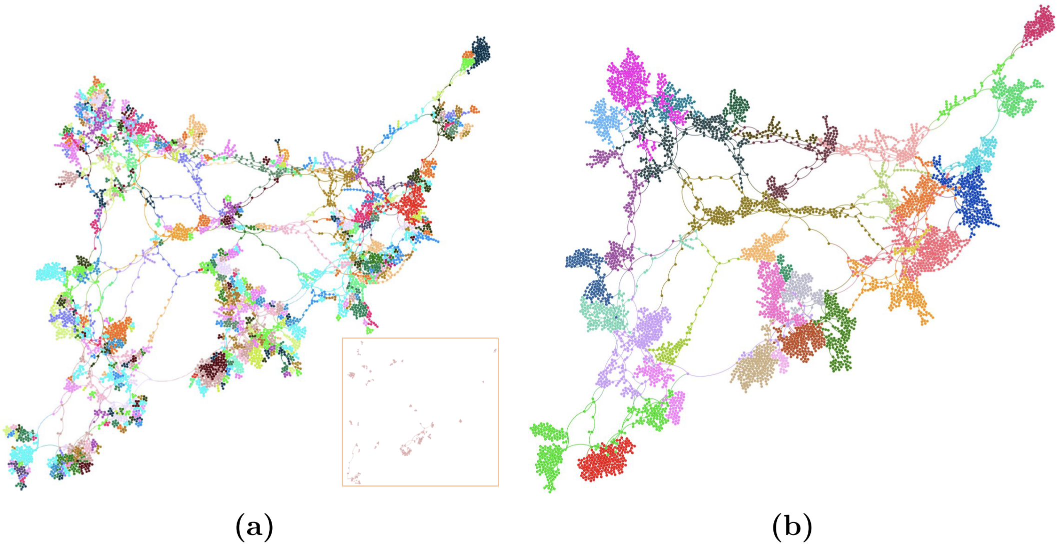


Figure 7.4: Communities detected on *Power* network by (a) the *smart merge* approach and (b) the *smart merge* considering the connected components.

7.3 Experimental Results

To assess the robustness and efficiency of the proposed multilevel approaches, three well-known benchmark networks were used, which are reported in Table 7.1. Obviously, the comparison with the relative basic versions is also presented in this section to check the improvements produced by the proposed approaches. In particular, Random-IA has been considered as the basic algorithm for the random and *smart explosion* approaches due to its stochastic nature; in this way, it can repair a small region of the network disaggregated by the explosion mechanism. On the other hand, as described in Section 7.2, Hybrid-IA has been used as underlying metaheuristic for the *smart merge* approach, because this algorithm reaches in just a few iterations solutions with high modularity value. Consequently, the backtracking approaches developed in *random* and *smart explosion*, if applied on Hybrid-IA should disaggregate a high number of communities at each stagnation of modularity, and then correct/repair all communities, increasing however the network size, and therefore considerably slowing down the convergence of the entire algorithm.

Table 7.1: The benchmark networks used in the experiments.

Name	*Description*	$\|V\|$	$\|E\|$
Email [74]	University e-mail network	1133	5451
Yeast [25]	Protein-protein interaction network in budding yeast	2284	6646
Power [146]	Topology of the Western States Power Grid of the US	4941	6594

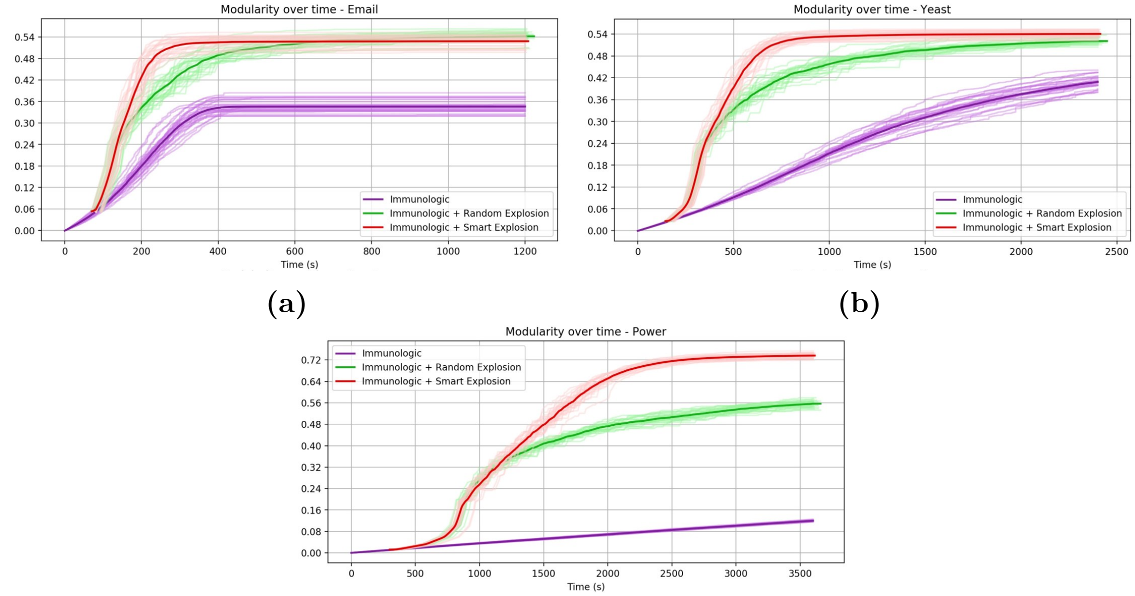
For all the experiments, both versions use the same parameter configurations, specifically a population of $d = 100$ solutions, a duplication factor $dup = 2$, $\tau_B = 20$ as the maximum age allowed, and a mutation shape $\rho = 1.0$. Due to the different algorithmic structures of the two versions, a different number of iterations $MaxGen$ was considered. In particular, for Random-IA we set $MaxGen = 1000$ for each level, while in Hybrid-IA the number of iterations is related to the size of the network: $MaxGen$ starts from 50 iterations and progressively decreases proportionally to the size of the network, to a minimum of 10 iterations.

For the multi-level optimization process, the *random explosion* reverts just $N_e = 1$ community to the original network, while the *smart explosion* approach disaggregates $N_e = 2$ communities using a threshold $T_e = 0.5$. The multi-level optimization with the *smart merge* mechanism instead uses a $T_m = 0.5$ to construct the network for the next level. Although multi-level optimization can stop its execution when it detects a modularity stagnation, for an easier comparison all algorithms were run 30 times for each instance and for a prefixed CPU time. In particular, in *random explosion* and *smart explosion*, which use Random-IA as basic algorithm, the CPU time limit was fixed, respectively, to 1200 seconds for *Email*, 2400 seconds for *Yeast* and 3600 seconds for *Power*. In *smart merge* approach, which uses Hybrid-IA as the underlying basic algorithm, the CPU time limit was fixed to 120 seconds for *Email*, 900 seconds for *Yeast* and 3600 seconds for *Power*.

7.3.1 Results

The first analysis of this research work focused on investigating the impact that the two *random* and *smart explosion* approaches have on the basic version (Random-IA), and how much they positively affect its overall performances. Figure 7.5 therefore shows the convergence behaviour of the proposed multi-level approaches compared with Random-IA. In particular, the three convergence curves of (1) Random-IA, (2) Random-IA with *random explosion*, and (3) Random-IA with *smart explosion* are displayed, from which it is possible to analyze how much improvement the two proposed multi-level approaches produce compared to the basic version. With regard to the larger benchmark networks, it can be clearly seen how the improvements produced by the two multi-level approaches are remarkably reaching significantly higher modularity values. Inspecting only the comparison between the two multi-level approaches it is possible to assert: (a) on the *Email* network the *random explosion* shows an initially slower convergence than *smart explosion*, whilst, afterwards, the two curves join showing the same convergence behaviour. However, towards the end of the run, *random explosion* is able to improve and reach a slightly higher modularity value than *smart explosion*; (b) on the *Yeast* and *Power* networks, instead, *smart explosion* clearly outperforms *random explosion*, especially on the larger network (*Power*), where the distance between the curves is quite significant and clear in favour of *smart explosion*.

In Table 7.2 we can see, respectively, the best modularity found, the mean of the best, and the standard deviation ($mean \pm \sigma$), for all three benchmark networks considered. The outcomes shown in the table confirm what was asserted from the



(a) **(b)**

(c)

Figure 7.5: Convergence analysis over time of Random-IA, Random-IA with *random explosion*, and Random-IA with *smart explosion* on the benchmark networks (a) *Email*, (b) *Yeast* and (c) *Power*.

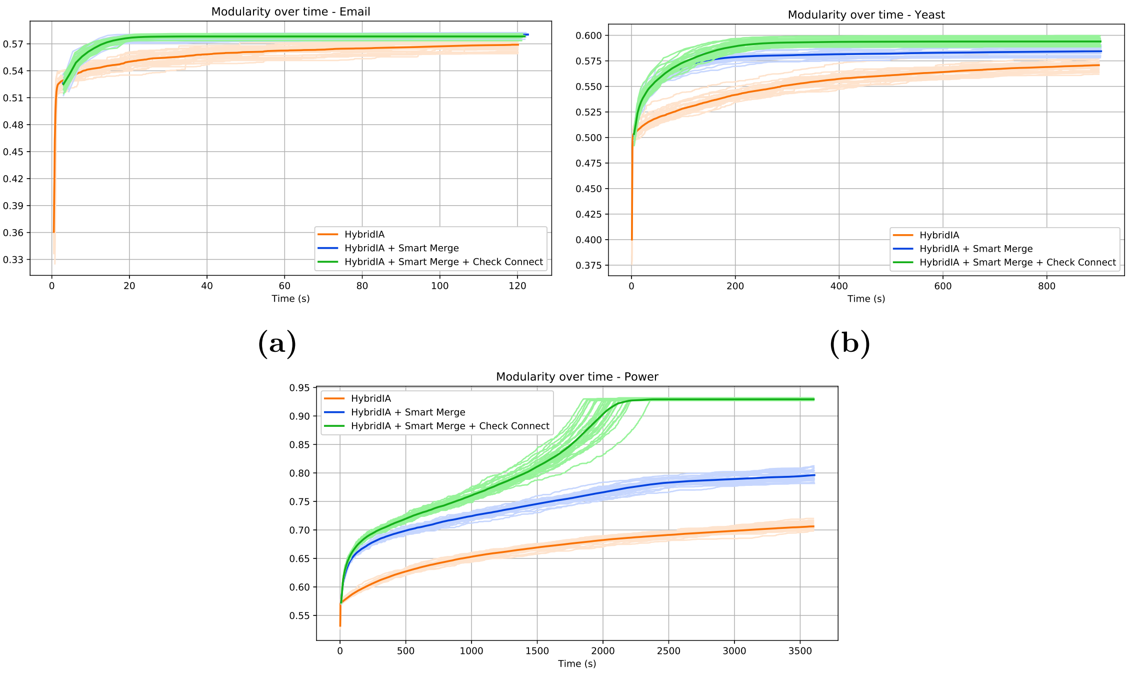
convergence plots, that the *random explosion* works better on the smaller networks (i.e. *Email*), whilst *smart explosion* on the other two. With regard to the *Power* network, which is the larger and then the most significant from the multi-level approach perspective, the modularity value found by *smart explosion* is way better than the others, especially with respect to the basic version that instead finds low values of modularity (0.1260). This points out, then, how the multi-level approach designed in smart explosion helps the random-search algorithm (Random-IA) in revealing good community structures.

Table 7.2: Comparative results of *random* and *smart explosion* and Random-IA. Best modularity found, mean and standard deviation (σ) as comparison measures.

	Email		Yeast		Power	
Algorithm	*best*	*mean*$\pm\sigma$	*best*	*mean*$\pm\sigma$	*best*	*mean*$\pm\sigma$
Random-IA	0.3841	0.3465 ± 0.0186	0.4411	0.4089 ± 0.0171	0.1260	0.1200 ± 0.0026
Random-IA+RE	0.5627	0.5416 ± 0.0142	0.5388	0.5210 ± 0.0091	0.5791	0.5568 ± 0.0124
Random-IA+SE	0.5539	0.5282 ± 0.0160	0.5538	0.5404 ± 0.0092	0.7532	0.7364 ± 0.0093

The same analysis was conducted to understand how the *smart merge* approach affects the performance of Hybrid-IA, which is the basic version on which it is applied.

