

Clonal selection: an immunological algorithm for global optimization over continuous spaces

Mario Pavone · Giuseppe Narzisi · Giuseppe Nicosia

Received: 7 October 2009 / Accepted: 23 May 2011
© Springer Science+Business Media, LLC. 2011

Abstract In this research paper we present an immunological algorithm (IA) to solve global numerical optimization problems for high-dimensional instances. Such optimization problems are a crucial component for many real-world applications. We designed two versions of the IA: the first based on binary-code representation and the second based on real values, called OPT-IMMALG01 and OPT-IMMALG, respectively. A large set of experiments is presented to evaluate the effectiveness of the two proposed versions of IA. Both OPT-IMMALG01 and OPT-IMMALG were extensively compared against several nature inspired methodologies including a set of Differential Evolution algorithms whose performance is known to be superior to many other bio-inspired and deterministic algorithms on the same test bed. Also hybrid and deterministic global search algorithms (e.g., DIRECT, LEGO, PSWARM) are compared with both IA versions, for a total 39 optimization algorithms. The results suggest that the proposed immunological algorithm is effective, in terms of accuracy, and capable of solving large-scale instances for well-known benchmarks. Experimental results also indicate that both IA versions are comparable, and often outperform, the state-of-the-art optimization algorithms.

Keywords Nonlinear optimization · Global optimization · Derivative-free optimization · Black-box optimization · Immunological algorithms · Evolutionary algorithms

M. Pavone · G. Nicosia (✉)
Department of Mathematics and Computer Science, University of Catania, Viale A. Doria 6,
95125 Catania, Italy
e-mail: nicosia@dmi.unict.it

M. Pavone
e-mail: mpavone@dmi.unict.it

G. Narzisi
Computer Science Department, Courant Institute of Mathematical Sciences,
New York University, New York, NY 10012, USA
e-mail: narzisi@nyu.edu

1 Introduction

Artificial Immune Systems (AIS) is a paradigm in biologically inspired computing, which has been successfully applied to several real-world applications in computer science and engineering [17, 23, 37–39]. AIS are bio-inspired algorithms that take their inspiration from the natural immune system, whose function is to detect and protect the organism against foreign organisms, like viruses, bacteria, fungi and parasites, that can be cause of diseases. The main research work on AIS was concentrated primarily on three immunological theories: (1) immune networks, (2) negative selection and (3) clonal selection. Such algorithms have been successfully employed in a variety of different application areas [18, 35]. All algorithms based on the simulation of the clonal selection principle are included into a special class called *Clonal Selection Algorithms* (CSA), and represents an effective mechanism for searching and optimization [13, 15, 16]. The core components of the CSAs are *cloning* and *hypermutation* operators: the first triggers the growth of a new population of high-value B cells (the candidate solutions) centered on a higher affinity value, whereas the last can be seen as a local search procedure that leads to a faster *maturation* during the learning phase.

We designed and implemented an Immunological Algorithm (IA) to tackle the global numerical optimization problems, based on CSAs. We give two different versions of the proposed IA, using either binary-code or real values representations, called respectively OPT-IMMALG01 and OPT-IMMALG.

Global optimization is the task of finding the best set of parameters to optimize a given objective function; global optimization problems are typically quite difficult to solve because of the presence of many locally optimal solutions [22]. In many real-world applications analytical solutions, even for simple problems, are not always available, so numerical continuous optimization by approximate methods is often the only viable alternative [33, 22].

The global optimization consists of finding a variable (or a set of variables) $\mathbf{x} = (x_1, x_2, \dots, x_n) \in S$, where $S \subseteq \mathbb{R}^n$ is a bounded set on \mathbb{R}^n , such that a certain n -dimensional objective function $f : S \rightarrow \mathbb{R}$ is optimized. Specifically the goal for global minimization problem is to find a point $\mathbf{x}_{\min} \in S$ such that $f(\mathbf{x}_{\min})$ is a global minimum on S , i.e. $\forall \mathbf{x} \in S : f(\mathbf{x}_{\min}) \leq f(\mathbf{x})$. The problem of continuous optimization is a difficult task for three main reasons [33]: (1) it is difficult to decide when a global (or local) optimum has been reached; (2) there could be many local optimal solutions where the search algorithm can get trapped; (3) the number of suboptimal solutions grows dramatically with the dimension of the search space [22].

In this research, we consider the following numerical minimization problem:

$$\min(f(\mathbf{x})), \quad \mathbf{B}_l \leq \mathbf{x} \leq \mathbf{B}_u \quad (1)$$

where $\mathbf{x} = (x_1, x_2, \dots, x_n)$ is the variable vector in \mathbb{R}^n , $f(\mathbf{x})$ denotes the objective function to *minimize* and $\mathbf{B}_l = (B_{l_1}, B_{l_2}, \dots, B_{l_n})$, $\mathbf{B}_u = (B_{u_1}, B_{u_2}, \dots, B_{u_n})$ represent, respectively, the lower and the upper bounds of the variables, such that $x_i \in [B_{l_i}, B_{u_i}]$ ($i = 1, \dots, n$).

To evaluate the performance and convergence ability of the proposed IAs compared to the state-of-the-art optimization algorithms [22], we have used the classic benchmark proposed in Yao et al. [43], that includes twenty-three functions (see Table 1 in Sect. 3.1). These functions belong to three different categories: unimodal, multimodal with many local optima, and multimodal with few local optima. Moreover we compare both IA versions with several immunological algorithms. For some of these experiments we tackled the functions proposed in Timmis and Kelsey [40] (Table 2, described in Sect. 3.1).

The paper is structured as follows: in Sect. 2 we describe the proposed immunological algorithm and its main features; in Sect. 3 we describe the benchmark and the metrics used

Table 1 First class of functions to optimize [43]

Test function	n	S
$f_1(\mathbf{x}) = \sum_{i=1}^n x_i^2$	30	$[-100, 100]^n$
$f_2(\mathbf{x}) = \sum_{i=1}^n x_i + \prod_{i=1}^n x_i $	30	$[-10, 10]^n$
$f_3(\mathbf{x}) = \sum_{i=1}^n \left(\sum_{j=1}^i x_j \right)^2$	30	$[-100, 100]^n$
$f_4(\mathbf{x}) = \max_i \{ x_i , 1 \leq i \leq n\}$	30	$[-100, 100]^n$
$f_5(\mathbf{x}) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	30	$[-30, 30]^n$
$f_6(\mathbf{x}) = \sum_{i=1}^n (x_i + 0.5)^2$	30	$[-100, 100]^n$
$f_7(\mathbf{x}) = \sum_{i=1}^n i \cdot x_i^4 + \text{random}[0, 1]$	30	$[-1.28, 1.28]^n$
$f_8(\mathbf{x}) = \sum_{i=1}^n -x_i \sin(\sqrt{ x_i })$	30	$[-500, 500]^n$
$f_9(\mathbf{x}) = \sum_{i=1}^n [x_i^2 - 10 \cos(2\pi x_i) + 10]$	30	$[-5.12, 5.12]^n$
$f_{10}(\mathbf{x}) = -20 \exp\left(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2}\right) - \exp\left(\frac{1}{n} \sum_{i=1}^n \cos 2\pi x_i\right) + 20 + e$	30	$[-32, 32]^n$
$f_{11}(\mathbf{x}) = \frac{1}{4000} \sum_{i=1}^n x_i^2 - \prod_{i=1}^n \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	30	$[-600, 600]^n$
$f_{12}(\mathbf{x}) = \frac{\pi}{n} \{10 \sin^2(\pi y_1) + \sum_{i=1}^{n-1} (y_i - 1)^2 [1 + 10 \sin^2(\pi y_{i+1})] + (y_n - 1)^2\} + \sum_{i=1}^n u(x_i, 10, 100, 4)$ $y_i = 1 + \frac{1}{4}(x_i + 1)$	30	$[-50, 50]^n$
$u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & \text{if } x_i > a, \\ 0, & \text{if } -a \leq x_i \leq a, \\ k(-x_i - a)^m, & \text{if } x_i < -a. \end{cases}$		
$f_{13}(\mathbf{x}) = 0.1 \{\sin^2(3\pi x_1) + \sum_{i=1}^{n-1} (x_i - 1)^2 [1 + \sin^2(3\pi x_{i+1})] + (x_n - 1) [1 + \sin^2(2\pi x_n)]\} + \sum_{i=1}^n u(x_i, 5, 100, 4)$	30	$[-50, 50]^n$
$f_{14}(\mathbf{x}) = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^2 (x_i - a_{ij})^6} \right]^{-1}$	2	$[-65.536, 65.536]^n$
$f_{15}(\mathbf{x}) = \sum_{i=1}^{11} \left[a_i - \frac{x_i(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4} \right]^2$	4	$[-5, 5]^n$
$f_{16}(\mathbf{x}) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	2	$[-5, 5]^n$
$f_{17}(\mathbf{x}) = \left(x_2 - \frac{5.1}{4\pi^2} x_1^2 + \frac{5}{\pi} x_1 - 6 \right)^2 + 10 \left(1 - \frac{1}{8\pi} \right) \cos x_1 + 10$	2	$[-5, 10] \times [0, 15]$
$f_{18}(\mathbf{x}) = [1 + (x_1 + x_2 + 1)^2 (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times [30 + (2x_1 - 3x_2)^2 (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	2	$[-2, 2]^n$
$f_{19}(\mathbf{x}) = -\sum_{i=1}^4 c_i \exp\left[-\sum_{j=1}^4 a_{ij}(x_j - p_{ij})^2\right]$	4	$[0, 1]^n$
$f_{20}(\mathbf{x}) = -\sum_{i=1}^4 c_i \exp\left[-\sum_{j=1}^6 a_{ij}(x_j - p_{ij})^2\right]$	6	$[0, 1]^n$
$f_{21}(\mathbf{x}) = -\sum_{i=1}^5 \left[(\mathbf{x} - a_i)(\mathbf{x} - a_i)^T + c_i \right]^{-1}$	4	$[0, 10]^n$
$f_{22}(\mathbf{x}) = -\sum_{i=1}^7 \left[(\mathbf{x} - a_i)(\mathbf{x} - a_i)^T + c_i \right]^{-1}$	4	$[0, 10]^n$
$f_{23}(\mathbf{x}) = -\sum_{i=1}^{10} \left[(\mathbf{x} - a_i)(\mathbf{x} - a_i)^T + c_i \right]^{-1}$	4	$[0, 10]^n$

We indicate with n the number of variables employed and with $S \subseteq \mathcal{R}^n$ the variable bounds

Table 2 Second class of numerical functions [40], with $S \subseteq \mathcal{R}^n$ the variable bounds

Test function	S
$g_1(x) = 2(x - 0.75)^2 + \sin(5\pi x - 0.4\pi) - 0.125$	$0 \leq x \leq 1$
$g_2(x, y) = (4 - 2.1x^2 + \frac{x^4}{3})x^2 + xy + (-4 + 4y^2)y^2$	$-3 \leq x \leq 3$ $-2 \leq y \leq 2$
$g_3(x) = -\sum_{j=1}^5 [j \sin((j+1)x + j)]$	$-10 \leq x \leq 10$
$g_4(x, y) = a(y - bx^2 + cx - d)^2 + h(i - f) \cos(x) + h$	$a = 1, b = \frac{5.1}{4\pi^2}, c = \frac{5}{\pi}$ $d = 6, f = \frac{1}{8\pi}, h = 10$ $-5 \leq x \leq 10, 0 \leq y \leq 15$
$g_5(x, y) = \sum_{j=1}^5 j \cos[(j+1)x + j]$	$-10 \leq x \leq 10$ $-10 \leq y \leq 10, \beta = 0.5$
$g_6 = \sum_{j=1}^5 j \cos[(j+1)x + j] + \beta [(x + 1, 4513)^2 + (y + 0.80032)^2]$	$-10 \leq x \leq 10$ $-10 \leq y \leq 10, \beta = 1$
$g_7(x, y) = x \sin(4\pi x) - y \sin(4\pi y\pi) + 1$	$-10 \leq x \leq 10$ $-10 \leq y \leq 10$
$g_8(y) = \sin^6 5\pi x$	$-10 \leq x \leq 10$ $-10 \leq y \leq 10$
$g_9(x, y) = \frac{x^4}{4} - \frac{x^2}{2} + \frac{x}{10} + \frac{y^2}{2}$	$-10 \leq x \leq 10$ $-10 \leq y \leq 10$
$g_{10}(x, y) = \sum_{j=1}^5 j \cos[(j+i)x + j] \sum_{j=1}^5 j \cos[(j+i)y + j]$	$-10 \leq x \leq 10$ $-10 \leq y \leq 10$
$g_{11}(x) = 418.9829n - \sum_{i=1}^n nx_i \sin(\sqrt{ x_i })$	$-512.03 \leq x_i \leq 511.97n = 3$
$g_{12}(x) = 1 + \sum_{i=1}^n \frac{4000}{x_i^2} - \prod_{i=1}^n \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$-600 \leq x_i \leq 600$ $n = 20$

to compare OPT-IMMALG01 and OPT-IMMALG algorithms with the state-of-the-art optimization algorithms; in the same section we show the influence of the different potential mutations on the dynamics of both IAs; Sect. 4 presents a large set of experiments, comparing the two IA versions with several nature inspired methodologies; finally, Sect.5 contains the concluding remarks.

2 The immunological algorithm

In this section we describe the IA based on the clonal selection principle. The main features of the algorithm are: (i) cloning, (ii) inversely proportional hypermutation and (iii) aging operator. The cloning operator clones each candidate solution in order to explore its neighbourhood in the search space; the inversely proportional hypermutation perturbs each candidate solution using an inversely proportional law to its objective function value; and the aging operator eliminates old candidate solutions from the current population in order to introduce diversity and to avoid local minima during the evolutionary search process.

We present two versions of the IA: the first one is based on binary code representation (OPT-IMMALG01), and the second on real values (OPT-IMMALG). Both algorithms model antigens (Ag) and B cells; the Ag represents the problem to tackle, i.e. the function to optimize, while the B cell receptors are points (candidate solutions) in the search space for the problem. At each time step t the algorithm maintains a population of B cells $P^{(t)}$ of size d (i.e., d candidate solutions). Algorithm 1 shows the pseudo-code of the algorithm.

2.1 Initialize population

The population is initialized at time $t = 0$ (steps 1–4 in Algorithm 1) by randomly generating each solution using uniform distribution in the corresponding domains for each function (see last column of Tables 1, 2). For binary string representation, each real value x_i is coded using bit strings of length $k = 32$. The mapping from the binary string $\mathbf{b} = \langle b_1, b_2, \dots, b_k \rangle$ into a real number x consists of two steps: (1) convert the bit string $\mathbf{b} = \langle b_1, \dots, b_k \rangle$ from base 2 to base 10 : $\sum_{i=0}^{k-1} b_i * 2^{k-i} = x'$, (2) finding the corresponding real value:

$$x = B_{l_i} + \frac{x'(B_{u_i} - B_{l_i})}{2^k - 1} \tag{2}$$

B_{l_i} and B_{u_i} are the lower and upper bounds of the i th variable, respectively. In the case of real value representation, each variable is randomly initialized as follows:

$$x_i = B_{l_i} + \beta \cdot (B_{u_i} - B_{l_i}) \tag{3}$$

where β is a random number in $[0, 1]$ and B_{l_i}, B_{u_i} are the lower and upper bounds of the real coded variable x_i respectively. The strategy used to initialize the population plays a crucial role in evolutionary algorithms, since it influences the later performance of the algorithm. In traditional evolutionary computing, the initial population is generated using a random numbers distribution or chaotic sequences [4]. After the population is initialized, the objective function value is computed for each candidate solution $\mathbf{x} \in P^{(t)}$, using the function $Compute_Fitness(P^{(t)})$ (step 5 in Algorithm 1).

2.2 Cloning operator

The cloning operator (step 8 in Algorithm 1) clones each candidate solution dup times producing an intermediate population $P^{(clo)}$ of size $d \times dup$ and assigns to each clone a random age chosen in the range $[0, \tau_B]$. The age for a candidate solution determines its life time in the population: when a candidate solution reaches the maximum age (τ_B) it is discarded, i.e. *it dies*. This strategy reduces the premature convergence of the algorithm and keeps high diversity in the population. An improvement of the performances can be obtained choosing the age of each clone into the range $[0, \frac{2}{3}\tau_B]$, as showed in Sect. 4. The cloning operator, coupled with the hypermutation operator, performs a local search around the cloned solutions. The introduction of blind mutations can produce individuals with higher affinities (higher objective function values) which will be then selected forming the improved mature progenies.

2.3 Hypermutation operator

The hypermutation operator (step 9 in Algorithm 1) acts on each candidate solution of population $P^{(clo)}$. Although there are different ways of implementing this operator (see [11, 12]), in this research work we use an *inversely proportional* strategy where each candidate solution is subject to M mutations without explicitly using a mutation probability. The number of mutations M is determined by an inversely proportional law: the better is the objective function value of the candidate solution, the lower is the number of mutations performed. In this work we employ two different *potential mutations* to determine the number of mutations M :

$$\alpha = \frac{e^{-\hat{f}(\mathbf{x})}}{\rho}, \tag{4}$$

and

$$\alpha = e^{-\rho \hat{f}(\mathbf{x})}, \tag{5}$$

where α represents the *mutation rate*, ρ determines the shape of the mutation rates and $\hat{f}(\mathbf{x})$ the objective function value normalized in $[0, 1]$.

Thus the number of mutations M is given by

$$M = \lfloor (\alpha \times \ell) + 1 \rfloor, \tag{6}$$

where ℓ is the length of any candidate solution: (1) $\ell = kn$ for OPT-IMMALG01, with k the number of bits used to code each real variable and n the dimension of the function; whilst (2) $\ell = n$ for OPT-IMMALG, that is the dimension of the problem. By this equation at least one mutation is guaranteed on any candidate solution; this happens exactly when the solution represented by a candidate solution is very close to the optimal one into the space of the solutions. Once the objective function is normalized into the range $[0, 1]$, the best solutions are those whose values are closer to 1, whilst the worst ones are closer to 0. During normalization of the objective function value we use the best current objective function value decreased by a *user-defined threshold* θ , rather than the global optima. This way we do not use any a priori knowledge about the problem. In OPT-IMMALG01, the hypermutation operator is based on the classical *bit-flip mutation without redundancy*: in any x candidate solution the operator randomly chooses x_i , and inverts its value (from 0 to 1 or from 1 to 0). Since M mutations are performed in any candidate solution the x_i are randomly chose without repetition. In OPT-IMMALG instead the mutation operator randomly chooses two indexes $1 \leq i, j \leq \ell$, such that $i \neq j$, and replaces $x_i^{(t)}$ with a new value in according to the following rule:

$$x_i^{(t+1)} = \left((1 - \beta) x_i^{(t)} \right) + \left(\beta x_j^{(t)} \right) \tag{7}$$

where $\beta \in [0, 1]$ is a random number generated with uniform distribution.

Immunological Algorithm ($d, dup, \rho, \tau_B, T_{max}$);

$t \leftarrow 0$;

$FFE \leftarrow 0$;

$N_c \leftarrow d \cdot dup$;

$P^{(t)} \leftarrow \text{Initialize_Population}(d)$;

Compute_Fitness($P^{(t)}$);

$FFE \leftarrow FFE + d$;

while $FFE < T_{max}$ **do**

$P^{(clo)} \leftarrow \text{Cloning}(P^{(t)}, dup)$;

$P^{(hyp)} \leftarrow \text{Hypermutation}(P^{(clo)}, \rho)$;

 Compute_Fitness($P^{(hyp)}$);

$FFE \leftarrow FFE + N_c$;

$(P_a^{(t)}, P_a^{(hyp)}) = \text{Aging}(P^{(t)}, P^{(hyp)}, \tau_B)$;

$P^{(t+1)} \leftarrow (\mu + \lambda)\text{-Selection}(P_a^{(t)}, P_a^{(hyp)})$;

$t \leftarrow t + 1$;

end

Algorithm 1: Pseudo-code of the Immunological Algorithm

2.4 Aging operator

The aging operator (step 12 in Algorithm 1) eliminates all old candidate solutions in the populations $P^{(t)}$ and $P^{(hyp)}$. The main goal of this operator is to produce high diversity in the current population and to avoid premature convergence. Each candidate solution is allowed to remain in the population for a fixed number of generations according to the parameter τ_B . Hence, τ_B indicates the maximum number of generations allowed; when a candidate solution is $\tau_B + 1$ old it is discarded from the current population independently from its objective function value. Such kind of operator is called *static aging operator*. The algorithm makes only one exception: when generating a new population the selection mechanism always keeps the best candidate solution, i.e. the solution with the best objective function value so far, even if older than τ_B . This variant is called *elitist aging operator*.

2.5 $(\mu + \lambda)$ -Selection operator

After performing the aging operator, the best candidate solutions that have survived the aging step are selected to generate the new population $P^{(t+1)}$, of d candidate solutions from the populations $P_a^{(t)}$ and $P_a^{(hyp)}$. If only $d_1 < d$ candidate solutions have survived then the $(\mu + \lambda)$ -Selection operator randomly selects $d - d_1$ candidate solutions among those “dead”, i.e. from the set

$$\left((P^{(t)} \setminus P_a^{(t)}) \sqcup (P^{(hyp)} \setminus P_a^{(hyp)}) \right).$$

The $(\mu + \lambda)$ -Selection operator, with $\mu = d$ and $\lambda = N_c$, reduces the offspring population of size $\lambda \geq \mu$, created by cloning and hypermutation operators, to a new parent population of size $\mu = d$. The selection operator identifies the d best elements from the offspring set and the old parent candidate solutions, thus guaranteeing monotonicity in the evolution dynamics.

Both algorithms terminate the execution when the fitness function evaluation (*FFE*), is great or equal to T_{max} , the maximum number of objective function evaluations.

3 Benchmarks and metrics

Before presenting the comparative performance analysis to the state-of-the-art (Sect. 4), we explore some of the feature of the two IAs described in this work. We first present the test functions and the experimental protocol used in our tests. We then explore the influence of different mutation schemes on the performance of the IA. Next we show the experimental tuning of some of the parameters of the algorithm. Finally the dynamics and learning capabilities of both algorithms are explored.

3.1 Test functions and experimental protocol

We have used a large benchmark of test functions belonging to different classes and with different features. Specifically we combined two benchmarks proposed respectively in Yao et al. [43] (23 functions showed in Table 1) and [40] (12 functions showed in Table 2). These functions can be divided in two categories of different complexity: *unimodal* and *multimodal* (with many and few local optima) functions. Although their complexity gets larger as the dimension space increase, optimizing unimodal functions is not a major issue, so for this kind of functions the convergence rate becomes the main interest. Moreover, we have used

Table 3 Number of the objective function evaluations (T_{max}) used for each test function of Table 1, as proposed in Yao et al. [43]

Function	T_{max}	Function	T_{max}	Function	T_{max}
f_1	150,000	f_9	500,000	f_{17}	10,000
f_2	200,000	f_{10}	150,000	f_{18}	10,000
f_3	500,000	f_{11}	200,000	f_{19}	10,000
f_4	500,000	f_{12}	150,000	f_{20}	20,000
f_5	2×10^6	f_{13}	150,000	f_{21}	10,000
f_6	150,000	f_{14}	10,000	f_{22}	10,000
f_7	300,000	f_{15}	400,000	f_{23}	10,000
f_8	900,000	f_{16}	10,000		

another set of functions taken from Cassioli et al. [5], which includes 8 functions with number of variables $n \in \{10, 20\}$. The main goal when applying optimization algorithms to these functions is to get a picture of their convergence speed. Multimodal functions are instead characterized by a rugged fitness landscape difficult to explore, so the quality of the result obtained by any optimization method is crucial since it reflects the ability of the algorithm to *escape* from local optima. This last category of functions represents the most difficult class of problems for many optimization algorithms. Using a very large benchmark is necessary in order to reduce biases and analyze the overall robustness of evolutionary algorithms [24]. Also we have tested our IAs using different dimensions: from small (1 variable) to very large values (5000 variables).

We use the same experimental protocol proposed in Yao et al. [43]: 50 independent runs were performed for each test function. For all runs we compute both the *mean value* of the best candidate solutions and the *standard deviation*. The dimension was fixed as follows: $n = 30$ for functions from f_1 , to f_{13} ; $n = 2$ for functions ($f_{14}, f_{16}, f_{17}, f_{18}$); $n = 4$ for functions ($f_{15}, f_{19}, f_{21}, \dots, f_{23}$); and $n = 6$ for function f_{20} . Finally, for these experiments we used the same stopping criteria, T_{max} value, proposed in Yao et al. [43] and shown in Table 3.

3.2 Influence of different mutation potentials

Two different potential mutations (Eqs. 4, 5) are used in OPT-IMMAG01 and OPT-IMMAG to determine the number of mutations M (Eq. 6). In this section we present a comparison of their relative performances. Table 4 shows for each function the *mean* of the best candidate solutions and the *standard deviation* for all runs (the best result is highlighted in boldface).

These results were obtained using the experimental protocol described previously in Sect. 3.1. Moreover, we fixed for OPT-IMMAG01 $d \in \{10, 20\}$, $dup = 2$, $\tau_B \in \{5, 10, 15, 20, 50\}$, while for OPT-IMMAG $d = 100$, $dup = 2$, $\tau_B = 15$. For both versions we used ρ in the set $\{50, 75, 100, 125, 150, 175, 200\}$ for the mutation rate 4, and ρ in the set $\{4, 5, 6, 7, 8, 9, 10, 11\}$ for mutation rate 5. After inspecting the table it is easy to conclude that the second potential mutation has in the overall a better performance for both versions of the algorithm. For OPT-IMMAG the improvements obtained using mutation rate 5 are more evident rather than OPT-IMMAG01. In fact for OPT-IMMAG01 the potential mutation 4 reaches better solutions in the second class, i.e. the ones with many local optima.

Table 4 Comparison of the results obtained by both versions, OPT-IMMALG01 and OPT-IMMALG, using the two potential mutations (Eqs. 4, 5)

	OPT- IMMALG01		OPT- IMMALG	
	$\alpha = \frac{e^{-\hat{f}(x)}}{\rho}$	$\alpha = e^{-\rho \hat{f}(x)}$	$\alpha = \frac{e^{-\hat{f}(x)}}{\rho}$	$\alpha = e^{-\rho \hat{f}(x)}$
f_1	1.7×10^{-8} 3.5×10^{-15}	9.23×10^{-12} 2.44×10^{-11}	4.663×10^{-19} 7.365×10^{-19}	0.0 0.0
f_2	7.1×10^{-8} 0.0	0.0 0.0	3.220×10^{-17} 1.945×10^{-17}	0.0 0.0
f_3	1.9×10^{-10} 2.63×10^{-10}	0.0 0.0	3.855 5.755	0.0 0.0
f_4	4.1×10^{-2} 5.3×10^{-2}	1.0×10^{-2} 5.3×10^{-3}	8.699×10^{-3} 3.922×10^{-2}	0.0 0.0
f_5	28.4 0.42	3.02 12.2	22.32 11.58	16.29 13.96
f_6	0.0 0.0	0.2 0.44	0.0 0.0	0.0 0.0
f_7	3.9×10^{-3} 1.3×10^{-3}	3.0×10^{-3} 1.2×10^{-3}	1.143×10^{-4} 1.411×10^{-4}	1.995×10^{-5} 2.348×10^{-5}
f_8	-12568.27 0.23	-12508.38 155.54	-12559.69 34.59	-12535.15 62.81
f_9	2.66 2.39	19.98 7.66	0.0 0.0	0.596 4.178
f_{10}	1.1×10^{-4} 3.1×10^{-5}	18.98 0.35	1.017×10^{-10} 5.307×10^{-11}	0.0 0.0
f_{11}	4.55×10^{-2} 4.46×10^{-2}	7.7×10^{-2} 8.63×10^{-2}	2.066×10^{-2} 5.482×10^{-2}	0.0 0.0
f_{12}	3.1×10^{-2} 5.7×10^{-2}	0.137 0.23	7.094×10^{-21} 5.621×10^{-21}	1.770×10^{-21} 8.774×10^{-24}
f_{13}	3.20 0.13	1.51 0.1	1.122×10^{-19} 2.328×10^{-19}	1.687×10^{-21} 5.370×10^{-24}
f_{14}	1.21 0.54	1.02 7.1×10^{-2}	0.999 7.680×10^{-3}	0.998 1.110×10^{-3}
f_{15}	7.7×10^{-3} 1.4×10^{-2}	7.1×10^{-4} 1.3×10^{-4}	3.27×10^{-4} 3.651×10^{-5}	3.2×10^{-4} 2.672×10^{-5}
f_{16}	-1.02 1.1×10^{-2}	-1.032 1.5×10^{-4}	-1.017 2.039×10^{-2}	-1.013 2.212×10^{-2}
f_{17}	0.450 0.21	0.398 2.0×10^{-4}	0.425 4.987×10^{-2}	0.423 3.217×10^{-2}
f_{18}	3.0 0.0	3.0 0.0	6.106 4.748	5.837 3.742
f_{19}	-3.72 1.1×10^{-2}	-3.72 1.1×10^{-4}	-3.72 8.416×10^{-3}	-3.72 7.846×10^{-3}
f_{20}	-3.31 5.9×10^{-3}	-3.31 7.4×10^{-2}	-3.293 3.022×10^{-2}	-3.292 3.097×10^{-2}
f_{21}	-5.36 2.20	-9.11 1.82	-10.153 (7.710×10^{-8})	-10.153 1.034×10^{-7}
f_{22}	-5.34 2.11	-9.86 1.88	-10.402 (1.842×10^{-6})	-10.402 1.082×10^{-5}
f_{23}	-6.03 2.66	-9.96 1.46	-10.536 7.694×10^{-7}	-10.536 1.165×10^{-5}

Each result indicates the *mean* of the best solutions (in the first line of each table entry), and the *standard deviation* (in the second line). The best result for each function is highlighted in boldface

3.3 The parameters of the immunological algorithms

In this section we present an analysis of the parameter settings that influence the performance of the algorithms. Independently from the experimental protocol, we fixed $d \in \{10, 20\}$, $dup = 2$, $\tau_B \in \{5, 10, 15, 20, 50\}$ for OPT-IMMAlg01 and $d = 100$, $dup = 2$, $\tau_B = 15$ for OPT-IMMAlg. These values were chosen after a deep investigation of the parameter tuning for each algorithm, not shown in this work (see [8,9,14] for details). In the first set of experiments the values for parameter ρ were fixed as follows: $\{50, 75, 100, 125, 150, 175, 200\}$ using mutation rate 4 and $\{4, 5, 6, 7, 8, 9, 10, 11\}$ for mutation rate 5. Since OPT-IMMAlg presents one more parameter θ than OPT-IMMAlg01 (see Sect. 2), we first analyzed the best tuning for θ . After several experiments (not shown in this work), the best value found was $\theta = 75\%$ for both potential mutations. Such setting allows both algorithms to perform better on 14 functions out of 23. These experiments were made on 50 independent runs. We have also tested OPT-IMMAlg using different ranges to randomly choose the age of each clone, and we have discovered that choosing the age in the range $[0, \frac{2}{3}\tau_B]$ improves its performance. For this new variant of OPT-IMMAlg we used only the potential mutation from Eq. 5, because it appears to be the best (as shown in Sect. 4). We will call this new version OPT-IMMAlg*. After several experiments, the best tuning for OPT-IMMAlg* was: $dup = 2$, $\tau_B = 10$, $\theta = 50\%$, and $d = 1000$ for all $n \geq 30$, $d = 100$ otherwise.

Next we explored the performances of parameter ρ when tackling functions with different dimension value, with the goal of finding the best setting for ρ for each dimension. Figure 1 shows the dynamics of the number of mutations for different dimensions and ρ values. Using this figure we have fixed ρ as follows: $\rho = 3.5$ for dimension $n = 30$; $\rho = 4.0$ for dimension $n = 50$; $\rho = 6.0$ for dimension $n = 100$; and $\rho = 7$ for dimension $n = 200$. Instead for dimension $n = 2$ and $n = 4$ (not showed in the figure) we found the best values to be $\rho = 0.8$ for $n = 2$, and $\rho = 1.5$ for $n = 4$.