In Figure 7.6 is therefore shown the convergence behaviour of the multi-level approach compared to the basic one. By inspecting the three plots, it can be seen how *smart merge* and the *smart merge + check connect* variant are similar on the *Email* network, whilst in the *Yeast* one the connected-components version is shown to be slightly better than the *smart merge* version alone. It is important to point out that both multi-level versions improve in any case the performance of the basic algorithm, although such improvements are moderate.



(a) **(b)**

(c)

Figure 7.6: Convergence analysis over time of HYBRID-IA, HYBRID-IA with *smart merge*, and HYBRID-IA with *smart merge + check connect* on the benchmark networks (a) *Email*, (b) *Yeast* and (c) *Power*.

The improvements produced by the *smart merge* and *smart merge + check connect* approaches are best seen on the larger network *Power*, where the gap between the three curves is clear and marked. In particular, the variant *smart merge + check connect* produces sharply best performance, reaching considerably higher modularity values than the basic version, and the *smart merge* one.

These improvements are also confirmed by the results reported in Table 7.3, both in terms of best and average modularity value found. Indeed, by inspecting the table,

Table 7.3: Comparative results of *smart merge* and Hybrid-IA. Best modularity found, mean and standard deviation (σ) as comparison measures.

	Email		Yeast		Power	
Algorithm	*best*	*mean*$\pm\sigma$	*best*	*mean*$\pm\sigma$	*best*	*mean*$\pm\sigma$
Hybrid-IA	0.5782	0.5690 ± 0.0049	0.5858	0.5710 ± 0.0057	0.7202	0.7065 ± 0.0063
Hybrid-IA+SM	0.5824	0.5801 ± 0.0018	0.5929	0.5845 ± 0.0045	0.8125	0.7964 ± 0.0099
Hybrid-IA+SM+C	0.5813	0.5782 ± 0.0019	0.5998	0.5940 ± 0.0033	0.9321	0.9294 ± 0.0015

it clearly appears that, due to the high-quality solutions produced by Hybrid-IA on networks not excessively large, the effects and improvements produced by the multi-level approach are limited, while instead on the large one, where the basic algorithm struggles to reach high modularity values, the improvement contribution given by the multi-level approach is notable and mainly relevant (0.7202 vs 0.9321).

Finally, Table 7.4 reports the comparisons between the *smart merge + check connect* variant (being the best approach) and the state-of-the-art: SS+ML, a multi-level algorithm based on a single-step greedy coarsening and fast greedy refinement [110]; MSG-VM, a multistep greedy algorithm with vertex mover [124]; Louvain, a fast multi-level greedy algorithm [12]; CNTS, a combined neighbourhood tabu search [65]; and CNTS-ML, the multi-level version of the CNTS algorithm [65]. It is possible to see how the proposed multilevel approach is competitive with the community detection state-of-the-art on the first two benchmark networks, a little less on the *Power* one. However, on this last network, the results obtained by Hybrid-IA with *smart merge + check connect* are not so far from the compared ones.

Table 7.4: Comparative results of Hybrid-IA with *smart merge* and state-of-the-art algorithms.

Network	MSG-VM	SS+ML	Louvain	CNTS	CNTS-ML	Hybrid-IA+SM+C
Email	0.5746	0.5813	0.5758	0.5820	0.5815	0.5813
Yeast	0.5948	0.6068	0.5962	0.6053	0.6055	0.5998
Power	0.9381	0.9392	0.9371	0.9380	0.9392	0.9321

7.4 Conclusions

The multi-level models we propose for community detection on quite large networks and which are based on two variants of an immune-inspired algorithm were experimentally shown to be very competitive and efficient. Yet, still trailing some state-of-the-art methodologies, especially on extremely large networks. Given such promising initial results, as future work, we plan to tackle even larger networks, in particular biological and online social networks. We will focus our research direction on implementing mechanisms, such as reinforcement and probabilistic learning, to better guide the level construction phase of the multi-level approaches to further improve both the objective function and convergence.

8

Conclusions

In this thesis work, some research contributions in the field of hybrid metaheuristics have been presented. In particular, combinatorial optimization problems on graphs have been analyzed, such as vertex/arc removal or grouping problems, using an immunological algorithm combined with local search techniques in the first phase, and reinforcement learning components in the second step.

Initially, I focused my attention on designing a hybrid immunological algorithm, called Hybrid-IA, to address the problem of Feedback Vertex Set, a well-known combinatorial optimization problem that finds application in many real problems. The proposed algorithm has been tested on a dataset of over 800 instances with different characteristics. The results of an initial preliminary experimental phase have shown that the proposed algorithm has performance comparable to results obtained by other metaheuristic algorithms. Moreover, tuning the parameters has allowed adapting the search process of the algorithm to the features of the input instance, further improving the results, especially on large instances of the benchmark dataset.

In parallel, a completely random immunological algorithm guided by stochastic operators was developed to solve the problem of Community Detection on social, biological and synthetic networks. The results of the experimental phase and the analysis conducted show the reliability of the proposed algorithm. But the limits of this type of search process are the large number of generations needed to converge towards acceptable solutions for large instances of the problem. In light of this, this algorithm has been extended by introducing a local search procedure, which aim is to locally maximizes the modularity function in a greedy way. The results obtained by the proposed algorithm have been analyzed with respect to three similarity measures, showing the ability of Hybrid-IA to detect a community structure more similar to the real one on synthetic networks. Also, for this problem, it was investigated whether the location of the local search procedure within the immunological algorithm affects its performance, both in terms of solution quality and learning process. Analysis of the results has shown that a high diversity of the population allows the algorithm to discover better solutions.

Furthermore, two multi-level approaches have been proposed to tackle large instances of the community detection problem. The first approach is based on a backtracking mechanism, while the second one is on a quality-based aggregation method. Both multi-level optimization approaches have been applied using two different immunological algorithms as an underlying heuristic. From the experimental analysis emerges that the two proposed models help the underlying algorithms to significantly improve their performances from both quality of solutions found and the computational point of view.

Finally, I focused my research on designing a general-purpose framework for combinatorial optimization problems. The proposed algorithm combines a population-based greedy metaheuristic with reinforcement learning techniques to extract useful information from the local optima discovered during the search. The framework employs a randomized greedy algorithm to construct new solutions and a probability learning component to learn which solution components are more promising. The Feedback Vertex Set problem has been considered as a case study. The results on the benchmark dataset have shown that the proposed framework reaches better solutions than the compared algorithms on squared and not squared grid graphs while obtaining comparable results on random, toroidal and hypercube graphs.

Other contributions, not included in this document, concern experimental analysis of how different strategies on an Ant Colony Optimization (ACO) algorithm affect the optimization efficiency of the entire colony, in an unknown and dynamic environment [39, 38]. Furthermore, the ant colony optimization has been used to hybridize an agent-based model to evaluate the effects of different behaviours in crowd simulations [37, 36, 35], with respect to three evaluation metrics: the number of agents that reach the goal and the time and cost required to reach it.

Currently, my research work is focused on an extension of the *Construct, Merge, Solve & Adapt* (CMSA) framework, initially proposed in [15, 14]. The main idea is to introduce a learning component to guide the construction of new solutions, similarly as shown in Chapter 3. Some initial experiments have shown promising results on the Weighed Vertex Cover.

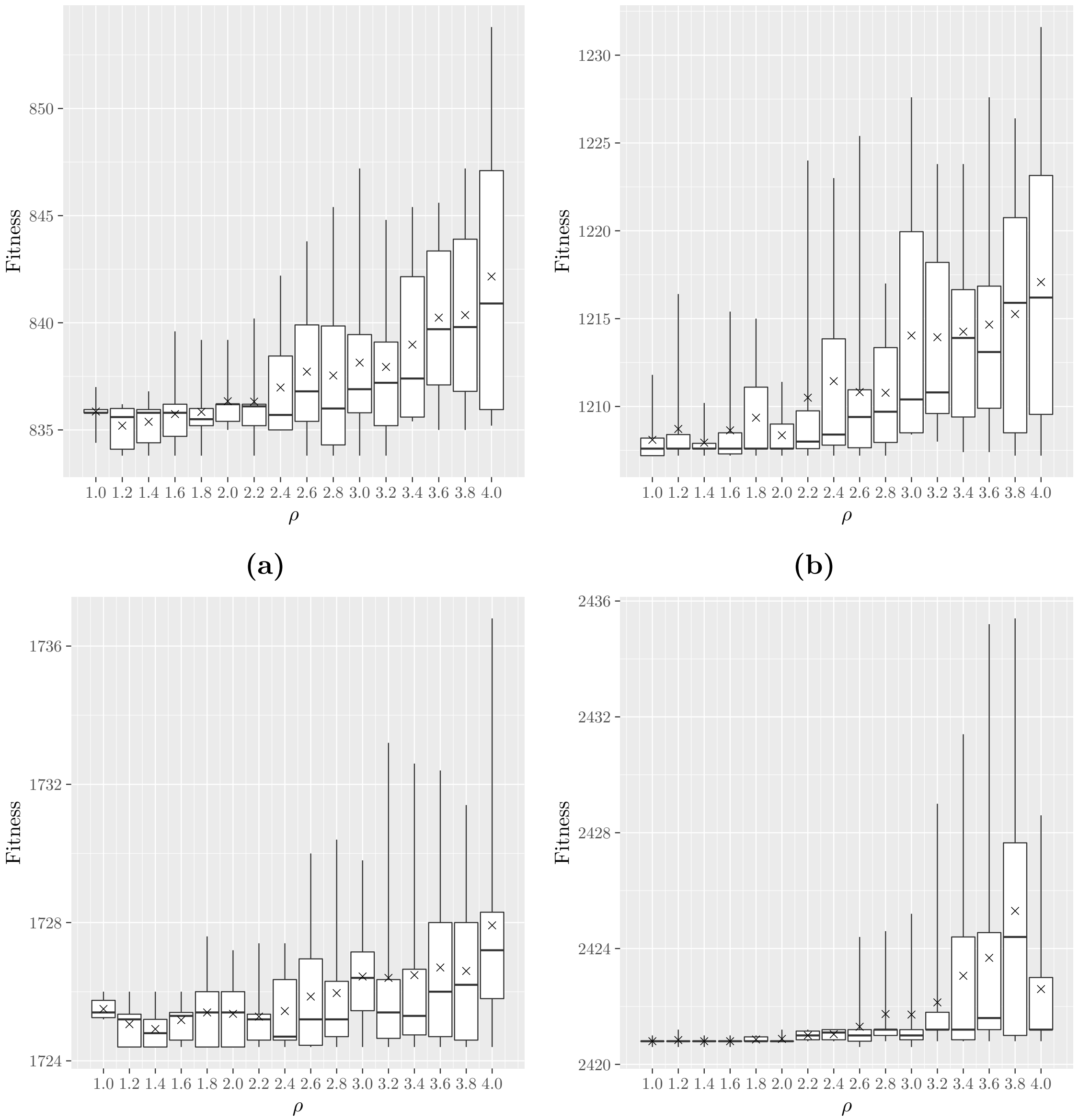
In future work, I plan to investigate more in detail the probability updating procedure of the framework proposed in Chapter 3 and to study some mechanisms to reduce solution components probabilities periodically in order to forget some earlier decisions. Additionally, I plan to apply the proposed algorithm to other types of combinatorial optimization problems. Moreover, starting from the idea proposed in Section 7.2, I would like to develop a smart merge approach with probability learning within an iterated multi-level optimization algorithm.

A

Appendix

A.1 Parameter Tuning Results

In this section are reported the outcomes of the parameter tuning of Hybrid-IA for the Weighted Feedback Vertex Set described in Section 2.2.3. For each value of ρ the box-plots in Figures A.1–A.10 represents the distribution of the results over 10 runs on instances of the training set, grouped by number of vertices, density (number of edges) and weight range. The box represents the distribution between the first and the third quartile, with a horizontal line to denote the second quartile (the median), while the two vertical lines reach the minimum and the maximum values respectively. The cross-shaped point represents the average value.



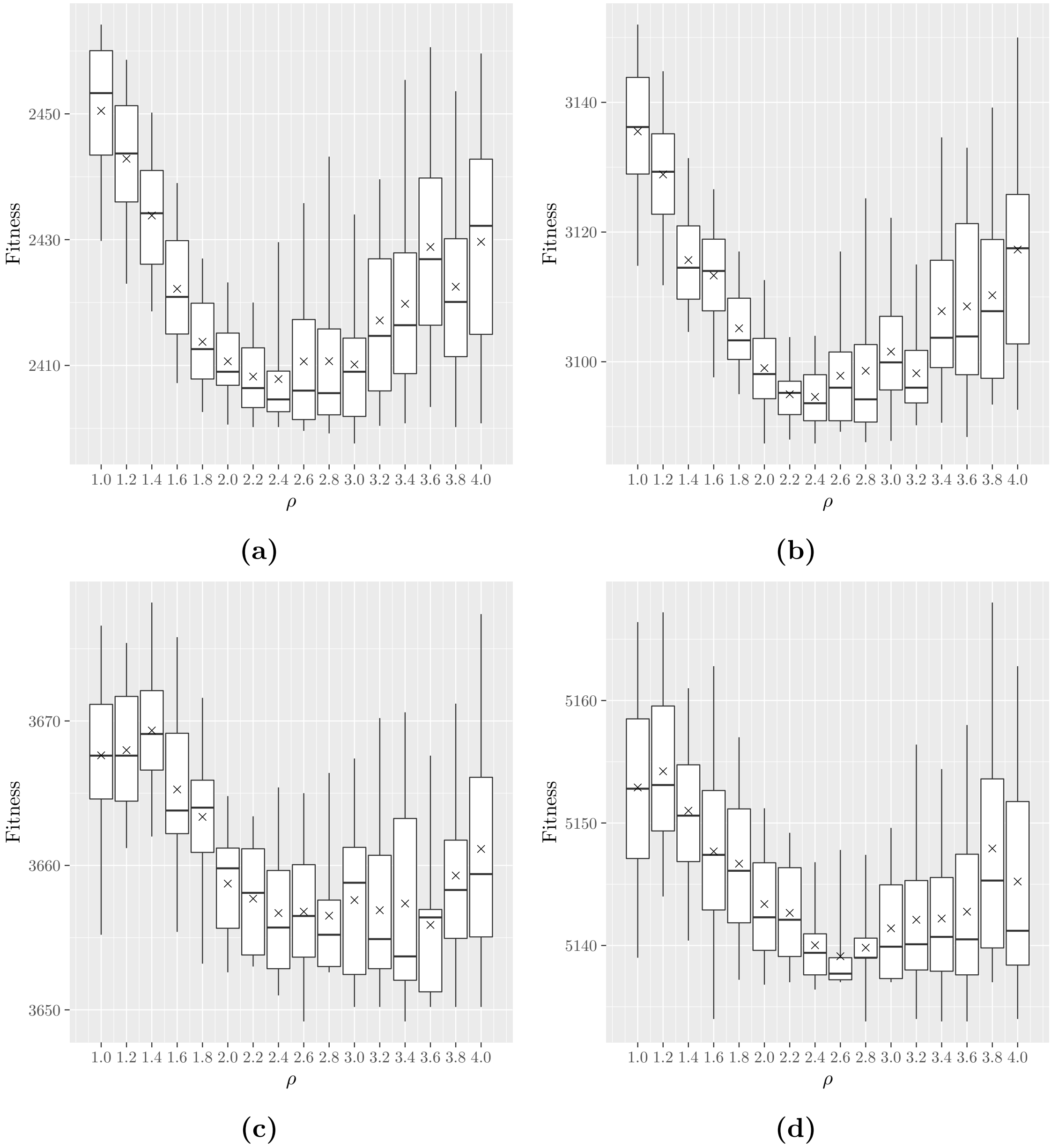
(a)

(b)

(c)

(d)

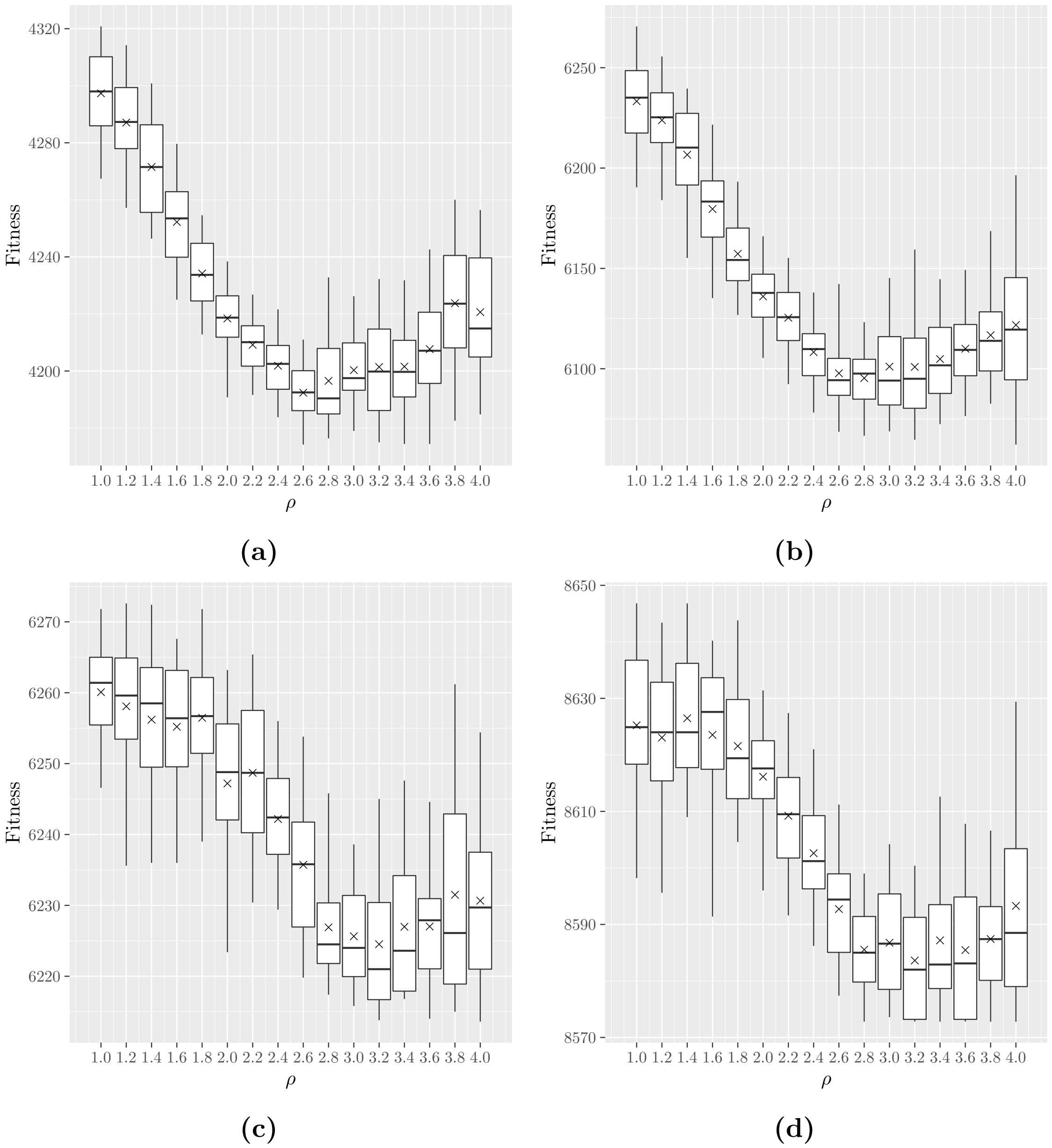
Figure A.1: Tuning results for different values of the mutation shape ρ on random instances with $n = 100$, $m = 247$, weight range $[10, 50]$ (a) and $[10, 75]$ (b), and on random instances with $n = 100$, $m = 841$, weight range $[10, 50]$ (c) and $[10, 75]$ (d).



(a) **(b)**

(c) **(d)**

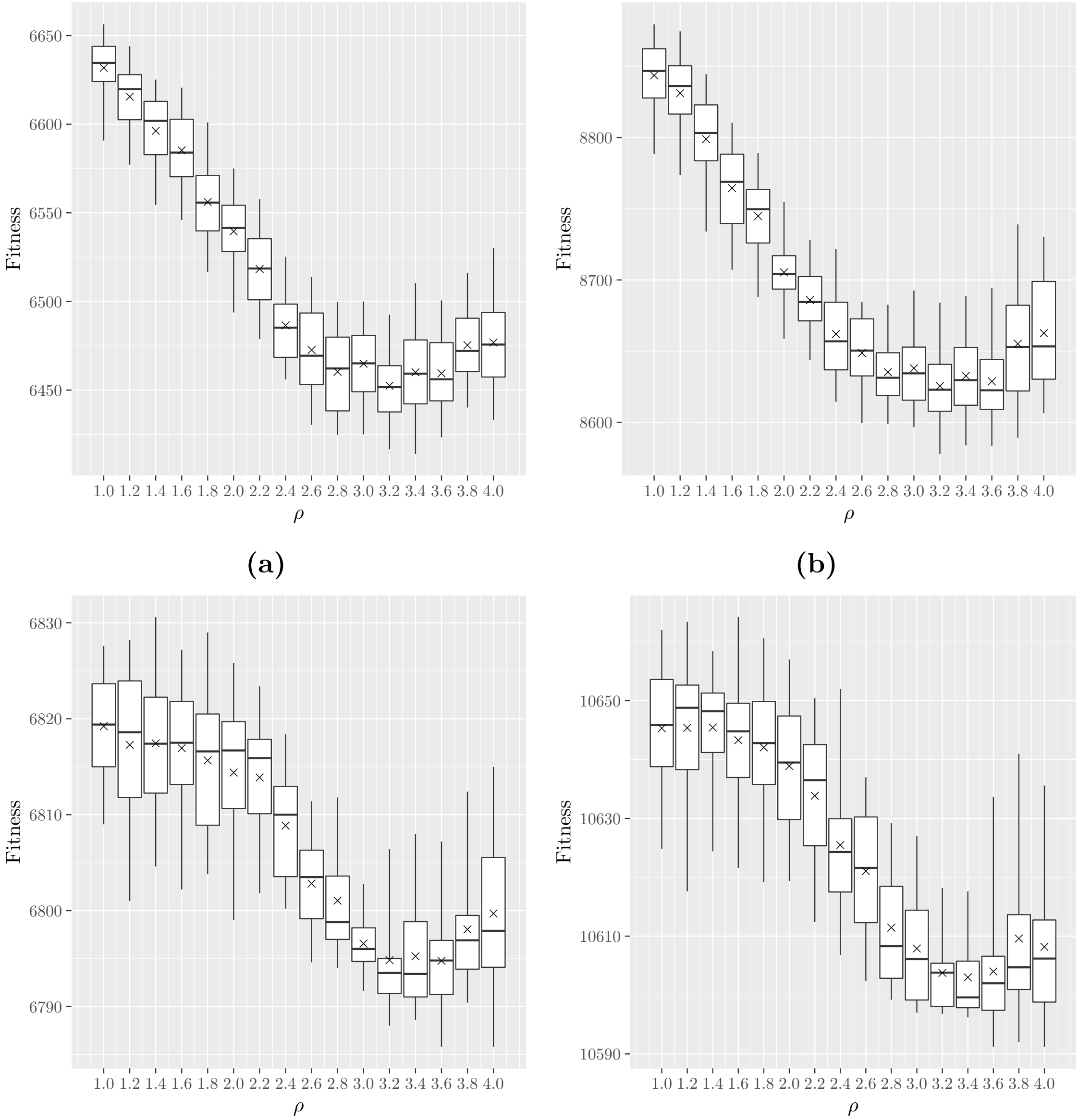
Figure A.2: Tuning results for different values of the mutation shape ρ on random instances with $n = 200$, $m = 796$, weight range $[10, 50]$ (a) and $[10, 75]$ (b), and on random instances with $n = 200$, $m = 3184$, weight range $[10, 50]$ (c) and $[10, 75]$ (d).