Moreover, we considered functions with very large dimension: $n = 1000$ and $n = 5000$. In this case we have tuned $\rho = 9.0$ and $\rho = 11.5$ respectively (see Fig. 2). From the figure we

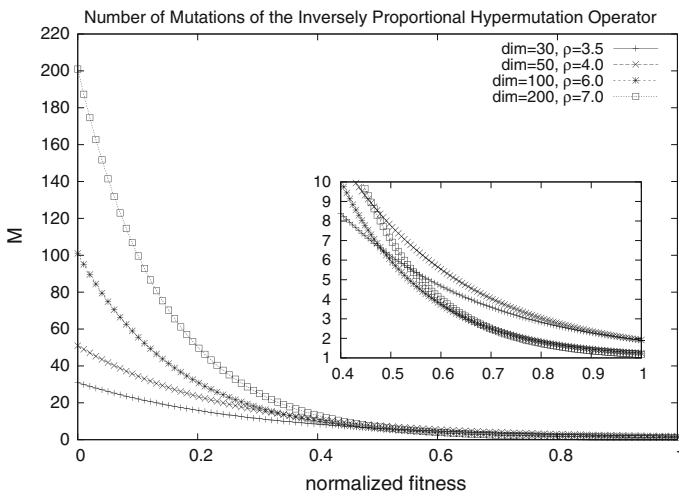


Fig. 1 Number of mutations M obtained on several dimensions

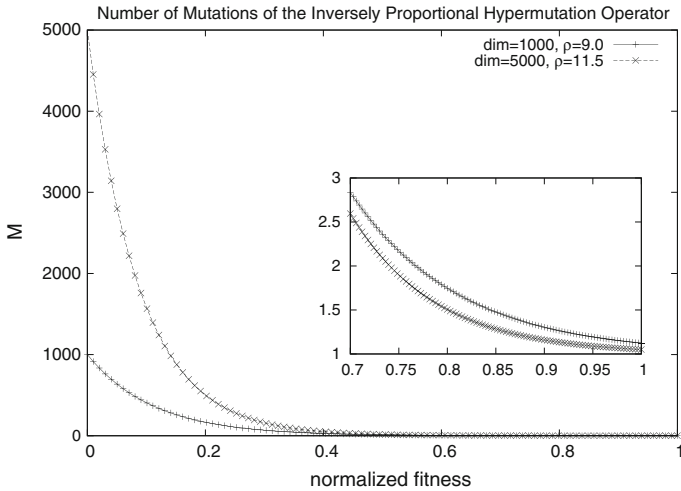


Fig. 2 Number of mutations M obtained for high dimension values

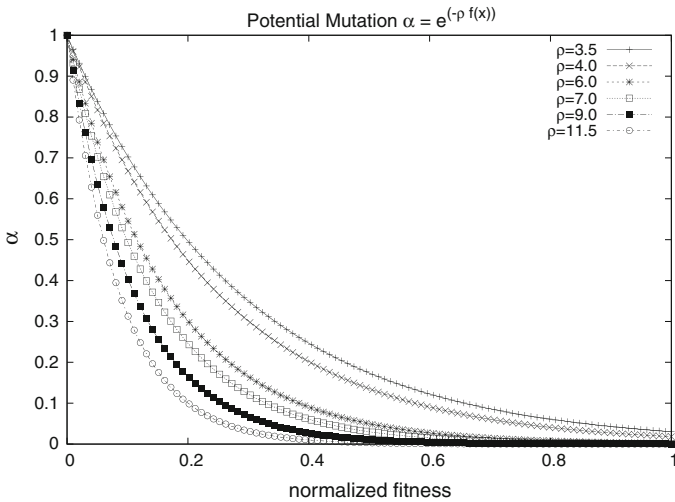


Fig. 3 Potential mutation α (Eq. 5) used by OPT- IMMALG*

can conclude that when the mutation rate is low the corresponding objective values improve, whereas high mutation rates correspond to bad objective function values (which agrees with the behaviour of the B cells in the natural immune systems). The inset plot shows a zoom of mutation rates in the range $[0.7, 1]$.

Finally, Fig. 3 instead shows the curves produced by α mutation potential using Eq. 5 at the different ρ values. Also in this figure it is possible to see an inversely proportional behaviour with respect to the normalized objective function, where higher α values correspond to worst solutions, whose normalized objective function value is closer to zero. Vice versa, the lower α values are obtained for good normalized objective function values (i.e. closer to one).

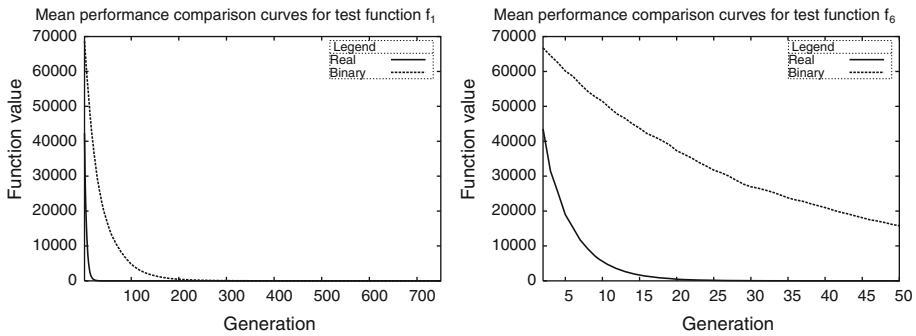


Fig. 4 Evolution curves of OPT-IMMAG01 and OPT-IMMAG algorithms on two unimodal functions f_1 (left plot) and f_6 (right plot)

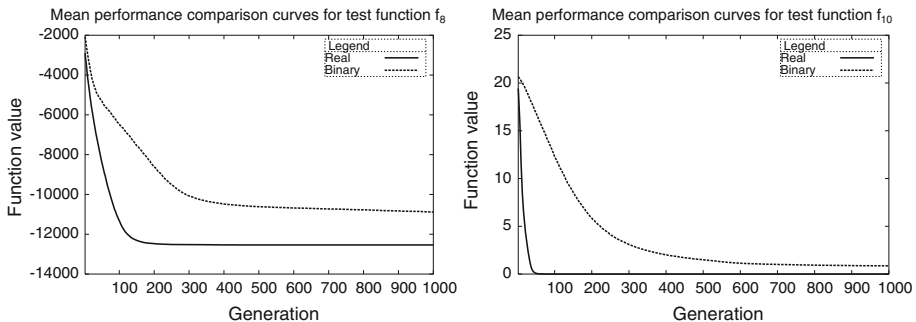


Fig. 5 Evolution curves of OPT-IMMAG01 and OPT-IMMAG algorithms on two multimodal functions f_8 (left plot) and f_{10} (right plot) with many local optima

3.4 The convergence and learning processes

Two important features that have an impact on the performance of any optimization algorithm are the *convergence speed* and the *learning ability*. In this section we examine the performance of the two versions of the IA according to these two properties. For this purpose we tested the IAs on two functions for each class from Table 1: f_1 and f_6 for the unimodal functions; f_8 and f_{10} for the multimodal functions with many local optima, and f_{18} and f_{21} for the multimodal functions with a few local optima. All the results are averaged over 50 independent runs.

Figures 4, 5 and 6 show the evolution curves produced by OPT-IMMAG01 (labelled as binary) and OPT-IMMAG (labelled as real), on the full set of test functions. Inspecting the results of the plots it is clear that OPT-IMMAG presents faster and better quality convergence than OPT-IMMAG01 in all instances.

The analysis of the learning process of the algorithm is performed using an entropic function, the *Information Gain*. This function measures the quantity of information the system discovers during the learning phase [10, 13]. For this purpose we define the candidate solutions distribution function $f_m^{(t)}$ as the ratio between the number, B_m^t , of candidate solutions at time step t with objective function value m and the total number of candidate

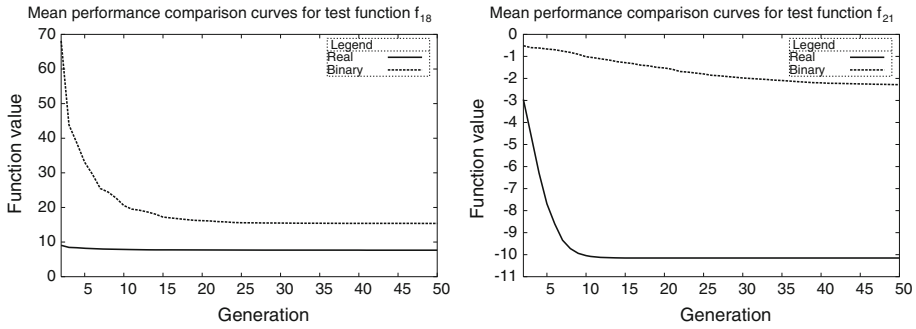


Fig. 6 Convergence process of OPT-IMMAG01 and OPT-IMMAG algorithms on two multimodal functions f_{18} (left plot) and f_{21} (right plot) with few local optima

solutions:

$$f_m^{(t)} = \frac{B_m^t}{\sum_{m=0}^h B_m^t} = \frac{B_m^t}{d}. \tag{8}$$

It follows that the information gain $K(t, t_0)$ and entropy $E(t)$ can be defined as:

$$K(t, t_0) = \sum_m f_m^{(t)} \log \left(f_m^{(t)} / f_m^{(t_0)} \right) \tag{9}$$

$$E(t) = \sum_m f_m^{(t)} \log f_m^{(t)}. \tag{10}$$

The gain is the amount of information the system has already learnt from the given problem instance compared to the randomly generated initial population $P^{(t=0)}$ (the initial distribution). Once the learning process begins, the information gain increases monotonically until it reaches a final steady state (see Fig. 7). This is consistent with the *maximum information-gain principle*: $\frac{dK}{dt} \geq 0$. Figure 7 shows the dynamics of the Information Gain of OPT-IMMAG* when applied to the functions f_5 , f_7 , and f_{10} . The algorithm quickly gains high information on functions f_7 and f_{10} and reaches a steady state at generation 20. However, more generations are required for function f_5 as the information gain starts growing only after generation 22. This behaviour is correlated to the different search space of function f_5 whose complexity is higher than the search spaces of function f_7 and f_{10} . This response is consistent with the experimental results: both OPT-IMMAG and OPT-IMMAG* algorithms require greater number of objective function evaluations to achieve good solutions (see the experimental protocol in Table 3). The plot in Fig. 8 shows the monotonous behaviour of the information gain for function f_5 , together with the standard deviation (inset plot); the standard deviation increases quickly (the spike in the inset plot) when the algorithm begins to learn information, than it rapidly decreases towards zero as the algorithm approaches the steady state of the information gain. The algorithm converges to the best solution in this temporal window. Thus, the highest point of information learned corresponds to the lowest value of uncertainty, i.e. standard deviation. Finally, Fig. 9 shows the curves of the information gain $K(t, t_0)$, and entropy $E(t)$ for OPT-IMMAG* of the function f_5 . The inset plot instead shows average objective function values versus best objective function value for the first 10 generations obtained on the same function f_5 ; the algorithm quickly decreases from solutions of the order 10^9 to solutions of the order $(10^1 - 1)$. The best solution for the results presented in Figs. 8 and 9 was 0.0 and the mean of the best solutions was 15.6

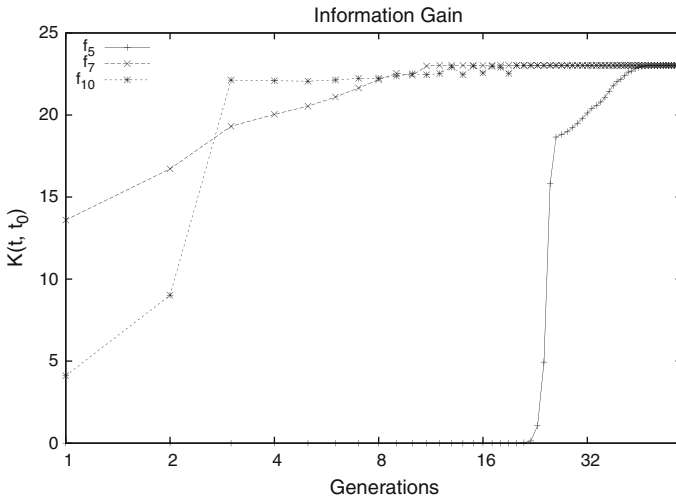


Fig. 7 Learning of the problem. Information Gain curves of OPT-IMMALG* algorithm on the functions f_5 , f_7 , and f_{10} . Each curve was obtained over 50 independent runs, with the following parameters: $d = 100$, $dup = 2$, $\tau_B = 15$, $\rho = 3.5$ and $T_{max} = 5 \times 10^5$

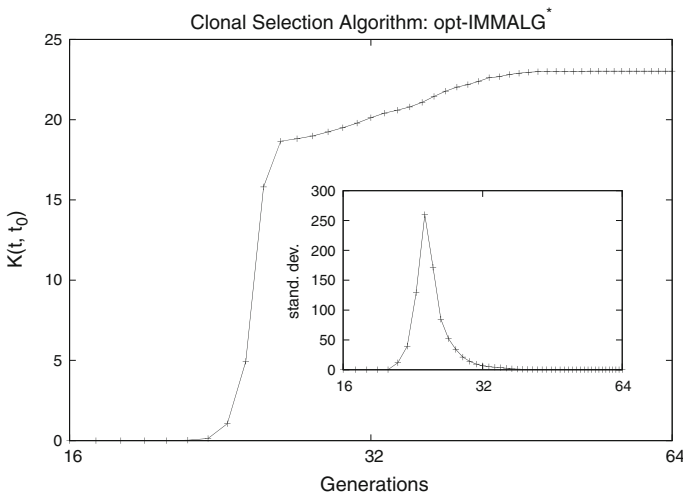


Fig. 8 Information Gain curves of OPT-IMMALG* algorithm on functions f_5 . The *inset* plot shows the standard deviation

(14.07 as standard deviation). The experiments were performed fixing parameters as follows: $d = 100$, $dup = 2$, $\tau_B = 15$, $\rho = 3.5$ and $T_{max} = 5 \times 10^5$.

3.5 Time-to-target analysis

Time-To-Target plots [2,20] are a way to characterize the running time of stochastic algorithms to solve a given combinatorial optimization problem. They display the probability that an algorithm will find a solution as good as a target within a given running time. Nowadays

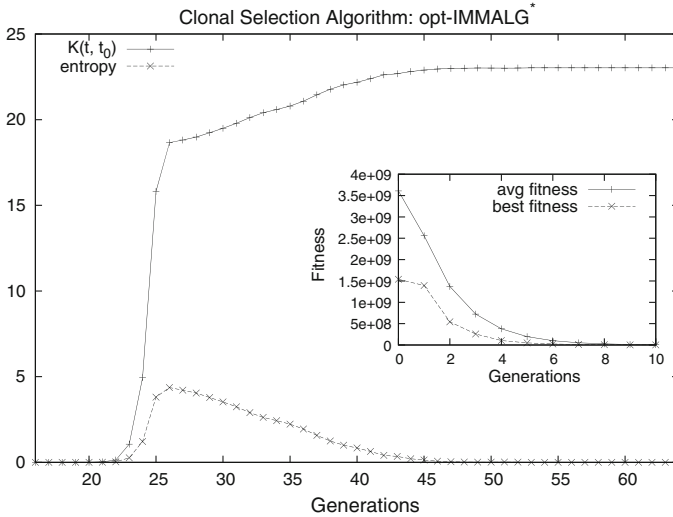


Fig. 9 Information Gain $K(t, t_0)$ and Entropy $E(t)$ curves of OPT-IMMALG* on the function f_5 . The inset plot shows the average objective function values versus the best objective function value for the first 10 generations. All curves are averaged over 50 independent runs with the following parameter setting: $d = 100$, $dup = 2$, $\tau_B = 15$, $\rho = 3.5$ and $T_{max} = 5 \times 10^5$

they are a standard graphical methodology for data analysis [6] to compare the empirical and theoretical distributions.¹

In Aiex et al. [1] is presented a Perl program (called `tttplots.pl`) to create time-to-target plots, as an useful tool for the comparisons of different stochastic algorithms or, in general, strategies for solving a given problem. Such program can be downloaded from <http://www2.research.att.com/~mgcr/tttplots/> By `tttplots.pl` two kinds of plots are produced: QQ -plot with superimposed variability information, and superimposed empirical and theoretical distributions.