(a) **(b)**

(c) **(d)**

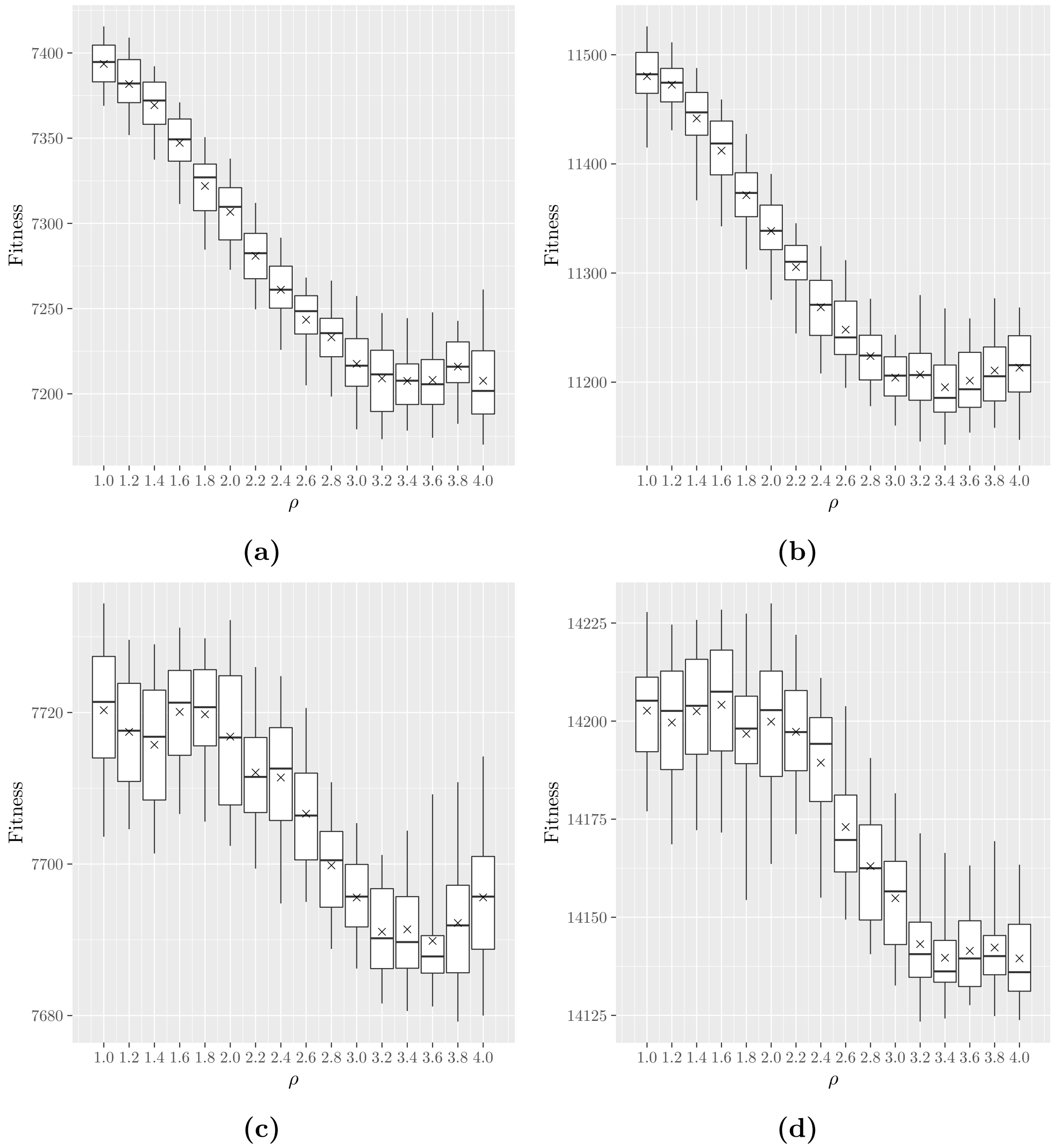
Figure A.3: Tuning results for different values of the mutation shape ρ on random instances with $n = 300$, $m = 1644$, weight range $[10, 50]$ (a) and $[10, 75]$ (b), and on random instances with $n = 300$, $m = 7026$, weight range $[10, 50]$ (c) and $[10, 75]$ (d).



(a) **(b)**

(c) **(d)**

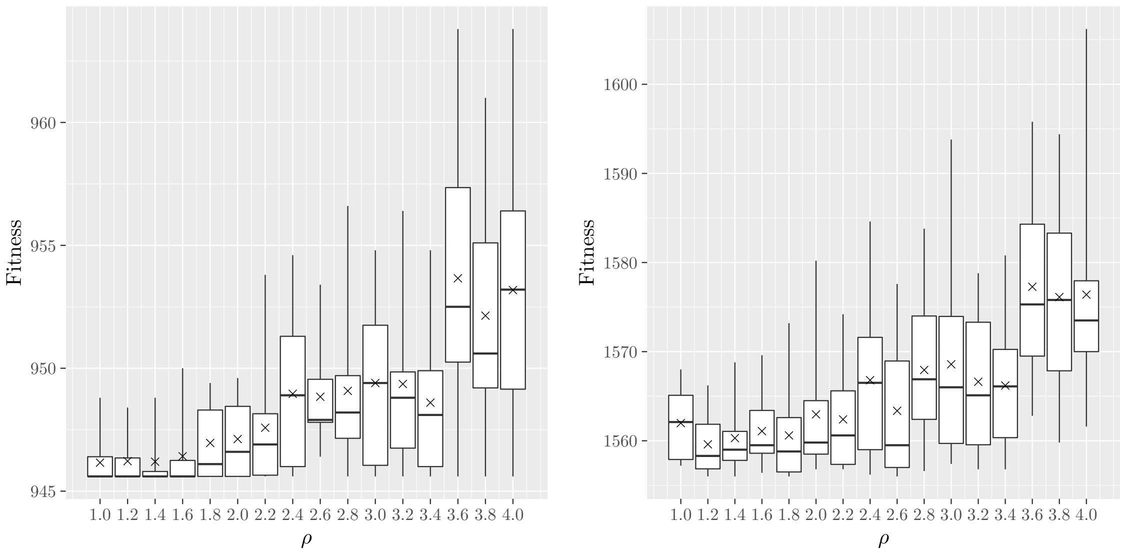
Figure A.4: Tuning results for different values of the mutation shape ρ on random instances with $n = 400$, $m = 2793$, weight range $[10, 50]$ (a) and $[10, 75]$ (b), and on random instances with $n = 400$, $m = 12369$, weight range $[10, 50]$ (c) and $[10, 75]$ (d).



(a) **(b)**

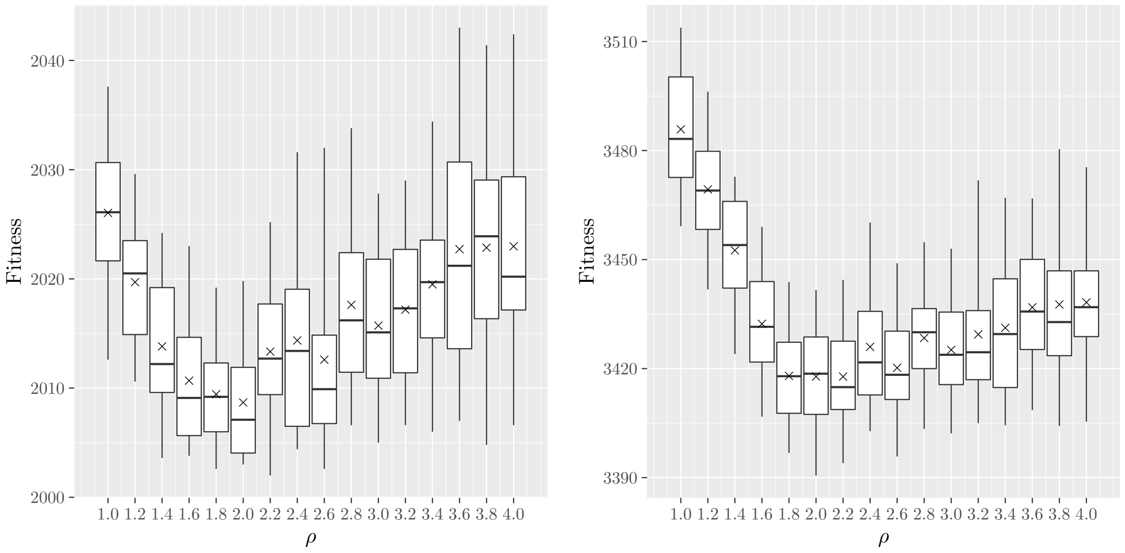
(c) **(d)**

Figure A.5: Tuning results for different values of the mutation shape ρ on random instances with $n = 500$, $m = 4241$, weight range $[10, 50]$ (a) and $[10, 75]$ (b), and on random instances with $n = 500$, $m = 19211$, weight range $[10, 50]$ (c) and $[10, 75]$ (d).



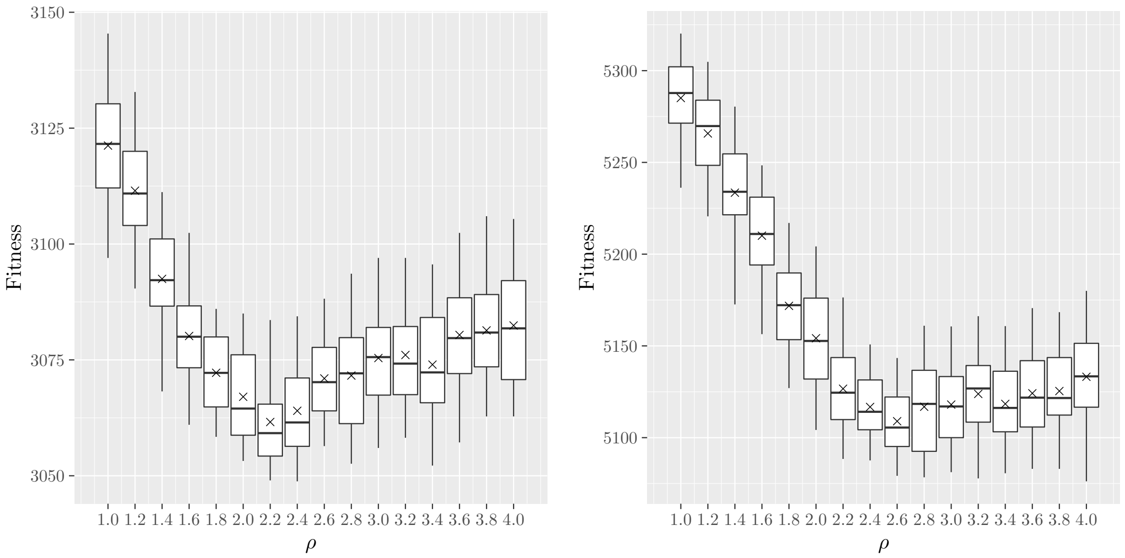
(a) **(b)**

Figure A.6: Tuning results for different values of the mutation shape ρ on squared grid instances with $x = 10$, $y = 10$, weight range $[10, 50]$ (a) and $[10, 75]$ (b).