Following the example presented in Aiex et al. [1], we ran OPT-IMMALG* on the first 13 functions of the Table 1 (for $n = 30$) where the obtained mean is equal to the optimal solution; that is when the success rate is 100%. For these experiments, of course, the termination criterion was properly changed, and that is until finding the target, i.e. the optimal solution. Moreover, since that larger is the number of runs closer is the empirical distribution to the theoretical distribution, we include in this work only the plots produced after 200 runs. For each of the 200 runs (as made for all the experiments and results presented in this article) the random number generator is initialized with a distinct seed, that is, each run is independent.

The Figs. 10, 11 and 12 show the convergence process produced by OPT-IMMALG* using `tttplots.pl` on the functions: $f_1, \dots, f_6, f_9, \dots, f_{13}$. The left plots show the comparisons among empirical and theoretical distributions, whilst the right plots display the QQ -plots with variability information. Inspecting the plots in the rightmost column is possible to see that the empirical and theoretical distributions are often the same, except for function f_6 which seems to be the easiest for OPT-IMMALG* among the given benchmark.

¹ For major details on this methodology see [1,2].

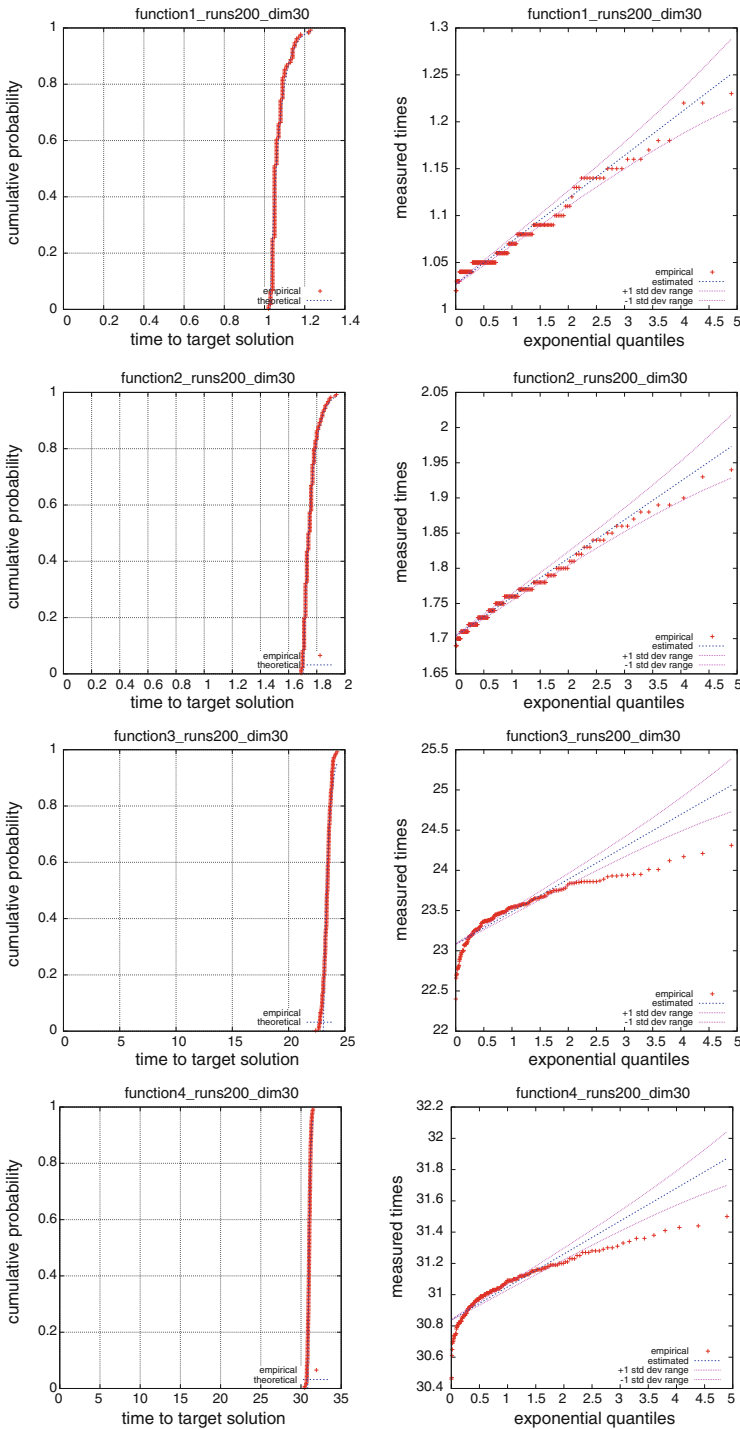


Fig. 10 Empirical versus theoretical distributions (*left plot*) and *QQ*-plots with variability information (*right plot*). The curves have been obtained for the functions: f_1 , f_2 , f_3 and f_4

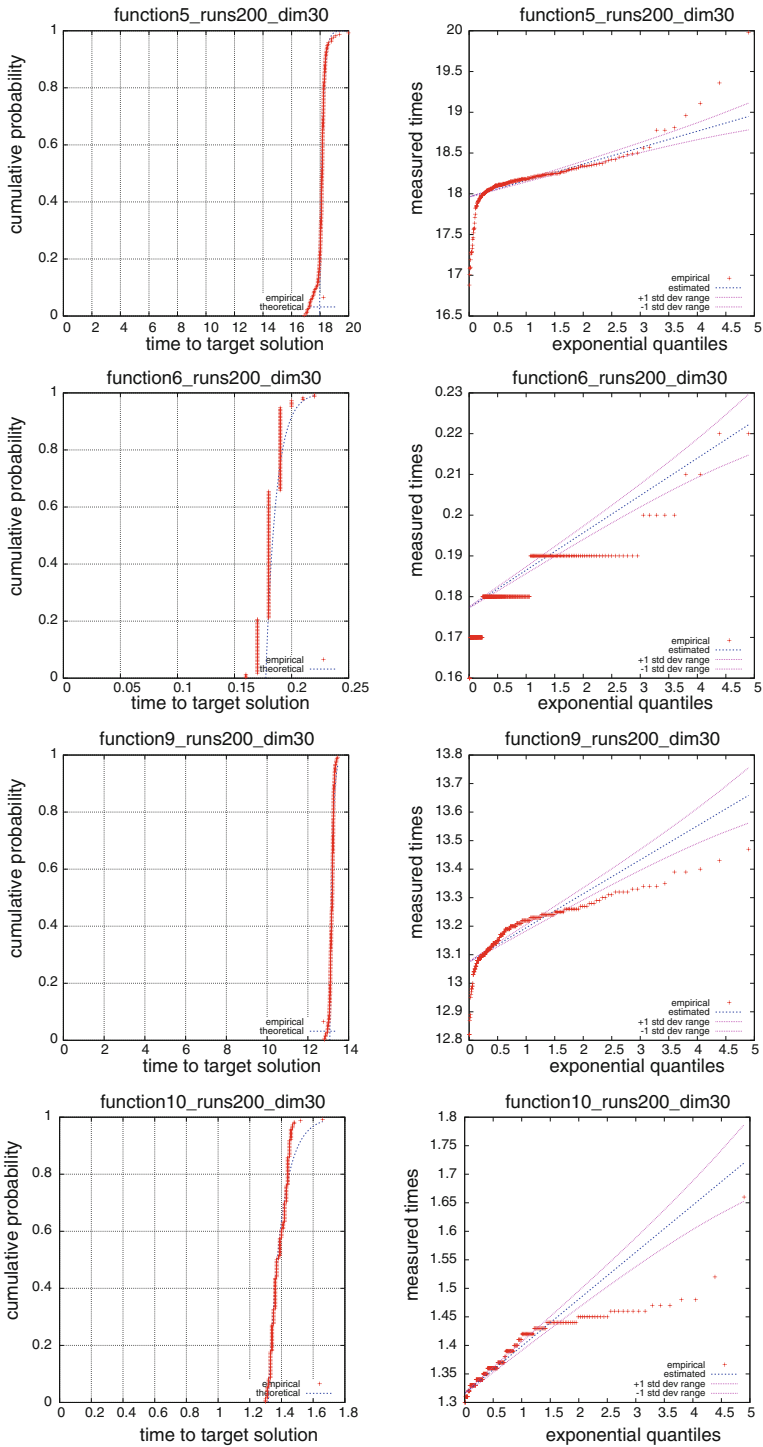


Fig. 11 Empirical versus theoretical distributions (*left plot*) and *QQ*-plots with variability information (*right plot*). The curves have been obtained for the functions: f_5 , f_6 , f_9 and f_{10}

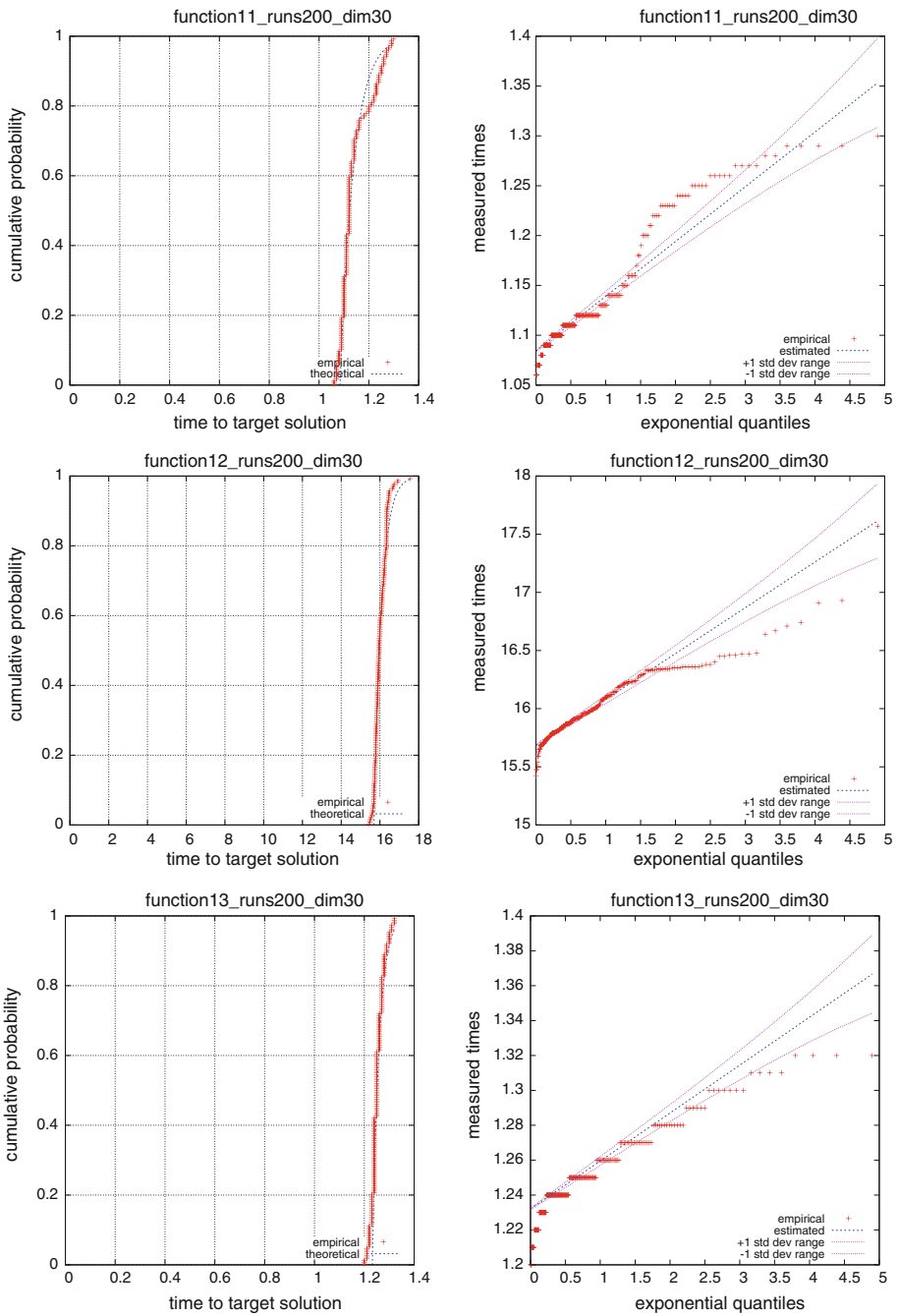


Fig. 12 Empirical versus theoretical distributions (*left plot*) and *QQ*-plots with variability information (*right plot*). The curves have been obtained for the functions: f_{11} , f_{12} and f_{13}

4 Comparisons and results

In this section we present an exhaustive comparative study between OPT-IMMALG01, OPT-IMMALG and OPT-IMMALG* with 39 *state-of-the-art optimization algorithms* from the literature. Such a large simulation protocol is required to fairly compare the IA to the current best nature-inspired, deterministic and hybrid optimization algorithms and to demonstrate its ability to outperform many of these techniques.

4.1 IA versus FEP and I-FEP

In the first experiment we compare OPT-IMMALG01 and OPT-IMMALG with FEP algorithm (*Fast Evolutionary Programming*), which was proposed in Yao et al. [43]. FEP is based on *Conventional Evolutionary Programming* (CEP [7]) and it uses a mutation operator based on Cauchy random numbers that helps the algorithm to escape from local optima. The results of this comparison are shown in Table 5. Both OPT-IMMALG01 and OPT-IMMALG outperform FEP in the majority of the instances. In particular OPT-IMMALG reaches the best values in 16 functions out of 23; 12 using the potential mutation of Eq. 5, and only 5 with Eq. 4. Comparing the two IA versions we can observe that OPT-IMMALG, using both potential mutations, outperforms OPT-IMMALG01 on 18 out of 23 functions. The best results are obtained using the second potential mutation (Eq. 5). It is important to mention that OPT-IMMALG outperforms OPT-IMMALG01 mainly on the multimodal functions. This result reflects its ability to *escape* from local optima. The analysis presented in Yao et al. [43] shows that Cauchy mutations perform better when the current search point is far away from the global optimum, whilst Gaussian mutations are better when the search points are in the neighbourhood of the global optimum. Based on these observations, the authors of [43] proposed an improved version of FEP. This algorithm, called I-FEP, is based on both Cauchy and Gaussian mutations, and it differs from FEP in the way offspring are created. Two new offspring are generated as follows: the first using Cauchy mutation and the second using Gaussian mutation; only the best offspring is chosen. Therefore we compared OPT-IMMALG and OPT-IMMALG01 also with I-FEP, and the results are reported in Table 6. We used functions f_1 , f_2 , f_{10} , f_{11} , f_{21} , f_{22} and f_{23} from Table 1, and for each function we show the *mean* of the best candidate solutions averaged over all runs (as proposed in Yao et al. [43]). Inspecting the results, we can infer that both versions of the IA obtain better performances (i.e. better solutions quality), than I-FEP on all functions.

Finally, since FEP is based on *Conventional Evolutionary Programming* (CEP), we present in Table 7 a comparison between the two versions of IA and CEP algorithm. CEP is based on three different mutation operators (as proposed in Chellapilla [7]): *Gaussian Mutation Operator* (GMO); *Cauchy Mutation Operator* (CMO); and *Mean Mutation Operator* (MMO). For this set of experiments, we used the same functions and the same experimental protocol proposed in Chellapilla [7]: i.e. $T_{max} = 2.5 \times 10^5$ for all functions, except for functions f_1 and f_{10} , where $T_{max} = 1.5 \times 10^5$ was used. The obtained results by OPT-IMMALG01 and OPT-IMMALG indicate that both IA versions outperform CEP on most of the instances. Moreover OPT-IMMALG shows an overall better performance compared to OPT-IMMALG01 and CEP.

Table 5 Comparison between OPT-IMMALG (real values representation), OPT-IMMALG01 (binary values representation) and FEP (Fast Evolutionary Algorithm), using the same experimental protocol proposed in Yao et al. [43]

	$\alpha = e^{-\rho \hat{f}(x)}$		FEP [43]	$\alpha = \frac{e^{-\hat{f}(x)}}{\rho}$	
	OPT-IMMALG	OPT-IMMALG01		OPT-IMMALG	OPT-IMMALG01
f_1	0.0 (0.0)	9.23×10^{-12} 2.44×10^{-11}	5.7×10^{-4} 1.3×10^{-4}	4.663×10^{-19} (7.365×10^{-19})	1.7×10^{-8} 3.5×10^{-15}
f_2	0.0 (0.0)	0.0 (0.0)	8.1×10^{-3} 7.7×10^{-4}	3.220×10^{-17} (1.945×10^{-17})	7.1×10^{-8} (0.0)
f_3	0.0 (0.0)	0.0 (0.0)	1.6×10^{-2} 1.4×10^{-2}	3.855 (5.755)	1.9×10^{-10} (2.63×10^{-10})
f_4	0.0 (0.0)	1.0×10^{-2} (5.3×10^{-3})	0.3 0.5	8.699×10^{-3} (3.922×10^{-2})	4.1×10^{-2} (5.3×10^{-2})
f_5	16.29 (13.96)	3.02 (12.2)	5.06 5.87	22.32 (11.58)	28.4 (0.42)
f_6	0.0 (0.0)	0.2 (0.44)	0.0 0.0	0.0 (0.0)	0.0 (0.0)
f_7	1.995×10^{-5} (2.348×10^{-5})	3.0×10^{-3} (1.2×10^{-3})	7.6×10^{-3} 2.6×10^{-3}	1.143×10^{-4} (1.411×10^{-4})	3.9×10^{-3} (1.3×10^{-3})
f_8	-12535.15 (62.81)	-12508.38 (155.54)	-12554.5 52.6	-12559.69 (34.59)	-12568.27 (0.23)
f_9	0.596 (4.178)	19.98 (7.66)	4.6×10^{-2} 1.2×10^{-2}	0.0 (0.0)	2.66 (2.39)
f_{10}	0.0 (0.0)	18.98 (0.35)	1.8×10^{-2} 2.1×10^{-3}	1.017×10^{-10} (5.307×10^{-11})	1.1×10^{-4} (3.1×10^{-5})
f_{11}	0.0 (0.0)	7.7×10^{-2} (8.63×10^{-2})	1.6×10^{-2} 2.2×10^{-2}	2.066×10^{-2} (5.482×10^{-2})	4.55×10^{-2} (4.46×10^{-2})
f_{12}	1.770×10^{-21} ($8,774 \times 10^{-24}$)	0.137 (0.23)	9.2×10^{-6} 3.6×10^{-6}	7.094×10^{-21} (5.621×10^{-21})	3.1×10^{-2} (5.7×10^{-2})
f_{13}	1.687×10^{-21} (5.370×10^{-24})	1.51 (0.10)	1.6×10^{-4} 7.3×10^{-5}	1.122×10^{-19} (2.328×10^{-19})	3.20 (0.13)
f_{14}	0.998 (1.110×10^{-3})	1.02 (7.1×10^{-2})	1.22 0.56	0.999 (7.680×10^{-3})	1.21 (0.54)
f_{15}	3.200×10^{-4} (2.672×10^{-5})	7.1×10^{-4} (1.3×10^{-4})	5.0×10^{-4} 3.2×10^{-4}	3.270×10^{-4} (3.651×10^{-5})	7.7×10^{-3} (1.4×10^{-2})
f_{16}	-1.013 (2.212×10^{-2})	-1.032 (1.5×10^{-4})	-1.031 4.9×10^{-7}	-1.017 (2.039×10^{-2})	-1.02 (1.1×10^{-2})
f_{17}	0.423 (3.217×10^{-2})	0.398 (2.0×10^{-4})	0.398 1.5×10^{-7}	0.425 (4.987×10^{-2})	0.450 (0.21)
f_{18}	5.837 (3.742)	3.0 (0.0)	3.02 0.11	6.106 (4.748)	3.0 (0.0)
f_{19}	-3.72 (7.846×10^{-3})	-3.72 (1.1×10^{-4})	-3.86 1.4×10^{-5}	-3.72 (8.416×10^{-3})	-3.72 (1.1×10^{-2})
f_{20}	-3.292 (3.097×10^{-2})	-3.31 (7.4×10^{-2})	-3.27 5.9×10^{-2}	-3.293 (3.022×10^{-2})	-3.31 (5.9×10^{-3})
f_{21}	-10.153 (1.034×10^{-7})	-9.11 (1.82)	-5.52 1.59	-10.153 (7.710×10^{-8})	-5.36 (2.20)
f_{22}	-10.402 (1.082×10^{-5})	-9.86 (1.88)	-5.52 2.12	-10.402 (1.842×10^{-6})	-5.34 (2.11)
f_{23}	-10.536 (1.165×10^{-5})	-9.96 (1.46)	-6.57 3.14	-10.536 (7.694×10^{-7})	-6.03 (2.66)

For OPT-IMMALG and OPT-IMMALG01 we show the results obtained using both potential mutations. For each algorithm we report the *mean* of the best candidate solutions on all runs (in the first line of each table entry) and the *standard deviation* (in the second line). The best results are highlighted in boldface

Table 6 Comparison between OPT-IMMALG (real values representation), OPT-IMMALG01 (binary values representation) and I-FEP (Improved Fast Evolutionary Algorithm), on functions $f_1, f_2, f_{10}, f_{11}, f_{21}, f_{22}$ and f_{23} from Table 1

	$\alpha = e^{-\rho \hat{f}(x)}$		I-FEP [43]	$\alpha = \frac{e^{-\hat{f}(x)}}{\rho}$	
	OPT-IMMALG	OPT-IMMALG01		OPT-IMMALG	OPT-IMMALG01
f_1	0.0	9.23×10^{-12}	4.16×10^{-5}	4.663×10^{-19}	1.7×10^{-8}
f_2	0.0	0.0	2.44×10^{-2}	3.220×10^{-17}	7.1×10^{-8}
f_{10}	0.0	18.98	4.83×10^{-3}	1.017×10^{-10}	1.1×10^{-4}
f_{11}	0.0	7.7×10^{-2}	4.54×10^{-2}	2.066×10^{-2}	4.55×10^{-2}
f_{21}	-10.153	-9.11	-6.46	-10.153	-5.36
f_{22}	-10.402	-9.86	-7.10	-10.402	-5.34
f_{23}	-10.536	-9.96	-7.80	-10.536	-6.03

For each algorithm we report the *mean* of the best candidate solutions averaged over all runs. The best results are highlighted in boldface

Table 7 Comparison between OPT-IMMALG (real values representation), OPT-IMMALG01 (binary values representation) and the version of CEP (Conventional Evolutionary Programming) based on three different mutation operators [7]: GMO (Gaussian Mutation Operator), CMO (Cauchy Mutation Operator), and MMO (Mean Mutation Operator)

	$\alpha = e^{-\rho \hat{f}(x)}$		CEP [7]			$\alpha = \frac{e^{-\hat{f}(x)}}{\rho}$	
	OPT-IMMALG	OPT-IMMALG01	GMO	CMO	MMO	OPT-IMMALG	OPT-IMMALG01
f_1	0.0	9.23×10^{-12}	3.09×10^{-7}	3.07×10^{-7}	9.81×10^{-7}	4.663×10^{-19}	1.7×10^{-8}
f_2	0.0	0.0	1.99×10^{-3}	5.87×10^{-3}	3.23×10^{-3}	3.220×10^{-17}	7.1×10^{-8}
f_3	0.0	0.0	17.60	5.78	11.80	3.855	1.9×10^{-10}
f_4	0.0	1.0×10^{-2}	5.18	0.66	1.88	8.699×10^{-3}	4.1×10^{-2}
f_5	16.29	3.02	86.70	114.0	63.8	22.32	28.4
f_7	1.995×10^{-5}	12.20	9.42	9.53	7.6×10^{-3}	1.143×10^{-4}	3.9×10^{-3}
f_9	0.596	19.98	120.0	4.73	9.52	0.0	2.66
f_{10}	0.0	18.98	9.10	1.3×10^{-3}	7.49×10^{-4}	1.017×10^{-10}	1.1×10^{-4}
f_{11}	0.0	7.7×10^{-2}	2.52×10^{-7}	2.2×10^{-6}	6.99×10^{-7}	2.066×10^{-2}	4.55×10^{-2}

For each algorithm the *mean* of the best candidate solutions on all runs is presented. The best results are highlighted in boldface

4.2 IA versus DIRECT, PSO and EO

Next we compared OPT-IMMALG01 and OPT-IMMALG with other two well-known biologically inspired algorithms: *Particle Swarm Optimization* (PSO) and *Evolutionary Optimization* (EO) [3]. For this other set of experiments we used functions f_1, f_5, f_9 and f_{11} as proposed in Angeline [3], and we fixed the maximum number of objective function evaluations $T_{max} = 2.5 \times 10^5$. The results presented in Table 8 strongly demonstrate the superior performance of OPT-IMMALG and OPT-IMMALG01 both in terms of convergence and quality of the solutions.

Table 9 presents the comparison between both versions of the IA and DIRECT [21, 28], a deterministic global search algorithm for bound constrained optimization based on Lipschitz constant estimation. Since, the results by DIRECT are not available for all functions of Table 1, we used only a subset of such functions $\{f_5, f_7, f_8, f_{12}, \dots, f_{23}\}$. The

Table 8 Comparison between OPT-IMMALG (real values representation), OPT-IMMALG01 (binary values representation), PSO (particle swarm optimization), and EO (Evolutionary Optimization) [3]

	$\alpha = e^{-\rho \hat{f}(x)}$		PSO [3]	EO [3]	$\alpha = \frac{e^{-\hat{f}(x)}}{\rho}$	
	OPT-IMMALG	OPT-IMMALG01			OPT-IMMALG	OPT-IMMALG01
f_1	0.0 (0.0)	9.23×10^{-12} 2.44×10^{-11}	11.75 1.3208	9.8808 0.9444	4.663×10^{-19} (7.365×10^{-19})	1.7×10^{-8} 3.5×10^{-15}
f_5	16.29 (13.96)	3.02 (12.2)	1911.598 374.2935	1610.39 293.5783	22.32 (11.58)	28.4 (0.42)
f_9	0.596 (4.178)	19.98 (7.66)	47.1354 1.8782	46.4689 2.4545	0.0 (0.0)	2.66 (2.39)
f_{11}	0.0 (0.0)	7.7×10^{-2} (8.63×10^{-2})	0.4498 0.0566	0.4033 0.0436	2.066×10^{-2} (5.482×10^{-2})	4.55×10^{-2} (4.46×10^{-2})

For each algorithm we report the *mean* of the best candidate solutions on all runs (in the first line of each table entry) and the *standard deviation* (in the second line). The best results are highlighted in boldface

Table 9 Comparison between OPT-IMMALG (real values representation), OPT-IMMALG01 (binary values representation) and DIRECT, a deterministic global search algorithm for bound constrained optimization based on Lipschitz constant estimation [21,28]

	$\alpha = e^{-\rho \hat{f}(x)}$		DIRECT [21,28]	$\alpha = \frac{e^{-\hat{f}(x)}}{\rho}$	
	OPT-IMMALG	OPT-IMMALG01		OPT-IMMALG	OPT-IMMALG01
f_5	16.29	3.02	27.89	22.32	28.4
f_7	1.995×10^{-5}	3.0×10^{-3}	8.9×10^{-3}	1.143×10^{-4}	3.9×10^{-3}
f_8	-12535.15	-12508.38	-4093.0	-12559.69	-12568.27
f_{12}	1.770×10^{-21}	0.137	0.03	7.094×10^{-21}	3.1×10^{-2}
f_{13}	1.687×10^{-21}	1.51	0.96	1.122×10^{-19}	3.20
f_{14}	0.998	1.02	1.0	0.999	1.21
f_{15}	3.2×10^{-4}	7.1×10^{-4}	1.2×10^{-3}	3.27×10^{-4}	7.7×10^{-3}
f_{16}	-1.013	-1.032	-1.031	-1.017	-1.02
f_{17}	0.423	0.398	0.398	0.425	0.450
f_{18}	5.837	3.0	3.01	6.106	3.0
f_{19}	-3.72	-3.72	-3.86	-3.72	-3.72
f_{20}	-3.292	-3.31	-3.30	-3.293	-3.31
f_{21}	-10.153	-9.11	-6.84	-10.153	-5.36
f_{22}	-10.402	-9.86	-7.09	-10.402	-5.34
f_{23}	-10.536	-9.96	-7.22	-10.536	-6.03

For each algorithm we report the *mean* of the best candidate solutions averaged over all runs. The best results are highlighted in boldface

reason why some functions could not be tested with DIRECT is that the optimum for these functions lays on the centre of the variable bounds and that is the point from which DIRECT starts its search. For these tests we used the same values for T_{max} as showed in Table 3 (Sect. 3.1).

By inspecting the results in the table we can claim that, except for function f_{19} , both OPT-IMMALG and OPT-IMMALG01 show again superior performance, in particular in the presence of rugged landscapes (multimodal functions).

4.3 IA versus CLONALG and BCA

We have compared OPT-IMMIALG01 and OPT-IMMIALG with two well-known immunological inspired algorithms, both based on the clonal selection principle: CLONALG [19] and BCA [40]. Two populations characterize CLONALG: a population of antigens Ag and a population of antibodies Ab . The individual antibody, Ab , and antigen, Ag , are represented by string attributes $m = m_L, \dots, m_1$, that is, a point in a L -dimensional shape space S , with $m \in S^L$. Two different strategies were adopted by CLONALG, labelled as CLONALG₁ and CLONALG₂ [9], based on different selection schemes: in CLONALG₁ each Ab at time step t will be replaced for the new generation (time step $t + 1$) with the best mutated clone; whilst, in CLONALG₂ the new population for generation $t + 1$ will be produced by the n best Ab 's of the mutated clones at time step t (n is the population size). Both schemes of CLONALG are based on the same potential mutation, produced by both Eqs. 4 and 5. Also for these experiments we have used the same values of T_{max} showed in Table 3.

Table 10 presents the comparative analysis between both versions of the IA and CLONALG [19]: OPT-IMMIALG01, OPT-IMMIALG, CLONALG₁, and CLONALG₂. The potential mutation of Eq. 4 was used for all four algorithms. The table shows the *mean* of the best candidate solutions on all runs and the *standard deviation*. All the results presented for the two versions of CLONALG were previously reported in Cutello et al. [9]. The results indicate that OPT-IMMIALG outperforms both versions of CLONALG on all classes of functions, except for functions f_{11} , f_{16} and f_{17} . If we compare the algorithms only on the multimodal functions with many local optima ($f_8 - f_{13}$), we can claim that OPT-IMMIALG is capable of reaching the best solutions more easily than the other clonal selection algorithm CLONALG. Table 11 presents the same comparison between the IAs and CLONALG but this time using the potential mutation of Eq. 5. The results show that both OPT-IMMIALG and OPT-IMMIALG01 again outperform both versions of CLONALG, in particular in the unimodal and multimodal (with many local optima) classes.

So far the experimental results have demonstrated OPT-IMMIALG being the superior implementation of the two IAs, so we next compare only this version with another immunological inspired optimization algorithm, BCA [40], and a Hybrid Genetic Algorithm, HGA. For this comparison we used the functions listed in Table 2 and we set the T_{max} value as proposed in Timmis and Kelsey [40]. 50 independent runs were performed. Table 12 compares these three algorithms. OPT-IMMIALG outperforms both BCA and HGA on 8 out of 12 test functions. In particular the results for function g_7 , g_8 , g_{11} , and g_{12} are significant.