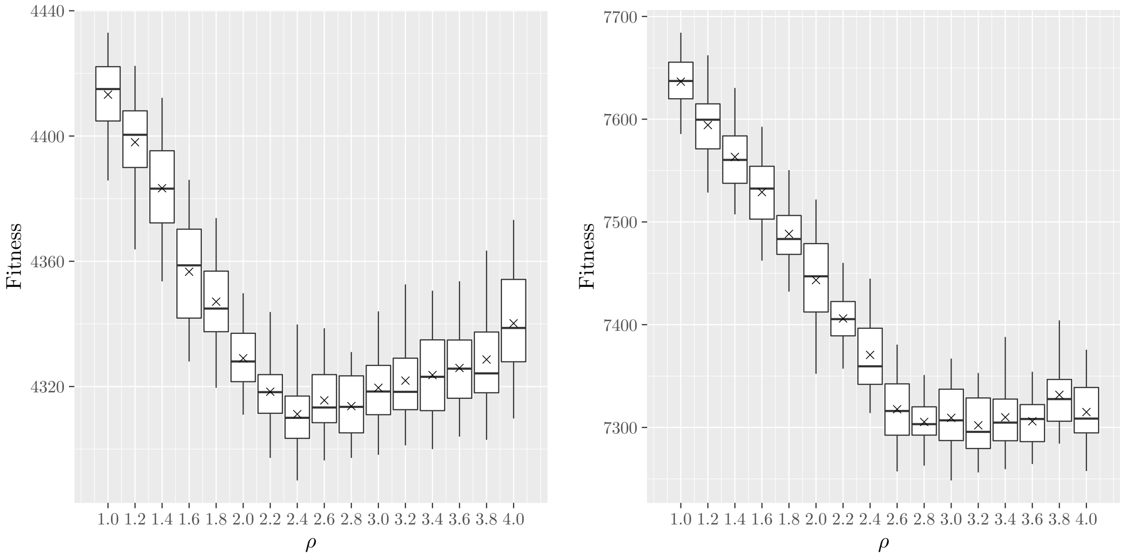
(a) **(b)**

Figure A.7: Tuning results for different values of the mutation shape ρ on squared grid instances with $x = 14$, $y = 14$, weight range $[10, 50]$ (a) and $[10, 75]$ (b).



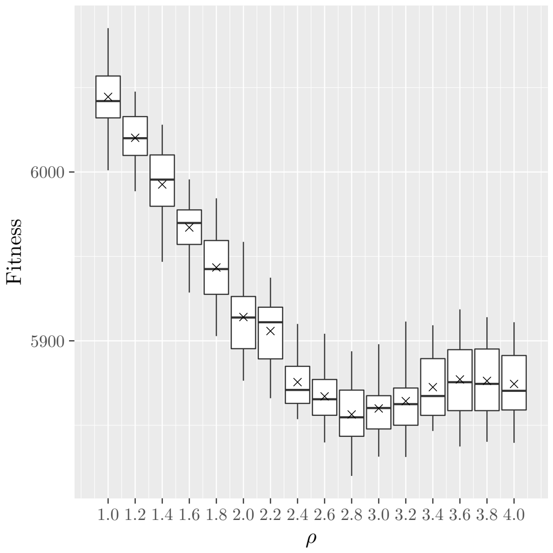
(a) **(b)**

Figure A.8: Tuning results for different values of the mutation shape ρ on squared grid instances with $x = 17$, $y = 17$, weight range $[10, 50]$ (a) and $[10, 75]$ (b).

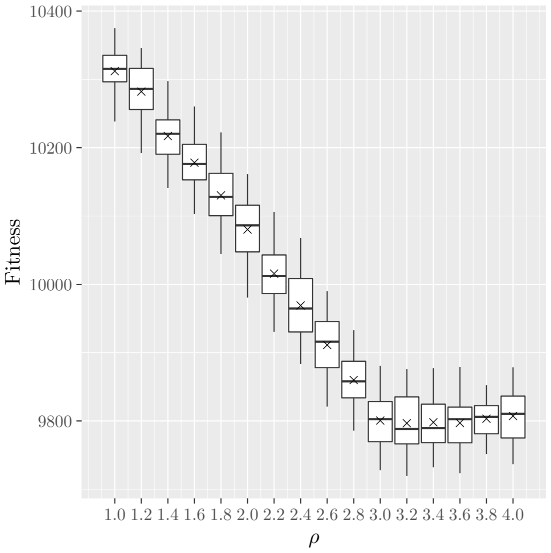


(a) **(b)**

Figure A.9: Tuning results for different values of the mutation shape ρ on squared grid instances with $x = 20$, $y = 20$, weight range $[10, 50]$ (a) and $[10, 75]$ (b).



(a)



(b)

Figure A.10: Tuning results for different values of the mutation shape ρ on squared grid instances with $x = 23$, $y = 23$, weight range $[10, 50]$ (a) and $[10, 75]$ (b).

Bibliography

[1] Aiex, R. M., Resende, M. G. C. and Ribeiro, C. C. 'Probability Distribution of Solution Time in GRASP: An Experimental Investigation'. In: *Journal of Heuristics* 8 (2002), pp. 343–373. DOI: 10.1023/A:1015061802659.

[2] Aiex, R. M., Resende, M. G. C. and Ribeiro, C. C. 'TTT plots: a perl program to create time-to-target plots'. In: *Optimization Letters* 1 (2007), pp. 355–366. DOI: 10.1007/s11590-006-0031-4.

[3] Arenas, A., Duch, J., Fernández, A. and Gómez, S. 'Size reduction of complex networks preserving modularity'. In: *New Journal of Physics* 9.6 (June 2007), pp. 176–176. DOI: 10.1088/1367-2630/9/6/176.

[4] Atay, Y., Koc, I., Babaoglu, I. and Kodaz, H. 'Community detection from biological and social networks: A comparative analysis of metaheuristic algorithms'. In: *Applied Soft Computing* 50 (2017), pp. 194–211. DOI: 10.1016/j.asoc.2016.11.025.

[5] Aureli, M., Masilamani, A. P., Illuzzi, G., Loberto, N., Scandroglio, F., Prinetti, A., Chigorno, V. and Sonnino, S. 'Activity of plasma membrane β-galactosidase and β-glucosidase'. In: *FEBS Letters* 583.15 (2009), pp. 2469–2473. DOI: 10.1016/j.febslet.2009.06.048.

[6] Bafna, V., Berman, P. and Fujito, T. 'A 2-Approximation Algorithm for the Undirected Feedback Vertex Set Problem'. In: *SIAM Journal on Discrete Mathematics* 12.3 (1999), pp. 289–297. DOI: 10.1137/S0895480196305124.

[7] Barabási, A.-L., Albert, R. and Jeong, H. 'Scale-free characteristics of random networks: the topology of the world-wide web'. In: *Physica A: Statistical Mechanics and its Applications* 281.1 (2000), pp. 69–77. DOI: 10.1016/S0378-4371(00)00018-2.

[8] Barabási, A.-L., Gulbahce, N. and Loscalzo, J. 'Network medicine: a network-based approach to human disease'. In: *Nature Reviews Genetics* 12.1 (2011), pp. 56–68. DOI: 10.1038/nrg2918.

[9] Barabási, A.-L. and Oltvai, Z. N. 'Network biology: understanding the cell's functional organization'. In: *Nature Reviews Genetics* 5.2 (2004), pp. 101–113. DOI: 10.1038/nrg1272.

[10] Benedettini, S., Blum, C. and Roli, A. 'A Randomized Iterated Greedy Algorithm for the Founder Sequence Reconstruction Problem'. In: *Learning and Intelligent Optimization (LION 2010)*. Ed. by Blum, C. and Battiti, R. Berlin, Heidelberg: Springer, 2010, pp. 37–51. DOI: 10.1007/978-3-642-13800-3_4.

[11] Benlic, U., Epitropakis, M. G. and Burke, E. K. 'A hybrid breakout local search and reinforcement learning approach to the vertex separator problem'. In: *European Journal of Operational Research* 261.3 (2017), pp. 803–818. DOI: 10.1016/j.ejor.2017.01.023.

[12] Blondel, V. D., Guillaume, J.-L., Lambiotte, R. and Lefebvre, E. 'Fast unfolding of communities in large networks'. In: *Journal of Statistical Mechanics: Theory and Experiment* 2008.10 (Oct. 2008), P10008. DOI: 10.1088/1742-5468/2008/10/P10008.

[13] Blum, C. and Raidl, G. R. *Hybrid Metaheuristics: Powerful Tools for Optimization*. Springer, 2016. DOI: 10.1007/978-3-319-30883-8.

[14] Blum, C. and Ochoa, G. 'A comparative analysis of two matheuristics by means of merged local optima networks'. In: *European Journal of Operational Research* 290.1 (2021), pp. 36–56. DOI: 10.1016/j.ejor.2020.08.008.

[15] Blum, C., Pinacho, P., López-Ibáñez, M. and Lozano, J. A. 'Construct, Merge, Solve & Adapt A new general algorithm for combinatorial optimization'. In: *Computers & Operations Research* 68 (2016), pp. 75–88. DOI: 10.1016/j.cor.2015.10.014.

[16] Blum, C., Puchinger, J., Raidl, G. R. and Roli, A. 'Hybrid metaheuristics in combinatorial optimization: A survey'. In: *Applied Soft Computing* 11.6 (2011), pp. 4135–4151. DOI: 10.1016/j.asoc.2011.02.032.

[17] Blum, C. and Roli, A. 'Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison'. In: *ACM Computing Surveys* 35.3 (Sept. 2003), pp. 268–308. DOI: 10.1145/937503.937505.

[18] Boccaletti, S., Latora, V., Moreno, Y., Chavez, M. and Hwang, D.-U. 'Complex networks: Structure and dynamics'. In: *Physics Reports* 424.4 (2006), pp. 175–308. DOI: 10.1016/j.physrep.2005.10.009.

[19] Bordonaro, G., Scollo, R. A., Cutello, V. and Pavone, M. 'A Comparative Analysis of Different Multilevel Approaches for Community Detection'. In: *Metaheuristics (MIC 2022)*. Ed. by Di Gaspero, L., Festa, P., Nakib, A. and Pavone, M. Vol. 13838. Lecture Notes in Computer Science. Cham: Springer, 2023, pp. 230–245. DOI: 10.1007/978-3-031-26504-4_17.

[20] Bouamama, S. and Blum, C. 'On solving large-scale instances of the knapsack problem with setup by means of an iterated greedy algorithm'. In: *2017 6th International Conference on Systems and Control (ICSC 2017)*. IEEE, 2017, pp. 342–347. DOI: 10.1109/ICoSC.2017.7958697.

[21] Bouamama, S., Blum, C. and Boukerram, A. 'A population-based iterated greedy algorithm for the minimum weight vertex cover problem'. In: *Applied Soft Computing* 12.6 (2012), pp. 1632–1639. DOI: 10.1016/j.asoc.2012.02.013.

[22] Bouamama, S., Blum, C. and Pinacho-Davidson, P. 'A Population-Based Iterated Greedy Algorithm for Maximizing Sensor Network Lifetime'. In: *Sensors* 22.5 (2022). 1804. DOI: 10.3390/s22051804.

[23] Brandes, U., Delling, D., Gaertler, M., Görke, R., Hoefer, M., Nikoloski, Z. and Wagner, D. 'On Modularity Clustering'. In: *IEEE Transactions on Knowledge and Data Engineering* 20.2 (Feb. 2008), pp. 172–188. DOI: 10.1109/TKDE.2007.190689.

[24] Brunetta, L., Maffioli, F. and Trubian, M. 'Solving the feedback vertex set problem on undirected graphs'. In: *Discrete Applied Mathematics* 101.1 (2000), pp. 37–51. DOI: 10.1016/S0166-218X(99)00180-8.

[25] Bu, D., Zhao, Y., Cai, L., Xue, H., Zhu, X., Lu, H., Zhang, J., Sun, S., Ling, L., Zhang, N., Li, G. and Chen, R. 'Topological structure analysis of the protein-protein interaction network in budding yeast'. In: *Nucleic Acids Research* 31.9 (May 2003), pp. 2443–2450. DOI: 10.1093/nar/gkg340.

[26] Cantini, L., Medico, E., Fortunato, S. and Caselle, M. 'Detection of gene communities in multi-networks reveals cancer drivers'. In: *Scientific Reports* 5 (2015), p. 17386. DOI: 10.1038/srep17386.

[27] Cao, C., Ni, Q. and Zhai, Y. 'A novel community detection method based on discrete particle swarm optimization algorithms in complex networks'. In: *2015 IEEE Congress on Evolutionary Computation (CEC 2015)*. 2015, pp. 171–178. DOI: 10.1109/CEC.2015.7256889.

[28] Carrabs, F., Cerrone, C. and Cerulli, R. 'A memetic algorithm for the weighted feedback vertex set problem'. In: *Networks* 64.4 (2014), pp. 339–356. DOI: 10.1002/net.21577.

[29] Carrabs, F., Cerulli, R., Gentili, M. and Parlato, G. 'A linear time algorithm for the minimum Weighted Feedback Vertex Set on diamonds'. In: *Information Processing Letters* 94.1 (2005), pp. 29–35. DOI: 10.1016/j.ipl.2004.12.008.

[30] Carrabs, F., Cerulli, R., Gentili, M. and Parlato, G. 'A Tabu Search Heuristic Based on k-Diamonds for the Weighted Feedback Vertex Set Problem'. In: *Network Optimization*. Ed. by Pahl, J., Reiners, T. and Voß, S. Berlin, Heidelberg: Springer, 2011, pp. 589–602. DOI: 10.1007/978-3-642-21527-8_66.

[31] *Cattle Protein-Protein Interactions.* 2009. URL: https://biit.cs.ut.ee/graphweb/welcome.cgi?t=examples.

[32] Chen, J. and Yuan, B. ‘Detecting functional modules in the yeast protein–protein interaction network’. In: *Bioinformatics* 22.18 (July 2006), pp. 2283–2290. DOI: 10.1093/bioinformatics/btl370.

[33] Civicioglu, P. ‘Transforming geocentric cartesian coordinates to geodetic coordinates by using differential search algorithm’. In: *Computers & Geosciences* 46 (2012), pp. 229–247. DOI: 10.1016/j.cageo.2011.12.011.

[34] Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C. *Introduction to algorithms.* MIT press, 2009.

[35] Crespi, C., Fargetta, G., Pavone, M. and Scollo, R. A. ‘An agent-based model for crowd simulation’. In: *Artificial Life and Evolutionary Computation (WIVACE 2022).* Communications in Computer and Information Science. (in press). Cham: Springer, 2023.

[36] Crespi, C., Fargetta, G., Pavone, M. and Scollo, R. A. ‘An Agent-Based Model to Investigate Different Behaviours in a Crowd Simulation’. In: *Bioinspired Optimization Methods and Their Applications (BIOMA 2022).* Ed. by Mernik, M., Črepinšek, M. and Eftimov, T. Vol. 13627. Lecture Notes in Computer Science. Cham: Springer, 2022, pp. 1–14. DOI: 10.1007/978-3-031-21094-5_1.

[37] Crespi, C., Fargetta, G., Pavone, M., Scollo, R. A. and Scrimali, L. ‘A Game Theory Approach for Crowd Evacuation Modelling’. In: *Bioinspired Optimization Methods and Their Applications (BIOMA 2020).* Ed. by Filipič, B., Minisci, E. and Vasile, M. Vol. 12438. Lecture Notes in Computer Science. Cham: Springer, 2020, pp. 228–239. DOI: 10.1007/978-3-030-63710-1_18.

[38] Crespi, C., Scollo, R. A., Fargetta, G. and Pavone, M. ‘How a Different Ant Behavior Affects on the Performances of the Whole Colony’. In: *Metaheuristics (MIC 2022).* Ed. by Di Gaspero, L., Festa, P., Nakib, A. and Pavone, M.

Vol. 13838. Lecture Notes in Computer Science. Cham: Springer, 2023, pp. 187–199. DOI: 10.1007/978-3-031-26504-4_14.

[39] Crespi, C., Scollo, R. A. and Pavone, M. 'Effects of Different Dynamics in an Ant Colony Optimization Algorithm'. In: *2020 7th International Conference on Soft Computing Machine Intelligence (ISCMI 2020)*. IEEE, Nov. 2020, pp. 8–11. DOI: 10.1109/ISCMI51676.2020.9311553.

[40] Csardi, G. and Nepusz, T. 'The igraph software package for complex network research'. In: *InterJournal* Complex Systems (2006), p. 1695. URL: https://igraph.org.

[41] Cutello, V., Fargetta, G., Pavone, M. and Scollo, R. A. 'Optimization Algorithms for Detection of Social Interactions'. In: *Algorithms* 13.6 (2020). DOI: 10.3390/a13060139.

[42] Cutello, V., Lee, D., Nicosia, G., Pavone, M. and Prizzi, I. 'Aligning Multiple Protein Sequences by Hybrid Clonal Selection Algorithm with Insert-Remove-Gaps and BlockShuffling Operators'. In: *International Conference on Artificial Immune Systems (ICARIS 2006)*. Ed. by Bersini, H. and Carneiro, J. Berlin, Heidelberg: Springer, 2006, pp. 321–334. DOI: 10.1007/11823940_25.

[43] Cutello, V., Nicosia, G. and Pavone, M. 'An immune algorithm with stochastic aging and Kullback entropy for the chromatic number problem'. In: *Journal of Combinatorial Optimization* 14.1 (2007), pp. 9–33. DOI: 10.1007/s10878-006-9036-2.

[44] Cutello, V., Nicosia, G., Pavone, M. and Prizzi, I. 'Protein multiple sequence alignment by hybrid bio-inspired algorithms'. In: *Nucleic Acids Research* 39.6 (Mar. 2011), pp. 1980–1992. DOI: 10.1093/nar/gkq1052.

[45] Cutello, V., Nicosia, G., Pavone, M. and Timmis, J. 'An Immune Algorithm for Protein Structure Prediction on Lattice Models'. In: *IEEE Transactions on Evolutionary Computation* 11.1 (Feb. 2007), pp. 101–117. DOI: 10.1109/TEVC.2006.880328.

[46] Cutello, V., Oliva, M., Pavone, M. and Scollo, R. A. 'A Hybrid Immunological Search for the Weighted Feedback Vertex Set Problem'. In: *Learning and Intelligent Optimization (LION 2019).* Ed. by Matsatsinis, N. F., Marinakis, Y. and Pardalos, P. Vol. 11968. Lecture Notes in Computer Science. Cham: Springer, 2020, pp. 1–16. DOI: 10.1007/978-3-030-38629-0_1.

[47] Cutello, V., Oliva, M., Pavone, M. and 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 2019).* IEEE, Dec. 2019, pp. 1928–1936. DOI: 10.1109/SSCI44817.2019.9002988.

[48] Danon, L., Díaz-Guilera, A., Duch, J. and Arenas, A. 'Comparing community structure identification'. In: *Journal of Statistical Mechanics: Theory and Experiment* 2005.09 (Sept. 2005), P09008–P09008. DOI: 10.1088/1742-5468/2005/09/p09008.

[49] De Castro, L. N. and Von Zuben, F. J. 'Learning and optimization using the clonal selection principle'. In: *IEEE Transactions on Evolutionary Computation* 6.3 (2002), pp. 239–251. DOI: 10.1109/TEVC.2002.1011539.

[50] De Castro, L. N. and Von Zuben, F. J. 'The Clonal Selection Algorithm with Engineering Applications'. In: *Workshop on Artificial Immune Systems and Their Applications (GECCO '00).* 2000, pp. 36–37.

[51] Deco, G. and Corbetta, M. 'The Dynamical Balance of the Brain at Rest'. In: *The Neuroscientist* 17.1 (2011), pp. 107–123. DOI: 10.1177/1073858409354384.

[52] Dell'Amico, M., Lodi, A. and Maffioli, F. 'Solution of the cumulative assignment problem with a well-structured tabu search method'. In: *Journal of Heuristics* 5.2 (1999), pp. 123–143. DOI: 10.1023/A:1009647225748.

[53] Dezső, B., Jüttner, A. and Kovács, P. 'LEMON – an Open Source C++ Graph Template Library'. In: *Electronic Notes in Theoretical Computer Science* 264.5 (2011), pp. 23–45. DOI: 10.1016/j.entcs.2011.06.003.

[54] Di Stefano, A., Vitale, A., Cutello, V. and Pavone, M. ‘How long should offspring lifespan be in order to obtain a proper exploration?’ In: *2016 IEEE Symposium Series on Computational Intelligence (SSCI 2016)*. Dec. 2016, pp. 1–8. DOI: 10.1109/SSCI.2016.7850270.

[55] Didimo, W. and Montecchiani, F. ‘Fast layout computation of clustered networks: Algorithmic advances and experimental analysis’. In: *Information Sciences* 260 (2014), pp. 185–199. DOI: 10.1016/j.ins.2013.09.048.

[56] Diss, G., Filteau, M., Freschi, L., Leducq, J.-B., Rochette, S., Torres-Quiroz, F. and Landry, C. R. ‘Integrative avenues for exploring the dynamics and evolution of protein interaction networks’. In: *Current Opinion in Biotechnology* 24.4 (2013), pp. 775–783. DOI: 10.1016/j.copbio.2013.02.023.

[57] Duch, J. and Arenas, A. ‘Community detection in complex networks using extremal optimization’. In: *Physical Review E* 72.2 (Aug. 2005), p. 027104. DOI: 10.1103/PhysRevE.72.027104.

[58] Erol, O. K. and Eksin, I. ‘A new optimization method: Big Bang–Big Crunch’. In: *Advances in Engineering Software* 37.2 (2006), pp. 106–111. DOI: 10.1016/j.advengsoft.2005.04.005.

[59] Feo, T. A., Resende, M. G. C. and Smith, S. H. ‘A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set’. In: *Operations Research* 42.5 (1994), pp. 860–878. DOI: 10.1287/opre.42.5.860.

[60] Festa, P., Pardalos, P. M. and Resende, M. G. C. ‘Feedback Set Problems’. In: *Handbook of Combinatorial Optimization: Supplement Volume A*. Ed. by Du, D.-Z. and Pardalos, P. M. Boston, MA: Springer, 1999, pp. 209–258. DOI: 10.1007/978-1-4757-3023-4_4.

[61] Festa, P., Pardalos, P. M. and Resende, M. G. C. ‘Feedback Set Problems’. In: *Encyclopedia of Optimization*. Ed. by Floudas, C. A. and Pardalos, P. M. Boston, MA: Springer, 2009, pp. 1005–1016. DOI: 10.1007/978-0-387-74759-0_178.

[62] Fortunato, S. ‘Community detection in graphs’. In: *Physics Reports* 486.3 (2010), pp. 75–174. DOI: 10.1016/j.physrep.2009.11.002.

[63] Fortunato, S. and Barthélemy, M. ‘Resolution limit in community detection’. In: *Proceedings of the National Academy of Sciences* 104.1 (2007), pp. 36–41. DOI: 10.1073/pnas.0605965104.

[64] Fouladvand, S., Osareh, A., Shadgar, B., Pavone, M. and Sharafi, S. ‘DENSA: An effective negative selection algorithm with flexible boundaries for self-space and dynamic number of detectors’. In: *Engineering Applications of Artificial Intelligence* 62 (2017), pp. 359–372. DOI: 10.1016/j.engappai.2016.08.014.

[65] Gach, O. and Hao, J.-K. ‘Combined neighborhood tabu search for community detection in complex networks’. In: *RAIRO-Oper. Res.* 50.2 (2016), pp. 269–283. DOI: 10.1051/ro/2015046.

[66] Girvan, M. and Newman, M. E. J. ‘Community structure in social and biological networks’. In: *Proceedings of the National Academy of Sciences* 99.12 (2002), pp. 7821–7826. DOI: 10.1073/pnas.122653799.

[67] Gleiser, P. M. and Danon, L. ‘Community structure in Jazz’. In: *Advances in Complex Systems* 06.04 (2003), pp. 565–573. DOI: 10.1142/S0219525903001067.

[68] Glover, F. ‘Heuristics for integer programming using surrogate constraints’. In: *Decision Sciences* 8.1 (1977), pp. 156–166. DOI: 10.1111/j.1540-5915.1977.tb01074.x.

[69] Goh, K.-I., Cusick, M. E., Valle, D., Childs, B., Vidal, M. and Barabási, A.-L. ‘The human disease network’. In: *Proceedings of the National Academy of Sciences* 104.21 (2007), pp. 8685–8690. DOI: 10.1073/pnas.0701361104.

[70] Goldberg, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

[71] Good, B. H., De Montjoye, Y.-A. and Clauset, A. 'Performance of modularity maximization in practical contexts'. In: *Physical Review E* 81.4 (Apr. 2010), p. 046106. DOI: 10.1103/PhysRevE.81.046106.