4.4 IA versus PSO, SEA, and RCMA

Using a different experimental protocol, we have compared OPT-IMMIALG, using both potential mutations (Eqs. 4, 5), with other evolutionary algorithms proposed in Versterstrøm and Thomsen [42]: *Particle Swarm Optimization* (PSO) and *Simple Evolutionary Algorithm* (SEA). In addition to the classical PSO, the authors in Versterstrøm and Thomsen [42] proposed the *attractive and repulsive PSO* (arPSO), which uses a modified scheme for PSO to avoid premature convergence. We performed the comparisons on all functions from Table 1, except for the functions f_{19} and f_{20} according to Versterstrøm and Thomsen [42]. For each experiment, the maximum number of objective function evaluations (T_{max}) was fixed to 5×10^5 for dimensions ≤ 30 and we performed 30 independent runs for each instance. For functions $f_1 - f_{13}$ the comparison was performed using 100 dimensions. In this case, T_{max}

Table 10 Comparison between OPT-IMMALG (real values representation), OPT-IMMALG01 (binary values representation) and the two versions of CLONALG [9, 19], using potential mutation 4 ($\alpha = \frac{e^{-\hat{f}(x)}}{\rho}$)

	OPT-IMMALG	OPT-IMMALG01	CLONALG ₁ [9, 19]	CLONALG ₂ [9, 19]
f_1	4.663 × 10⁻¹⁹ (7.365 × 10 ⁻¹⁹)	1.7 × 10 ⁻⁸ (3.5 × 10 ⁻¹⁵)	3.7 × 10 ⁻³ (2.6 × 10 ⁻³)	5.5 × 10 ⁻⁴ (2.4 × 10 ⁻⁴)
f_2	3.220 × 10⁻¹⁷ (1.945 × 10 ⁻¹⁷)	7.1 × 10 ⁻⁸ (0.0)	2.9 × 10 ⁻³ (6.6 × 10 ⁻⁴)	2.7 × 10 ⁻³ (7.1 × 10 ⁻⁴)
f_3	3.855 (5.755)	1.9 × 10⁻¹⁰ (2.63 × 10 ⁻¹⁰)	1.5 × 10 ⁺⁴ (1.8 × 10 ⁺³)	5.9 × 10 ⁺³ (1.8 × 10 ⁺³)
f_4	8.699 × 10⁻³ (3.922 × 10 ⁻²)	4.1 × 10 ⁻² (5.3 × 10 ⁻²)	4.91 (1.11)	8.7 × 10 ⁻³ (2.1 × 10 ⁻³)
f_5	22.32 (11.58)	28.4 (0.42)	27.6 (1.034)	2.35 × 10 ⁺² (4.4 × 10 ⁺²)
f_6	0.0 (0.0)	0.0 (0.0)	2.0 × 10 ⁻² (1.4 × 10 ⁻¹)	0.0 (0.0)
f_7	1.143 × 10⁻⁴ (1.411 × 10 ⁻⁴)	3.9 × 10 ⁻³ (1.3 × 10 ⁻³)	7.8 × 10 ⁻² (1.9 × 10 ⁻²)	5.3 × 10 ⁻³ (1.4 × 10 ⁻³)
f_8	-12559.69 (34.59)	-12568.27 (0.23)	-11044.69 (186.73)	-12533.86 (43.08)
f_9	0.0 (0.0)	2.66 (2.39)	37.56 (4.88)	22.41 (6.70)
f_{10}	1.017 × 10⁻¹⁰ (5.307 × 10 ⁻¹¹)	1.1 × 10 ⁻⁴ (3.1 × 10 ⁻⁵)	1.57 (3.9 × 10 ⁻¹)	1.2 × 10 ⁻¹ (4.1 × 10 ⁻¹)
f_{11}	2.066 × 10 ⁻² (5.482 × 10 ⁻²)	4.55 × 10 ⁻² (4.46 × 10 ⁻²)	1.7 × 10⁻² (1.9 × 10 ⁻²)	4.6 × 10 ⁻² (7.0 × 10 ⁻²)
f_{12}	7.094 × 10⁻²¹ (5.621 × 10 ⁻²¹)	3.1 × 10 ⁻² (5.7 × 10 ⁻²)	0.336 (9.4 × 10 ⁻²)	0.573 (2.6 × 10 ⁻¹)
f_{13}	1.122 × 10⁻¹⁹ (2.328 × 10 ⁻¹⁹)	3.20 (0.13)	1.39 (1.8 × 10 ⁻¹)	1.69 (2.4 × 10 ⁻¹)
f_{14}	0.999 (7.680 × 10 ⁻³)	1.21 (0.54)	1.0021 (2.8 × 10 ⁻²)	2.42 (2.60)
f_{15}	3.270 × 10⁻⁴ (3.651 × 10 ⁻⁵)	7.7 × 10 ⁻³ (1.4 × 10 ⁻²)	1.5 × 10 ⁻³ (7.8 × 10 ⁻⁴)	7.2 × 10 ⁻³ (8.1 × 10 ⁻³)
f_{16}	-1.017 (2.039 × 10 ⁻²)	-1.02 (1.1 × 10 ⁻²)	-1.0314 (5.7 × 10 ⁻⁴)	-1.0210 (1.9 × 10 ⁻²)
f_{17}	0.425 (4.987 × 10 ⁻²)	0.450 (0.21)	0.399 (2.0 × 10 ⁻³)	0.422 (2.7 × 10 ⁻²)
f_{18}	6.106 (4.748)	3.0 (0.0)	3.0 (1.3 × 10 ⁻⁵)	3.46 (3.28)
f_{19}	-3.72 (8.416 × 10 ⁻³)	-3.72 (1.1 × 10 ⁻²)	-3.71 (1.5 × 10 ⁻²)	-3.68 (6.9 × 10 ⁻²)
f_{20}	-3.293 (3.022 × 10 ⁻²)	-3.31 (5.9 × 10 ⁻³)	-3.23 (5.9 × 10 ⁻²)	-3.18 (1.2 × 10 ⁻¹)
f_{21}	-10.153 (7.710 × 10 ⁻⁸)	-5.36 (2.20)	-5.92 (1.77)	-3.98 (2.73)
f_{22}	-10.402 (1.842 × 10 ⁻⁶)	-5.34 (2.11)	-5.90 (2.09)	-4.66 (2.55)
f_{23}	-10.536 (7.694 × 10 ⁻⁷)	-6.03 (2.66)	-5.98 (1.98)	-4.38 (2.66)

Each result report the *mean* of the best candidate solutions on all runs (in the first line of each table entry), and the *standard deviation* (in the second line). The best results are highlighted in boldface

Table 11 Comparison between OPT-IMMALG (real values representation), OPT-IMMALG01 (binary values representation) and the two versions of CLONALG [9, 19], using Each result indicates the *mean* of the best candidate solutions on all runs (in the first line of each table entry), and the *standard deviation* (in the second line)

	OPT-IMMALG	OPT-IMMALG01	CLONALG ₁ [9, 19]	CLONALG ₂ [9, 19]
f_1	0.0 (0.0)	9.23×10^{-12} (2.44×10^{-11})	9.6×10^{-4} (1.6×10^{-3})	3.2×10^{-6} (1.5×10^{-6})
f_2	0.0 (0.0)	0.0 (0.0)	7.7×10^{-5} (2.5×10^{-5})	1.2×10^{-4} (2.1×10^{-5})
f_3	0.0 (0.0)	0.0 (0.0)	2.2×10^4 (1.3×10^{-4})	$2.4 \times 10^{+4}$ ($5.7 \times 10^{+3}$)
f_4	0.0 (0.0)	1.0×10^{-2} (5.3×10^{-3})	9.44 (1.98)	5.9×10^{-4} (3.5×10^{-4})
f_5	16.29 (13.96)	3.02 (12.2)	31.07 (13.48)	$4.67 \times 10^{+2}$ ($6.3 \times 10^{+2}$)
f_6	0.0 (0.0)	0.2 (0.44)	0.52 (0.49)	0.0 (0.0)
f_7	1.995×10^{-5} (2.348×10^{-5})	3.0×10^{-3} (1.2×10^{-3})	1.3×10^{-1} (3.5×10^{-2})	4.6×10^{-3} (1.6×10^{-3})
f_8	-12535.15 (62.81)	-12508.38 (155.54)	-11099.56 (112.05)	-1228.39 (41.08)
f_9	0.596 (4.178)	19.98 (7.66)	42.93 (3.05)	21.75 (5.03)
f_{10}	0.0 (0.0)	18.98 (0.35)	18.96 (2.2×10^{-1})	19.30 (1.9×10^{-1})
f_{11}	0.0 (0.0)	7.7×10^{-2} (8.63×10^{-2})	3.6×10^{-2} (3.5×10^{-2})	9.4×10^{-2} (1.4×10^{-1})
f_{12}	1.770×10^{-21} (8.774×10^{-24})	0.137 (0.23)	0.632 (2.2×10^{-1})	0.738 (5.3×10^{-1})
f_{13}	1.687×10^{-21} (5.370×10^{-24})	1.51 (0.10)	1.83 (2.7×10^{-1})	1.84 (2.7×10^{-1})
f_{14}	0.998 (1.110×10^{-3})	1.02 (7.1×10^{-2})	1.0062 (4.0×10^{-2})	1.45 (0.95)
f_{15}	3.2×10^{-4} (2.672×10^{-5})	7.1×10^{-4} (1.3×10^{-4})	1.4×10^{-3} (5.4×10^{-4})	8.3×10^{-3} (8.5×10^{-3})
f_{16}	-1.013 (2.212×10^{-2})	-1.032 (1.5×10^{-4})	-1.0315 (1.8×10^{-4})	-1.0202 (1.8×10^{-2})
f_{17}	0.423 (3.217×10^{-2})	0.398 (2.0×10^{-4})	0.401 (8.8×10^{-3})	0.462 (2.0×10^{-1})
f_{18}	5.837 (3.742)	3.0 (0.0)	3.0 (1.3×10^{-7})	3.54 (3.78)
f_{19}	-3.72 (7.846×10^{-3})	-3.72 (1.1×10^{-4})	-3.71 (1.1×10^{-2})	-3.67 (6.6×10^{-2})
f_{20}	-3.292 (3.097×10^{-2})	-3.31 (7.4×10^{-2})	-3.30 (1.0×10^{-2})	-3.21 (8.6×10^{-2})
f_{21}	-10.153 (1.034×10^{-7})	-9.11 (1.82)	-7.59 (1.89)	-5.21 (1.78)
f_{22}	-10.402 (1.082×10^{-5})	-9.86 (1.88)	-8.41 (1.4)	-7.31 (2.67)
f_{23}	-10.536 (1.165×10^{-5})	-9.96 (1.46)	-8.48 (1.51)	-7.12 (2.48)

The best results are highlighted in boldface

Table 12 Comparison between OPT-IMMALG, BCA and HGA [40]

	OPT-IMMALG using $\alpha = \frac{e^{-\hat{f}(x)}}{\rho}$	OPT-IMMALG using $\alpha = e^{-\rho \hat{f}(x)}$	BCA [40]	HGA [40]
g_1	$-1.12 \pm 1.17 \times 10^{-3}$	$-1.12 \pm 1.62 \times 10^{-3}$	-1.08	-1.12
g_2	$-1.03 \pm 8.82 \times 10^{-4}$	$-1.03 \pm 7.129 \times 10^{-4}$	-1.03	-0.99
g_3	$-12.03 \pm 8.196 \times 10^{-4}$	$-12.03 \pm 9.28 \times 10^{-4}$	-12.03	-12.03
g_4	$0.3984 \pm 6.73 \times 10^{-4}$	$0.3985 \pm 8.859 \times 10^{-4}$	0.40	0.40
g_5	-178.51 ± 11.49	-178.88 ± 9.83	-186.73	-186.73
g_6	-179.27 ± 11.498	-179.12 ± 10.02	-186.73	-186.73
g_7	-2.529 ± 0.2026	-2.571 ± 0.253	0.92	0.92
g_8	$1.314 \times 10^{-12} \pm 4.668 \times 10^{-12}$	$1.314 \times 10^{-12} \pm 4.668 \times 10^{-12}$	1.0	1.0
g_9	$-3.51 \pm 1.464 \times 10^{-3}$	$-0.351 \pm 1.62 \times 10^{-3}$	-0.91	-0.99
g_{10}	$-186.67 \pm 8.17 \times 10^{-2}$	-186.65 ± 0.1158	-186.73	-186
g_{11}	$3.81 \times 10^{-5} \pm 5.58 \times 10^{-15}$	$3.81 \times 10^{-5} \pm 6.98 \times 10^{-14}$	0.04	0.04
g_{12}	0.0 ± 0.0	0.0 ± 0.0	1	1

The functions used are listed in table 2. For OPT-IMMALG we show the *mean* of the best candidate solutions on all runs, and *standard deviation* values (*mean* \pm *sd*). The best results are highlighted in boldface

was set to 5×10^6 . For each instance we report the *mean* of the best solutions averaged over all runs and the *standard deviation*.

Table 13 presents the comparison between OPT-IMMALG, PSO (particle swarm optimization), arPSO (attractive and repulsive particle swarm optimization) and SEA (simple evolutionary algorithm), obtained using $n = 30$ dimensions. The results indicate a better performance of OPT-IMMALG than the above-cited algorithms, outperforming them in the majority of the functions. In Table 14 instead we show the same comparisons but with $n = 100$ dimensions. Again, OPT-IMMALG outperforms SEA, PSO and its modification on all functions. Therefore, from these experiments, we can claim that OPT-IMMALG is capable of tackling functions with high dimension better than these evolutionary algorithms.

Recent developments in the evolutionary algorithms field, have shown that in order to tackle complex search spaces, pure genetic algorithms (GA) need to use local search operators and specialized crossover [25]. Such kind of algorithms are called Memetic Algorithms (MA) [26]. Table 15 shows the comparisons of OPT-IMMALG with several real coded memetic algorithms (RCMA) [30,32]: CHC algorithm, Generalized Generation Gap($G3 - 1$), hybrid steady state RCMA (SW-100), Family Competition (FC) and RCMA with crossover Hill Climbing (RCMA-XHC). The detailed descriptions for these algorithms can be found in Lozano et al. [30], whilst the reported results were extracted from Noman and Iba [32]. Such experiments were performed using $n = 25$ dimensions, $T_{max} = 10^5$ maximum number of objective function evaluations and 30 independent runs. For this comparison we used the potential mutation from Eq. 5. As proposed in Noman and Iba [32], the tests were performed only on functions f_5, f_9 and f_{11} . By looking at the results reported in the table it is clear that OPT-IMMALG outperforms all RCMA. Although RMCA-XHC obtains the best result for the last function f_5 , the proposed IA presents notably better results than the others RMCA.

4.5 OPT- IMMALG versus OPT- IMMALG*

The analysis of the experiments reported so far have shown that OPT-IMMALG, using the second potential mutation (5), performs better in terms of solution’s quality and ability to escape from a local optima. While performing the parameter tuning of the algorithm we

Table 13 Comparison between OPT-IMMALG, PSO (particle swarm optimization), arPSO (attractive and repulsive particle swarm optimization) and SEA (simple evolutionary algorithm) [42], using 30 dimensions

	OPT-IMMALG		PSO [42]	arPSO [42]	SEA [42]
	$\alpha = \frac{e^{-\hat{f}(x)}}{\rho}$	$\alpha = e^{-\rho \hat{f}(x)}$			
f_1	0.0 0.0	0.0	0.0	6.8×10^{-13} 5.3×10^{-13}	1.79×10^{-3} 2.77×10^{-4}
f_2	0.0 0.0	0.0	0.0	2.09×10^{-2} 1.48×10^{-1}	1.72×10^{-2} 1.7×10^{-3}
f_3	0.0 0.0	0.0	0.0	0.0 2.13×10^{-25}	1.59×10^{-2} 4.25×10^{-3}
f_4	5.6×10^{-4} 2.18×10^{-3}	0.0	2.11×10^{-16} 8.01×10^{-16}	1.42×10^{-5} 8.27×10^{-6}	1.98×10^{-2} 2.07×10^{-3}
f_5	21.16 11.395	12 13.22	4.026 4.99	$3.55 \times 10^{+2}$ $2.15 \times 10^{+3}$	31.32 17.4
f_6	0.0 0.0	0.0	4×10^{-2} 1.98×10^{-1}	18.98 63	0.0 0.0
f_7	3.7×10^{-5} 5.62×10^{-5}	1.52×10^{-5} 2.05×10^{-5}	1.91×10^{-3} 1.14×10^{-3}	3.89×10^{-4} 4.78×10^{-4}	7.11×10^{-4} 3.27×10^{-4}
f_8	$-1.257 \times 10^{+4}$ 8.369	$-1.256 \times 10^{+4}$ 25.912	$-7.187 \times 10^{+3}$ $6.72 \times 10^{+2}$	$-8.598 \times 10^{+3}$ $2.07 \times 10^{+3}$	$-1.167 \times 10^{+4}$ $2.34 \times 10^{+2}$
f_9	0.0 0.0	0.0	49.17 16.2	2.15 4.91	7.18×10^{-1} 9.22×10^{-1}
f_{10}	4.74×10^{-16} 1.21×10^{-15}	0.0	1.4 7.91×10^{-1}	1.84×10^{-7} 7.15×10^{-8}	1.05×10^{-2} 9.08×10^{-4}
f_{11}	0.0 0.0	0.0	2.35×10^{-2} 3.54×10^{-2}	9.23×10^{-2} 3.41×10^{-1}	4.64×10^{-3} 3.96×10^{-3}
f_{12}	1.787×10^{-21} 5.06×10^{-23}	1.77×10^{-21} 7.21×10^{-24}	3.819×10^{-1} 8.4×10^{-1}	8.559×10^{-3} 4.79×10^{-2}	4.56×10^{-6} 8.11×10^{-7}
f_{13}	1.702×10^{-21} 4.0628×10^{-23}	1.686×10^{-21} 1.149×10^{-24}	-5.969×10^{-1} 5.17×10^{-1}	-9.626×10^{-1} 5.14×10^{-1}	-1.143 1.34×10^{-5}
f_{14}	9.98×10^{-1} 5.328×10^{-4}	9.98×10^{-1} 2.719×10^{-4}	1.157 3.68×10^{-1}	9.98×10^{-1} 2.13×10^{-8}	9.98×10^{-1} 4.33×10^{-8}
f_{15}	3.26×10^{-4} 3.64×10^{-5}	3.215×10^{-4} 2.56×10^{-5}	1.338×10^{-3} 3.94×10^{-3}	1.248×10^{-3} 3.96×10^{-3}	3.704×10^{-4} 8.78×10^{-5}
f_{16}	-1.023 1.52×10^{-2}	-1.017 3.625×10^{-2}	-1.032 3.84×10^{-8}	-1.032 3.84×10^{-8}	-1.032 3.16×10^{-8}
f_{17}	4.19×10^{-1} 2.9×10^{-2}	4.2×10^{-1} 3.5158×10^{-2}	3.98×10^{-1} 5.01×10^{-9}	3.98×10^{-1} 5.01×10^{-9}	3.98×10^{-1} 2.20×10^{-8}
f_{18}	4.973 2.9366	5.371 3.0449	3.0 0.0	3.516 3.65	3.0 0.0
f_{21}	-10.15 1.81×10^{-6}	-10.15 1.018×10^{-7}	-5.4 3.40	-8.18 2.60	-8.41 3.16
f_{22}	-10.4 1.19×10^{-6}	-10.4 9.3×10^{-6}	-6.946 3.70	-8.435 2.83	-8.9125 2.86
f_{23}	-10.54 6.788×10^{-7}	-10.54 7.29×10^{-6}	-6.71 3.77	-8.616 2.88	-9.8 2.24

For OPT-IMMALG we show the results obtained using both potential mutations (Eqs. 4, 5). For all algorithms we report *mean* of the best candidate solutions on all runs (in the first line of each table entry), and the *standard deviation* (in the second line). The best results are highlighted in boldface. Results have been averaged over 30 independent runs and $T_{max} = 5 \times 10^5$

Table 14 Comparison between OPT-IMMALG, PSO (particle swarm optimization), arPSO (attractive and repulsive particle swarm optimization) and SEA (simple evolutionary algorithm) [42], using 100 dimensions

	OPT	-IMMALG	PSO [42]	arPSO [42]	SEA [42]
	$\alpha = \frac{e^{-\hat{f}(x)}}{\rho}$	$\alpha = e^{-\rho \hat{f}(x)}$			
f_1	0.0 0.0	0.0 0.0	0.0 0.0	$7.4869 \times 10^{+2}$ $2.31 \times 10^{+3}$	5.229×10^{-4} 5.18×10^{-5}
f_2	0.0 0.0	0.0 0.0	$1.804 \times 10^{+1}$ $6.52 \times 10^{+1}$	$3.9637 \times 10^{+1}$ $2.45 \times 10^{+1}$	1.737×10^{-2} 9.43×10^{-4}
f_3	0.0 0.0	0.0 0.0	$3.666 \times 10^{+3}$ $6.94 \times 10^{+3}$	$1.817 \times 10^{+1}$ $2.50 \times 10^{+1}$	6.68×10^{-2} 3.06×10^{-3}
f_4	7.32×10^{-4} 2.109×10^{-3}	6.447×10^{-7} 3.338×10^{-6}	5.312 8.63×10^{-1}	2.4367 3.80×10^{-1}	7.6708×10^{-3} 5.71×10^{-4}
f_5	97.02 54.73	74.99 38.99	$2.02 \times 10^{+2}$ $7.66 \times 10^{+2}$	$2.36 \times 10^{+2}$ $1.25 \times 10^{+2}$	$9.249 \times 10^{+1}$ $1.29 \times 10^{+1}$
f_6	0.0 0.0	0.0 0.0	2.1 3.52	$4.118 \times 10^{+2}$ $4.21 \times 10^{+2}$	0.0 0.0
f_7	1.763×10^{-5} 2.108×10^{-5}	1.59×10^{-5} 3.61×10^{-5}	2.784×10^{-2} 7.31×10^{-2}	3.23×10^{-3} 7.87×10^{-4}	7.05×10^{-4} 9.70×10^{-5}
f_8	$-4.176 \times 10^{+4}$ $2.08 \times 10^{+2}$	$-4.16 \times 10^{+4}$ $2.06 \times 10^{+2}$	$-2.1579 \times 10^{+4}$ $1.73 \times 10^{+3}$	$-2.1209 \times 10^{+4}$ $2.98 \times 10^{+3}$	$-3.943 \times 10^{+4}$ $5.36 \times 10^{+2}$
f_9	0.0 0.0	0.0 0.0	$2.4359 \times 10^{+2}$ $4.03 \times 10^{+1}$	$4.809 \times 10^{+1}$ 9.54	9.9767×10^{-2} 3.04×10^{-1}
f_{10}	1.18×10^{-16} 6.377×10^{-16}	0.0 0.0	4.49 1.73	5.628×10^{-2} 3.08×10^{-1}	2.93×10^{-3} 1.47×10^{-4}
f_{11}	0.0 0.0	0.0 0.0	4.17×10^{-1} 6.45×10^{-1}	8.53×10^{-2} 2.56×10^{-1}	1.89×10^{-3} 4.42×10^{-3}
f_{12}	5.34×10^{-22} 9.81×10^{-24}	5.3169×10^{-22} 5.0655×10^{-24}	1.77×10^{-1} 1.75×10^{-1}	9.219×10^{-2} 4.61×10^{-1}	2.978×10^{-7} 2.76×10^{-8}
f_{13}	1.712×10^{-21} 9.379×10^{-23}	1.689×10^{-21} 9.877×10^{-24}	-3.86×10^{-1} 9.47×10^{-1}	$3.301 \times 10^{+2}$ $1.72 \times 10^{+3}$	-1.142810 2.41×10^{-8}

For OPT-IMMALG we show the results obtained using both potential mutations (Eqs. 4, 5). For all algorithms we report *mean* of the best candidate solutions on all runs (in the first line of each table entry), and the *standard deviation* (in the second line). The best results are highlighted in boldface. Results have been averaged over 30 independent runs and $T_{max} = 5 \times 10^6$

Table 15 Comparison between OPT-IMMALG and several Real Coded Memetic Algorithm (RCMA) proposed in Noman and Iba [32]

Algorithm	f_{11}	f_9	f_5
OPT-IMMALG	0.0	0.0	4.68
CHC	6.5×10^{-3}	$1.6 \times 10^{+1}$	$1.9 \times 10^{+1}$
G3-1	5.1×10^{-1}	$7.4 \times 10^{+1}$	$2.8 \times 10^{+1}$
SW-100	2.7×10^{-2}	7.6	$1 \times 10^{+1}$
FC	3.5×10^{-4}	5.5	$2.3 \times 10^{+1}$
RCMA-XHC	1.3×10^{-2}	1.4	2.2

We report the *mean* of the best individuals on all runs. The best results are highlighted in boldface. Results have been averaged over 30 independent runs, using $T_{max} = 10^5$ and $n = 25$ dimensions

Table 16 Comparison between the OPT-IMMALG and OPT-IMMALG*, with maximum number of objective function evaluations, $(T_{max}) = 5 \times 10^5$ for dimension $n = 30$, and $(T_{max}) = 5 \times 10^6$ for dimension $n = 100$

	$n = 30$		$n = 100$	
	OPT-IMMALG	OPT-IMMALG*	OPT-IMMALG	OPT-IMMALG*
f_1	0.0 0.0	0.0 0.0	0.0 0.0	0.0 0.0
f_2	0.0 0.0	0.0 0.0	0.0 0.0	0.0 0.0
f_3	0.0 0.0	0.0 0.0	0.0 0.0	0.0 0.0
f_4	0.0 0.0	0.0 0.0	6.447×10^{-7} 3.338×10^{-6}	0.0 0.0
f_5	12 13.22	0.0 0.0	74.99 38.99	22.116 39.799
f_6	0.0 0.0	0.0 0.0	0.0 0.0	0.0 0.0
f_7	1.521×10^{-5} 2.05×10^{-5}	7.4785×10^{-6} 6.463×10^{-6}	1.59×10^{-5} 3.61×10^{-5}	1.2×10^{-6} 1.53×10^{-6}
f_8	$-1.256041 \times 10^{+4}$ 25.912	$-9.05 \times 10^{+3}$ $1.91 \times 10^{+4}$	$-4.16 \times 10^{+4}$ $2.06 \times 10^{+2}$	$-2.727 \times 10^{+4}$ 7.627×10^{-4}
f_9	0.0 0.0	0.0 0.0	0.0 0.0	0.0 0.0
f_{10}	0.0 0.0	0.0 0.0	0.0 0.0	0.0 0.0
f_{11}	0.0 0.0	0.0 0.0	0.0 0.0	0.0 0.0
f_{12}	0.0 0.0	0.0 0.0	0.0 0.0	0.0 0.0
f_{13}	0.0 0.0	0.0 0.0	0.0 0.0	0.0 0.0

We report *mean* of the best candidate solutions on all runs (in the first line of each table entry), and the *standard deviation* (in the second line). The best results are highlighted in boldface

noticed that randomly choosing the age of the candidate solutions in the range $[0, \frac{2}{3}\tau_B]$ and fixing $\theta = 50\%$, OPT-IMMALG improves its own performances. We called this new variant OPT-IMMALG*.

Table 16 shows the improved performance of OPT-IMMALG* on the first 13 functions. For these experiments we fixed $T_{max} = 5 \times 10^5$ for 30 dimensions, and $T_{max} = 5 \times 10^6$ for 100 dimensions, which corresponds to the experimental protocol used in the previous subsection and proposed in Versterstrøm and Thomsen [42]. All the results with value $\leq 10^{-25}$ were reported as 0.0. The improved performance is particularly evident for function f_5 with $n = 30$, where now OPT-IMMALG* is able to reach the best solution while the previous variants failed.

In Tables 17 and 18, we present again the comparison with FEP but this time including the new variant of OPT-IMMALG. Table 17 shows the results obtained by OPT-IMMALG* on the first 13 functions, whilst Table 18 the results on the multimodal functions with a few local optima ($f_{14} - f_{23}$). The new variant OPT-IMMALG*, improves the overall quality of the results, in particular for functions f_5 , and f_9 . Opposite behaviour is instead obtained in Table 18, where the new variant (OPT-IMMALG*) is comparable, but not outperforming the performances of OPT-IMMALG. Most likely, for this class of functions, each candidate solution still needs longer life span.

Table 17 Comparison between OPT-IMMALG, OPT-IMMALG*, and FEP (Fast Evolutionary Algorithm) [43], on the first 13 functions

	OPT-IMMALG	OPT-IMMALG*	FEP [43]		OPT-IMMALG	OPT-IMMALG*	FEP [43]
f_1	0.0	0.0	5.7×10^{-4}	f_8	-12535.15	-8707.04	-12554.5
	0.0	0.0	1.3×10^{-4}		62.81	1.7×10^3	52.6
f_2	0.0	0.0	8.1×10^{-3}	f_9	0.596	0.0	4.6×10^{-2}
	0.0	0.0	7.7×10^{-4}		4.178	0.0	1.2×10^{-2}
f_3	0.0	0.0	1.6×10^{-2}	f_{10}	0.0	0.0	1.8×10^{-2}
	0.0	0.0	1.4×10^{-2}		0.0	0.0	2.1×10^{-3}
f_4	0.0	0.0	0.3	f_{11}	0.0	0.0	1.6×10^{-2}
	0.0	0.0	0.5		0.0	0.0	2.2×10^{-2}
f_5	16.29	0.0	5.06	f_{12}	0.0	0.0	9.2×10^{-6}
	13.96	0.0	5.87		0.0	0.0	3.6×10^{-6}
f_6	0.0	0.0	0.0	f_{13}	0.0	0.0	1.6×10^{-4}
	0.0	0.0	0.0		0.0	0.0	7.3×10^{-5}
f_7	1.995×10^{-5}	1.6×10^{-5}	7.6×10^{-3}				
	2.348×10^{-5}	1.37×10^{-5}	2.6×10^{-3}				

The used experimental protocol was the same described in Sect. 4.1. For all algorithms we report *mean* of the best candidate solutions on all runs (in the first line of each table entry), and the *standard deviation* (in the second line). The best results are highlighted in boldface

Table 18 Comparison between OPT-IMMALG, OPT-IMMALG*, and FEP (Fast Evolutionary Algorithm) [43], on all functions included into the last categories, i.e. multimodal functions with a few local optima

	OPT-IMMALG	OPT-IMMALG*	FEP [42]
f_{14}	0.998	1.255	1.22
	1.11×10^{-3}	1.14	0.56
f_{15}	3.20×10^{-4}	3.22×10^{-4}	5.0×10^{-4}
	2.672×10^{-5}	2.23×10^{-5}	3.2×10^{-4}
f_{16}	-1.013	-1.0033	-1.031
	2.212×10^{-2}	4.9×10^{-2}	4.9×10^{-7}
f_{17}	0.423	0.452	0.398
	3.217×10^{-2}	7.58×10^{-2}	1.5×10^{-7}
f_{18}	5.837	7.097	3.02
	3.742	5.61	0.11
f_{19}	-3.72	-3.65	-3.86
	7.846×10^{-3}	4.82×10^{-2}	1.4×10^{-5}
f_{20}	-3.29	-3.026	-3.27
	3.097×10^{-2}	0.12	5.9×10^{-2}
f_{21}	-10.153	-10.153	-5.52
	1.034×10^{-7}	1.46×10^{-7}	1.59
f_{22}	-10.402	-10.403	-5.52
	1.082×10^{-5}	1.75×10^{-5}	2.12
f_{23}	-10.536	-10.536	-6.57
	1.165×10^{-5}	1.76×10^{-5}	3.14

The used experimental protocol was the same described in Sect. 4.1. For all algorithms we report *mean* of the best candidate solutions on all runs (in the first line of each table entry), and the *standard deviation* (in the second line). The best results are highlighted in boldface

4.6 IA versus differential evolution algorithms

Among the many evolutionary methodologies able to effectively tackle global numerical optimization problems, differential evolution (DE) has shown better performances on complex

Table 19 Comparison between OPT-IMMALG, OPT-IMMALG*, and several DE variants, proposed in Mezura-Montes et al. [31]