[72] Gosavi, A. 'Reinforcement Learning: A Tutorial Survey and Recent Advances'. In: *INFORMS Journal on Computing* 21.2 (2009), pp. 178–192. DOI: 10.1287/ijoc.1080.0305.

[73] Gu, H., Zhu, P., Jiao, Y., Meng, Y. and Chen, M. 'PRIN: a predicted rice interactome network'. In: *BMC Bioinformatics* 12.161 (2011). DOI: 10.1186/1471-2105-12-161.

[74] Guimerà, R., Danon, L., Díaz-Guilera, A., Giralt, F. and Arenas, A. 'Self-similar community structure in a network of human interactions'. In: *Physical Review E* 68.6 (Dec. 2003), p. 065103. DOI: 10.1103/PhysRevE.68.065103.

[75] Gulbahce, N. and Lehmann, S. 'The art of community detection'. In: *BioEssays* 30.10 (2008), pp. 934–938. DOI: 10.1002/bies.20820.

[76] Gusfield, D. 'A Graph Theoretic Approach to Statistical Data Security'. In: *SIAM Journal on Computing* 17.3 (1988), pp. 552–571. DOI: 10.1137/0217034.

[77] Holland, J. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press, 1992.

[78] Hoos, H. H. and Stützle, T. *Stochastic Local Search: Foundations & Applications*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[79] Hubert, L. and Arabic, P. 'Comparing Partitions'. In: *Journal of Classification* 2 (1985), pp. 193–218. DOI: 10.1007/BF01908075.

[80] Jansen, T. and Zarges, C. 'On benefits and drawbacks of aging strategies for randomized search heuristics'. In: *Theoretical Computer Science* 412.6 (2011), pp. 543–559. DOI: 10.1016/j.tcs.2010.03.032.

[81] Jansen, T. and Zarges, C. ‘On the Role of Age Diversity for Effective Aging Operators’. In: *Evolutionary Intelligence* 4.2 (2011), pp. 99–125. DOI: 10.1007/s12065-011-0051-6.

[82] Jaynes, E. T. *Probability Theory: The Logic of Science*. Cambridge University Press, 2003.

[83] Johnson, D. B. ‘Finding All the Elementary Circuits of a Directed Graph’. In: *SIAM Journal on Computing* 4.1 (1975), pp. 77–84. DOI: 10.1137/0204007.

[84] Kaelbling, L. P., Littman, M. L. and Moore, A. W. ‘Reinforcement Learning: A Survey’. In: *Journal of Artificial Intelligence Research* 4 (May 1996), pp. 237–285. DOI: 10.1613/jair.301.

[85] Karp, R. M. ‘Reducibility among Combinatorial Problems’. In: *Complexity of Computer Computations*. Ed. by Miller, R. E., Thatcher, J. W. and Bohlinger, J. D. Boston, MA: Springer, 1972, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9.

[86] Kernighan, B. W. and Lin, S. ‘An efficient heuristic procedure for partitioning graphs’. In: *The Bell System Technical Journal* 49.2 (Feb. 1970), pp. 291–307. DOI: 10.1002/j.1538-7305.1970.tb01770.x.

[87] Kratsch, S. and Schweitzer, P. ‘Isomorphism for Graphs of Bounded Feedback Vertex Set Number’. In: *Algorithm Theory - SWAT 2010*. Ed. by Kaplan, H. Berlin, Heidelberg: Springer, 2010, pp. 81–92. DOI: 10.1007/978-3-642-13731-0_9.

[88] Krebs, V. *A network of books about recent US politics sold by the online bookseller Amazon.com*. 2008. URL: http://www.orgnet.com.

[89] Kullback, S. and Leibler, R. A. ‘On Information and Sufficiency’. In: *The Annals of Mathematical Statistics* 22.1 (1951), pp. 79–86. DOI: 10.1214/aoms/1177729694.

[90] Kullback, S. *Information Theory and Statistics*. Wiley, 1959.

[91] Lancichinetti, A. and Fortunato, S. 'Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities'. In: *Physical Review E* 80.1 (July 2009), p. 016118. DOI: 10.1103/PhysRevE.80.016118.

[92] Lancichinetti, A., Fortunato, S. and Radicchi, F. 'Benchmark graphs for testing community detection algorithms'. In: *Physical Review E* 78.4 (Oct. 2008), p. 046110. DOI: 10.1103/PhysRevE.78.046110.

[93] Lee, D.-S., Park, J., Kay, K. A., Christakis, N. A., Oltvai, Z. N. and Barabási, A.-L. 'The implications of human metabolic network topology for disease comorbidity'. In: *Proceedings of the National Academy of Sciences* 105.29 (2008), pp. 9880–9885. DOI: 10.1073/pnas.0802208105.

[94] Li, L., Wei, Z., Hao, J.-K. and He, K. 'Probability learning based tabu search for the budgeted maximum coverage problem'. In: *Expert Systems with Applications* 183 (2021), p. 115310. DOI: 10.1016/j.eswa.2021.115310.

[95] Li, M., Hao, J.-K. and Wu, Q. 'Learning-driven feasible and infeasible tabu search for airport gate assignment'. In: *European Journal of Operational Research* 302.1 (2022), pp. 172–186. DOI: 10.1016/j.ejor.2021.12.019.

[96] Li, W., Kang, Q., Kong, H., Liu, C. and Kang, Y. 'A novel iterated greedy algorithm for detecting communities in complex network'. In: *Social Network Analysis and Mining* 10.29 (2020), pp. 1–17. DOI: 10.1007/s13278-020-00641-y.

[97] Lim, Y. H., Charette, J. M. and Baserga, S. J. 'Assembling a Protein-Protein Interaction Map of the SSU Processome from Existing Datasets'. In: *PLOS ONE* 6.3 (Mar. 2011), e17701. DOI: 10.1371/journal.pone.0017701.

[98] Lusseau, D., Schneider, K., Boisseau, O. J., Haase, P., Slooten, E. and Dawson, S. M. 'The bottlenose dolphin community of Doubtful Sound features a large proportion of long-lasting associations'. In: *Behavioral Ecology and Sociobiology* 54 (2003), pp. 396–405. DOI: 10.1007/s00265-003-0651-y.

[99] Marbach, D., Costello, J. C., Küffner, R., Vega, N. M., Prill, R. J., Camacho, D. M., Allison, K. R., Kellis, M., Collins, J. J. and Stolovitzky, G. 'Wisdom of crowds for robust gene network inference'. In: *Nature Methods* 9.8 (2012), pp. 796–804. DOI: 10.1038/nmeth.2016.

[100] Martí, R., Laguna, M. and Glover, F. 'Principles of scatter search'. In: *European Journal of operational Research* 169.2 (2006), pp. 359–372. DOI: 10.1016/j.ejor.2004.08.004.

[101] Meilă, M. 'Comparing clusterings–an information based distance'. In: *Journal of Multivariate Analysis* 98.5 (2007), pp. 873–895. DOI: 10.1016/j.jmva.2006.11.013.

[102] Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D. and Alon, U. 'Network Motifs: Simple Building Blocks of Complex Networks'. In: *Science* 298.5594 (2002), pp. 824–827. DOI: 10.1126/science.298.5594.824.

[103] Mullard, A. 'Protein–protein interaction inhibitors get into the groove'. In: *Nature Reviews Drug Discovery* 11.3 (2012), pp. 173–175. DOI: 10.1038/nrd3680.

[104] Naeni, L. M., Berretta, R. and Moscato, P. 'MA-Net: A Reliable Memetic Algorithm for Community Detection by Modularity Optimization'. In: *Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems, Volume 1*. Ed. by Handa, H., Ishibuchi, H., Ong, Y.-S. and Tan, K. C. Cham: Springer, 2015, pp. 311–323. DOI: 10.1007/978-3-319-13359-1_25.

[105] Newman, M. E. J. 'Communities, modules and large-scale structure in networks'. In: *Nature Physics* 8.1 (2012), pp. 25–31. DOI: 10.1038/nphys2162.

[106] Newman, M. E. J. 'Fast algorithm for detecting community structure in networks'. In: *Physical Review E* 69.6 (June 2004), p. 066133. DOI: 10.1103/PhysRevE.69.066133.

[107] Newman, M. E. J. 'Finding community structure in networks using the eigenvectors of matrices'. In: *Physical Review E* 74.3 (Sept. 2006), p. 036104. DOI: 10.1103/PhysRevE.74.036104.

[108] Newman, M. E. J. 'The Structure and Function of Complex Networks'. In: *SIAM Review* 45.2 (2003), pp. 167–256. DOI: 10.1137/S003614450342480.

[109] Newman, M. E. J. and Girvan, M. 'Finding and evaluating community structure in networks'. In: *Physical Review E* 69.2 (Feb. 2004), p. 026113. DOI: 10.1103/PhysRevE.69.026113.

[110] Noack, A. and Rotta, R. 'Multi-level algorithms for modularity clustering'. In: *Experimental Algorithms (SEA 2009)*. Ed. by Vahrenhold, J. Berlin, Heidelberg: Springer, 2009, pp. 257–268. DOI: 10.1007/978-3-642-02011-7_24.

[111] Pavone, M., Narzisi, G. and Nicosia, G. 'Clonal selection: an immunological algorithm for global optimization over continuous spaces'. In: *Journal of Global Optimization* 53.4 (2012), pp. 769–808. DOI: 10.1007/s10898-011-9736-8.

[112] Peleg, D. 'Size bounds for dynamic monopolies'. In: *Discrete Applied Mathematics* 86.2 (1998), pp. 263–273. DOI: 10.1016/S0166-218X(98)00043-2.

[113] Poggiolini, M. and Engelbrecht, A. 'Application of the feature-detection rule to the Negative Selection Algorithm'. In: *Expert Systems with Applications* 40.8 (2013), pp. 3001–3014. DOI: 10.1016/j.eswa.2012.12.016.

[114] Pólya, G. *How to Solve It*. Princeton University Press, 1945.

[115] Porter, M. A., Onnela, J.-P. and Mucha, P. J. 'Communities in networks'. In: *Notices of the AMS* 56.9 (2009), pp. 1082–1097.

[116] Rain, J.-C., Selig, L., De Reuse, H., Battaglia, V., Reverdy, C., Simon, S., Lenzen, G., Petel, F., Wojcik, J., Schächter, V., Chemama, Y., Labigne, A. and Legrain, P. 'The protein-–protein interaction map of Helicobacter pylori'. In: *Nature* 409.6817 (2001), pp. 211–215. DOI: 10.1038/35051615.

[117] Rashedi, E., Nezamabadi-pour, H. and Saryazdi, S. 'GSA: A Gravitational Search Algorithm'. In: *Information Sciences* 179.13 (2009), pp. 2232–2248. DOI: 10.1016/j.ins.2009.03.004.

[118] Ravasi, T., Suzuki, H., Cannistraci, C. V., Katayama, S., Bajic, V. B., Tan, K., Akalin, A., Schmeier, S., Kanamori-Katayama, M., Bertin, N., Carninci, P., Daub, C. O., Forrest, A. R. R., Gough, J., Grimmond, S., Han, J.-H., Hashimoto, T., Hide, W., Hofmann, O., Kamburov, A., Kaur, M., Kawaji, H., Kubosaki, A., Lassmann, T., Nimwegen, E. van, MacPherson, C. R., Ogawa, C., Radovanovic, A., Schwartz, A., Teasdale, R. D., Tegnér, J., Lenhard, B., Teichmann, S. A., Arakawa, T., Ninomiya, N., Murakami, K., Tagami, M., Fukuda, S., Imamura, K., Kai, C., Ishihara, R., Kitazume, Y., Kawai, J., Hume, D. A., Ideker, T. and Hayashizaki, Y. 'An Atlas of Combinatorial Transcriptional Regulation in Mouse and Man'. In: *Cell* 140.5 (2010), pp. 744–752. DOI: 10.1016/j.cell.2010.01.044.

[119] Ross, R., Dagnone, D., Jones, P. J. H., Smith, H., Paddags, A., Hudson, R. and Janssen, I. 'Reduction in Obesity and Related Comorbid Conditions after Diet-Induced Weight Loss or Exercise-Induced Weight Loss in Men'. In: *Annals of Internal Medicine* 133.2 (2000), pp. 92–103. DOI: 10.7326/0003-4819-133-2-200007180-00008.

[120] Ruiz, R. and Stützle, T. 'A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem'. In: *European Journal of Operational Research* 177.3 (2007), pp. 2033–2049. DOI: 10.1016/j.ejor.2005.12.009.

[121] Said, A., Abbasi, R. A., Maqbool, O., Daud, A. and Aljohani, N. R. 'CC-GA: A clustering coefficient based genetic algorithm for detecting communities in social networks'. In: *Applied Soft Computing* 63 (2018), pp. 59–70. DOI: 10.1016/j.asoc.2017.11.014.

[122] Sánchez-Oro, J. and Duarte, A. 'Iterated Greedy algorithm for performing community detection in social networks'. In: *Future Generation Computer Systems* 88 (2018), pp. 785–791. DOI: 10.1016/j.future.2018.06.010.

[123] Schellenberger, J., Park, J. O., Conrad, T. M. and Palsson, B. Ø. 'BiGG: a Biochemical Genetic and Genomic knowledgebase of large scale metabolic reconstructions'. In: *BMC Bioinformatics* 11.213 (2010). DOI: 10.1186/1471-2105-11-213.

[124] Schuetz, P. and Caflisch, A. 'Efficient modularity optimization by multistep greedy algorithm and vertex mover refinement'. In: *Physical Review E* 77.4 (Apr. 2008), p. 046112. DOI: 10.1103/PhysRevE.77.046112.

[125] Scollo, R. A., Cutello, V. and Pavone, M. 'Where the Local Search Affects Best in an Immune Algorithm'. In: *AIxIA 2020 – Advances in Artificial Intelligence (AIxIA 2020)*. Ed. by Baldoni, M. and Bandini, S. Vol. 12414. Lecture Notes in Artificial Intelligence. Cham: Springer, 2021, pp. 99–114. DOI: 10.1007/978-3-030-77091-4_7.

[126] Scollo, R. A., Spampinato, A. G., Fargetta, G., Cutello, V. and Pavone, M. 'Discovering Entities Similarities in Biological Networks Using a Hybrid Immune Algorithm'. In: *Informatics* 10.1 (2023). DOI: 10.3390/informatics10010018.

[127] Shamir, A. 'A Linear Time Algorithm for Finding Minimum Cutsets in Reducible Graphs'. In: *SIAM Journal on Computing* 8.4 (1979), pp. 645–655. DOI: 10.1137/0208051.

[128] Shannon, C. E. 'A Mathematical Theory of Communication'. In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[129] Shen-Orr, S. S., Milo, R., Mangan, S. and Alon, U. 'Network motifs in the transcriptional regulation network of Escherichia coli'. In: *Nature Genetics* 31.1 (May 2002), pp. 64–68. DOI: 10.1038/ng881.

[130] Shi, C., Wang, Y., Wu, B. and Zhong, C. 'A New Genetic Algorithm for Community Detection'. In: *Complex Sciences (Complex 2009)*. Ed. by Zhou, J. Berlin, Heidelberg: Springer, 2009, pp. 1298–1309. DOI: 10.1007/978-3-642-02469-6_11.

[131] Smith, S. L. and Timmis, J. 'An immune network inspired evolutionary algorithm for the diagnosis of Parkinson's disease'. In: *Biosystems* 94.1 (2008), pp. 34–46. DOI: 10.1016/j.biosystems.2008.05.024.

[132] Spampinato, A. G., Scollo, R. A., Cavallaro, S., Pavone, M. and Cutello, V. 'An Immunological Algorithm for Graph Modularity Optimization'. In: *Advances in Computational Intelligence Systems (UKCI 2019)*. Ed. by Ju, Z., Yang, L., Yang, C., Gegov, A. and Zhou, D. Vol. 1043. Advances in Intelligent Systems and Computing. Cham: Springer, 2020, pp. 235–247. DOI: 10.1007/978-3-030-29933-0_20.

[133] Storn, R. and Price, K. 'Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces'. In: *Journal of Global Optimization* 11.4 (1997), pp. 341–359. DOI: 10.1023/A:1008202821328.

[134] Storn, R. and Price, K. *Differential Evolution: A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces*. Tech. rep. TR-95-012. Berkeley, CA, USA: International Computer Science Institute, 1995.

[135] Sundaresan, S. R., Fischhoff, I. R., Dushoff, J. and Rubenstein, D. I. 'Network metrics reveal differences in social organization between two fission–fusion species, Grevy's zebra and onager'. In: *Oecologia* 151 (2007), pp. 140–149. DOI: 10.1007/s00442-006-0553-6.

[136] Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[137] Talbi, E.-G. 'A Unified Taxonomy of Hybrid Metaheuristics with Mathematical Programming, Constraint Programming and Machine Learning'. In: *Hybrid Me-*

taheuristics. Ed. by Talbi, E.-G. Berlin, Heidelberg: Springer, 2013, pp. 3–76. DOI: 10.1007/978-3-642-30671-6_1.

[138] Talbi, E.-G. *Metaheuristics: from Design to Implementation*. Wiley Publishing, 2009.

[139] Tang, B., Hsu, H.-K., Hsu, P.-Y., Bonneville, R., Chen, S.-S., Huang, T. H. M. and Jin, V. X. 'Hierarchical Modularity in ERα Transcriptional Network Is Associated with Distinct Functions and Implicates Clinical Outcomes'. In: *Scientific Reports* 2 (2012), p. 875. DOI: 10.1038/srep00875.

[140] Taylor, I. W., Linding, R., Warde-Farley, D., Liu, Y., Pesquita, C., Faria, D., Bull, S., Pawson, T., Morris, Q. and Wrana, J. L. 'Dynamic modularity in protein interaction networks predicts breast cancer outcome'. In: *Nature Biotechnology* 27.2 (2009), pp. 199–204. DOI: 10.1038/nbt.1522.

[141] Vitale, A., Di Stefano, A., Cutello, V. and Pavone, M. 'The Influence of Age Assignments on the Performance of Immune Algorithms'. In: *Advances in Computational Intelligence Systems (UKCI 2018)*. Ed. by Lotfi, A., Bouchachia, H., Gegov, A., Langensiepen, C. and McGinnity, M. Cham: Springer, 2019, pp. 16–28. DOI: 10.1007/978-3-319-97982-3_2.

[142] Von Mering, C., Krause, R., Snel, B., Cornell, M., Oliver, S. G., Fields, S. and Bork, P. 'Comparative assessment of large-scale data sets of protein–protein interactions'. In: *Nature* 417.6887 (2002), pp. 399–403. DOI: 10.1038/nature750.

[143] Wang, C.-C., Lloyd, E. L. and Soffa, M. L. 'Feedback Vertex Sets and Cyclically Reducible Graphs'. In: *Journal of the ACM* 32.2 (Apr. 1985), pp. 296–313. DOI: 10.1145/3149.3159.

[144] Wang, X. and Tang, L. 'A machine-learning based memetic algorithm for the multi-objective permutation flowshop scheduling problem'. In: *Computers & Operations Research* 79 (2017), pp. 60–77. DOI: 10.1016/j.cor.2016.10.003.

[145] Wang, Y., Pan, S., Li, C. and Yin, M. 'A local search algorithm with reinforcement learning based repair procedure for minimum weight independent dominating set'. In: *Information Sciences* 512 (2020), pp. 533–548. DOI: 10.1016/j.ins.2019.09.059.

[146] Watts, D. J. and Strogatz, S. H. 'Collective dynamics of 'small-world' networks'. In: *Nature* 393.6684 (1998), pp. 440–442. DOI: 10.1038/30918.

[147] Wauters, T., Verbeeck, K., De Causmaecker, P. and Vanden Berghe, G. 'Boosting Metaheuristic Search Using Reinforcement Learning'. In: *Hybrid Metaheuristics*. Ed. by Talbi, E.-G. Berlin, Heidelberg: Springer, 2013, pp. 433–452. DOI: 10.1007/978-3-642-30671-6_17.

[148] Wilkinson, D. M. and Huberman, B. A. 'A method for finding communities of related genes'. In: *Proceedings of the National Academy of Sciences* 101.suppl 1 (2004), pp. 5241–5248. DOI: 10.1073/pnas.0307740100.

[149] Xenarios, I., Rice, D. W., Salwinski, L., Baron, M. K., Marcotte, E. M. and Eisenberg, D. 'DIP: the Database of Interacting Proteins'. In: *Nucleic Acids Research* 28.1 (Jan. 2000), pp. 289–291. DOI: 10.1093/nar/28.1.289.

[150] Yang, X.-S. 'A New Metaheuristic Bat-Inspired Algorithm'. In: *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*. Ed. by González, J. R., Pelta, D. A., Cruz, C., Terrazas, G. and Krasnogor, N. Berlin, Heidelberg: Springer, 2010, pp. 65–74. DOI: 10.1007/978-3-642-12538-6_6.

[151] Yannakakis, M. 'Node-and Edge-Deletion NP-Complete Problems'. In: *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing (STOC '78)*. New York, NY, USA: Association for Computing Machinery, 1978, pp. 253–264. DOI: 10.1145/800133.804355.

[152] Yanrui, D., Zhen, Z., Wenchao, W. and Yujie, C. 'Identifying the Communities in the Metabolic Network Using 'Component' Definition and Girvan-Newman Algorithm'. In: *2015 14th International Symposium on Distributed Computing*

and Applications for Business Engineering and Science (DCABES 2015). 2015, pp. 42–45. DOI: 10.1109/DCABES.2015.18.

[153] Yu, H., Braun, P., Yildirim, M. A., Lemmens, I., Venkatesan, K., Sahalie, J., Hirozane-Kishikawa, T., Gebreab, F., Li, N., Simonis, N., Hao, T., Rual, J.-F., Dricot, A., Vazquez, A., Murray, R. R., Simon, C., Tardivo, L., Tam, S., Svrzikapa, N., Fan, C., De Smet, A.-S., Motyl, A., Hudson, M. E., Park, J., Xin, X., Cusick, M. E., Moore, T., Boone, C., Snyder, M., Roth, F. P., Barabási, A.-L., Tavernier, J., Hill, D. E. and Vidal, M. 'High-Quality Binary Protein Interaction Map of the Yeast Interactome Network'. In: *Science* 322.5898 (2008), pp. 104–110. DOI: 10.1126/science.1158684.

[154] Zachary, W. W. 'An Information Flow Model for Conflict and Fission in Small Groups'. In: *Journal of Anthropological Research* 33.4 (1977), pp. 452–473. DOI: 10.1086/jar.33.4.3629752.

[155] Zhang, Y., Gao, P. and Yuan, J. S. 'Plant Protein-Protein Interaction Network and Interactome'. In: *Current Genomics* 11.1 (2010), pp. 40–46. DOI: 10.2174/138920210790218016.

[156] Zhou, Y., Duval, B. and Hao, J.-K. 'Improving probability learning based local search for graph coloring'. In: *Applied Soft Computing* 65 (2018), pp. 542–553. DOI: 10.1016/j.asoc.2018.01.027.

[157] Zhou, Y., Hao, J.-K. and Duval, B. 'Opposition-Based Memetic Search for the Maximum Diversity Problem'. In: *IEEE Transactions on Evolutionary Computation* 21.5 (2017), pp. 731–745. DOI: 10.1109/TEVC.2017.2674800.

[158] Zhou, Y., Hao, J.-K. and Duval, B. 'Reinforcement learning based local search for grouping problems: A case study on graph coloring'. In: *Expert Systems with Applications* 64 (2016), pp. 412–422. DOI: 10.1016/j.eswa.2016.07.047.

[159] Zhu, J., Zhang, B., Smith, E. N., Drees, B., Brem, R. B., Kruglyak, L., Bumgarner, R. E. and Schadt, E. E. 'Integrating large-scale functional gen-

omic data to dissect the complexity of yeast regulatory networks'. In: *Nature Genetics* 40.7 (2008), pp. 854–861. DOI: 10.1038/ng.167.