Algorithm	Unimodal functions					
	f_1	f_2	f_3	f_4	f_6	f_7
OPT-IMMALG*	0.0	0.0	0.0	0.0	0.0	2.79×10^{-5}
OPT-IMMALG	0.0	0.0	0.0	0.0	0.0	4.89×10^{-5}
DE rand/1/bin	0.0	0.0	0.02	1.9521	0.0	0.0
DE rand/1/exp	0.0	0.0	0.0	3.7584	0.84	0.0
DE best/1/bin	0.0	0.0	0.0	0.0017	0.0	0.0
DE best/1/exp	407.972	3.291	10.6078	1.701872	2737.8458	0.070545
DE current-to-best/1	0.54148	4.842	0.471730	4.2337	1.394	0.0
DE current-to-rand/1	0.69966	3.503	0.903563	3.298563	1.767	0.0
DE current-to-rand/1/bin	0.0	0.0	0.000232	0.149514	0.0	0.0
DE rand/2/dir	0.0	0.0	30.112881	0.044199	0.0	0.0

Algorithm	Multimodal functions					
	f_5	f_9	f_{10}	f_{11}	f_{12}	f_{13}
OPT-IMMALG*	16.2	0.0	0.0	0.0	0.0	0.0
OPT-IMMALG	11.69	0.0	0.0	0.0	0.0	0.0
DE rand/1/bin	19.578	0.0	0.0	0.001117	0.0	0.0
DE rand/1/exp	6.696	97.753938	0.080037	0.000075	0.0	0.0
DE best/1/bin	30.39087	0.0	0.0	0.000722	0.0	0.000226
DE best/1/exp	132621.5	40.003971	9.3961	5.9278	1293.0262	2584.85
DE current-to-best/1	30.984666	98.205432	0.270788	0.219391	0.891301	0.038622
DE current-to-rand/1	31.702063	92.263070	0.164786	0.184920	0.464829	5.169196
DE current-to-rand/1/bin	24.260535	0.0	0.0	0.0	0.001007	0.000114
DE rand/2/dir	30.654916	0.0	0.0	0.0	0.0	0.0

We report the mean of the best individuals on all runs. The best results are highlighted in boldface. Results have been averaged over 100 independent runs, using $T_{max} = 1.2 \times 10^5$, and $n = 30$ dimensions. For OPT-IMMALG* we fixed $d = 100$

and continuous search space [34,36]. For this purpose, we compared OPT-IMMALG and opt-IMMALG* with several DE variants [31,42], and their memetic versions [32] using the first 13 functions from Table 1. As previously described, for this class of experiments we used only the second potential mutation (Eq. 5) because it presents better performances. Several dimensions were used, from small ($n = 30$) to high values ($n = 200$). For these instances we fixed ρ as described in Sect. 3.1. In the first experiment, OPT-IMMALG and OPT-IMMALG* are compared with 8 DE variants, proposed in Mezura-Montes et al. [31], where T_{max} was fixed to 1.2×10^5 [31]. For each function 100 independent runs were performed, and the variable dimensions were fixed to 30. Results are shown in Table 19. Since the authors of Mezura-Montes et al. [31] modified the function f_8 to have its minimum at zero (rather than -12569.5), this function is not included in the table. Inspecting the comparison in the table, we can observe that the new variant OPT-IMMALG* outperforms all DE variants except for the functions f_5 and f_7 .

In Table 20 OPT-IMMALG and OPT-IMMALG* algorithms are compared to the rand/1/bin variant, one of the best DE variants, based on a different experimental protocol proposed in Versterstrøm and Thomsen [42]. For each experiment two different dimension values were used: $n = 30$ with $T_{max} = 5 \times 10^5$, and $n = 100$ with $T_{max} = 5 \times 10^6$. Thirty independent runs were performed for each benchmark function. In this table we present the mean of the best candidate solutions on all runs and the standard deviation (in a new line). All results

Table 20 Comparison between OPT-IMMALG and *rand/l/bin* variant, proposed in Versterstrøm and Thomsen [42]

	30 dimension			100 dimension		
	OPT-IMMALG	OPT-IMMALG*	DE <i>rand/l/bin</i> [42]	OPT-IMMALG	OPT-IMMALG*	DE <i>rand/l/bin</i> [42]
f_1	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0
f_2	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0
f_3	0.0	0.0	2.02×10^{-9}	0.0	0.0	5.87×10^{-10}
	0.0	0.0	8.26×10^{-10}	0.0	0.0	1.83×10^{-10}
f_4	0.0	0.0	3.85×10^{-8}	6.447×10^{-7}	0.0	1.128×10^{-9}
	0.0	0.0	9.17×10^{-9}	3.338×10^{-6}	0.0	1.42×10^{-10}
f_5	12	0.0	0.0	74.99	22.116	0.0
	13.22	0.0	0.0	38.99	39.799	0.0
f_6	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0
f_7	1.521×10^{-5}	7.48×10^{-6}	4.939×10^{-3}	1.59×10^{-5}	1.2×10^{-6}	7.664×10^{-3}
	2.05×10^{-5}	6.46×10^{-6}	1.13×10^{-3}	3.61×10^{-5}	1.53×10^{-6}	6.58×10^{-4}
f_8	$-1.256041 \times 10^{+4}$	$-9.05 \times 10^{+3}$	$-1.256948 \times 10^{+4}$	$-4.16 \times 10^{+4}$	$-2.727 \times 10^{+4}$	$-4.1898 \times 10^{+4}$
	25.912	1.91×10^4	2.3×10^{-4}	$2.06 \times 10^{+2}$	7.63×10^{-4}	1.06×10^{-3}
f_9	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0
f_{10}	0.0	0.0	-1.19×10^{-15}	0.0	0.0	8.023×10^{-15}
	0.0	0.0	7.03×10^{-16}	0.0	0.0	1.74×10^{-15}
f_{11}	0.0	0.0	0.0	0.0	0.0	5.42×10^{-20}
	0.0	0.0	0.0	0.0	0.0	5.42×10^{-20}
f_{12}	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0
f_{13}	0.0	0.0	-1.142824	0.0	0.0	-1.142824
	0.0	0.0	4.45×10^{-8}	0.0	0.0	2.74×10^{-8}

We report the *mean* of the best individuals on all runs (in the first line of each table entry), and the *standard deviation* (in the second line). The best results are highlight in boldface. The results are obtained using $n = 30$ and $n = 100$ dimensions

$\leq 10^{-25}$ were reported as 0.0 [42]. This is the same experimental protocol used for the results in Table 16, hence the two tables are similar. The results indicate that the overall performances of OPT-IMMALG and OPT-IMMALG* are comparable to the ones produced by *rand/l/bin* variant, in both 30 and 100 variables dimension. Two memetic versions of DE variants, based on *crossover local search* (XLS) and called DEfirDE and DEfirSPX, were proposed in Noman and Iba [32]. Then, as last experiments, in Tables 21 and 22 we compared OPT-IMMALG*, and OPT-IMMALG with these two DE algorithms, *rand/l/exp* and *best/l/exp*, and their memetic versions, DEfirDE and DEfirSPX [32], using $n = \{50, 100, 200\}$ dimensions. For each test, the maximum number of objective function evaluations T_{max} was fixed to 5×10^5 , and 30 independent runs were performed. We used only the functions f_1, f_5, f_9, f_{10} and f_{11} , the same used in Noman and Iba [32].

For the two DE algorithms and their memetic versions, in Tables 21 and 22, we report the results obtained varying the population size with $n, 5n$ and $10n$, (first three lines, respectively) where n indicate the dimensional search space [32]. Both tables demonstrate that the two variants of OPT-IMMALG achieve higher quality solutions rather than two DE algorithms

Table 21 Comparison between OPT-IMMALG*, OPT-IMMALG and two of the best DE variants, *rand/1/exp* and *best/1/exp*, proposed in Noman and Iba [32]

	OPT-IMMALG*	OPT-IMMALG	DE <i>rand/1/exp</i> [32]	DE <i>best/1/exp</i> [32]
<i>n</i> = 50 dimensional search space				
<i>f</i> ₁	0 ± 0	0 ± 0	0 ± 0 0 ± 0 0.0535 ± 0.0520	309.74 ± 481.05 0 ± 0 0.0027 ± 0.0013
<i>f</i> ₅	1.64 ± 8.7	30 ± 21.7	79.8921 ± 102.611 52.4066 ± 19.9109 90.0213 ± 33.8734	3.69 × 10 ⁺⁵ ± 5.011 × 10 ⁺⁵ 54.5985 ± 25.6652 58.1931 ± 9.4289
<i>f</i> ₉	0 ± 0	0 ± 0	0 ± 0 0 ± 0 0 ± 0	0.61256 ± 1.1988 0 ± 0 0 ± 0
<i>f</i> ₁₀	0 ± 0	0 ± 0	0 ± 0 9.36 × 10 ⁻⁶ ± 3.67 × 10 ⁻⁶ 0.0104 ± 0.0015	0.2621 ± 0.5524 6.85 × 10 ⁻⁶ ± 6.06 × 10 ⁻⁶ 0.0067 ± 0.0015
<i>f</i> ₁₁	0 ± 0	0 ± 0	0 ± 0 9.95 × 10 ⁻⁷ ± 4.3 × 10 ⁻⁷ 0.0053 ± 0.010	0.1651 ± 0.2133 0 ± 0 0.0012 ± 0.0028
<i>n</i> = 100 dimensional search space				
<i>f</i> ₁	0 ± 0	0 ± 0	1.58 × 10 ⁻⁶ ± 3.75 × 10 ⁻⁶ 59.926 ± 16.574 2496.82 ± 246.55	0.0046 ± 0.0247 30.242 ± 5.93 1729.40 ± 172.28
<i>f</i> ₅	26.7 ± 43	85.6 ± 31.758	120.917 ± 41.8753 12312.16 ± 3981.44 3.165 × 10 ⁺⁶ ± 6.052 × 10 ⁺⁵	178.465 ± 60.938 7463.633 ± 2631.92 1.798 × 10 ⁺⁶ ± 3.304 × 10 ⁺⁵
<i>f</i> ₉	0 ± 0	0 ± 0	0 ± 0 2.6384 ± 0.7977 234.588 ± 13.662	0 ± 0 0.7585 ± 0.2524 198.079 ± 18.947
<i>f</i> ₁₀	0 ± 0	0 ± 0	1.02 × 10 ⁻⁶ ± 1.6 × 10 ⁻⁷ 1.6761 ± 0.0819 7.7335 ± 0.1584	9.5 × 10 ⁻⁷ ± 1.1 × 10 ⁻⁷ 1.2202 ± 0.0965 6.7251 ± 0.1373
<i>f</i> ₁₁	0 ± 0	0 ± 0	0 ± 0 1.1316 ± 0.0124 20.037 ± 0.9614	0 ± 0 1.0530 ± 0.0100 13.068 ± 0.8876
<i>n</i> = 200 dimensional search space				
<i>f</i> ₁	0 ± 0	0 ± 0	50.005 ± 16.376 5.45 × 10 ⁺⁴ ± 2605.73 1.82 × 10 ⁺⁵ ± 6785.18	26.581 ± 7.4714 4.84 × 10 ⁺⁴ ± 1891.24 1.74 × 10 ⁺⁵ ± 6119.01
<i>f</i> ₅	88.65 ± 91.85	165.1 ± 71.2	9370.17 ± 3671.11 4.22 × 10 ⁺⁸ ± 3.04 × 10 ⁺⁷ 3.29 × 10 ⁺⁹ ± 2.12 × 10 ⁺⁸	6725.48 ± 1915.38 3.54 × 10 ⁺⁸ ± 3.54 × 10 ⁺⁷ 3.12 × 10 ⁺⁹ ± 1.65 × 10 ⁺⁸
<i>f</i> ₉	0 ± 0	0 ± 0	0.4245 ± 0.2905 1878.61 ± 60.298 5471.35 ± 239.67	0.2255 ± 0.1051 1761.55 ± 43.3824 5094.97 ± 182.77
<i>f</i> ₁₀	0 ± 0	0 ± 0	0.5208 ± 0.0870 15.917 ± 0.1209 19.253 ± 0.0698	0.4322 ± 0.0427 15.46 ± 0.1205 19.138 ± 0.0772
<i>f</i> ₁₁	0 ± 0	0 ± 0	0.7687 ± 0.0768 490.29 ± 21.225 1657.93 ± 47.142	0.5707 ± 0.0651 441.97 ± 15.877 1572.51 ± 53.611

We report the *mean* of the best individuals on all runs and the *standard deviation* (*mean* ± *sd*). The best results are highlight in boldface. The results are obtained using *n* = {50, 100, 200} dimensions

Table 22 Comparison between OPT-IMMALG*, OPT-IMMALG and the memetic versions of *rand/l/exp* and *best/l/exp* DE variants, called DefirDE and DefirSPX [32]

	OPT-IMMALG *	OPT-IMMALG	DefirDE [32]	DefirSPX [32]
<i>n</i> = 50 dimensional search space				
<i>f</i> ₁	0 ± 0	0 ± 0	0 ± 0 0 ± 0	0 ± 0 0 ± 0
<i>f</i> ₅	1.64 ± 8.7	30 ± 21.7	0.0026 ± 0.0023 72.0242 ± 47.1958 53.1894 ± 26.1913 66.9674 ± 23.7196	1 × 10 ⁻⁴ ± 4.75 × 10 ⁻⁵ 65.8951 ± 37.8933 45.8367 ± 10.2518 52.0033 ± 13.6881
<i>f</i> ₉	0 ± 0	0 ± 0	0 ± 0 0 ± 0 0 ± 0	0 ± 0 0 ± 0 0 ± 0
<i>f</i> ₁₀	0 ± 0	0 ± 0	0 ± 0 2.28 × 10 ⁻⁵ ± 1.45 × 10 ⁻⁵ 0.0060 ± 0.0015	0 ± 0 3.0 × 10 ⁻⁶ ± 1.07 × 10 ⁻⁶ 0.0019 ± 4.32 × 10 ⁻⁴
<i>f</i> ₁₁	0 ± 0	0 ± 0	0 ± 0 0 ± 0 4.96 · 10 ⁻⁴ ± 6.68 · 10 ⁻⁴	0 ± 0 0 ± 0 5.27 × 10 ⁻⁴ ± 0.0013
<i>n</i> = 100 dimensional search space				
<i>f</i> ₁	0 ± 0	0 ± 0	0 ± 0 11.731 ± 5.0574 358.57 ± 108.12	0 ± 0 1.2614 ± 0.4581 104.986 ± 22.549
<i>f</i> ₅	26.7 ± 43	85.6 ± 31.758	107.5604 ± 28.2529 2923.108 ± 1521.085 2.822 × 10 ⁺⁵ ± 3.012 × 10 ⁺⁵	99.1086 ± 18.5735 732.85 ± 142.22 16621.32 ± 6400.43
<i>f</i> ₉	0 ± 0	0 ± 0	0 ± 0 0.1534 ± 0.1240 17.133 ± 7.958	0 ± 0 0.0094 ± 0.0068 27.0537 ± 20.889
<i>f</i> ₁₀	0 ± 0	0 ± 0	1.2 × 10 ⁻⁶ ± 6.07 × 10 ⁻⁷ 0.5340 ± 0.1101 3.7515 ± 0.2773	0 ± 0 0.3695 ± 0.0734 3.4528 ± 0.1797
<i>f</i> ₁₁	0 ± 0	0 ± 0	0 ± 0 0.7725 ± 0.1008 3.7439 ± 0.7651	0 ± 0 0.5433 ± 0.1331 2.2186 ± 0.3010
<i>n</i> = 200 dimensional search space				
<i>f</i> ₁	0 ± 0	0 ± 0	17.678 ± 9.483 9056.0 ± 1840.45 44090.5 ± 6122.35	0.8568 ± 0.2563 2782.32 ± 335.69 9850.45 ± 1729.9
<i>f</i> ₅	88.65 ± 91.85	165.1 ± 71.2	5302.79 ± 2363.74 2.39 × 10 ⁺⁷ ± 6.379 × 10 ⁺⁶ 3.48 × 10 ⁺⁸ ± 1.75 × 10 ⁺⁸	996.69 ± 128.483 1.19 × 10 ⁺⁶ ± 4.10 × 10 ⁺⁵ 1.21 × 10 ⁺⁷ ± 4.73 × 10 ⁺⁶
<i>f</i> ₉	0 ± 0	0 ± 0	0.1453 ± 0.2771 352.93 ± 46.11 1193.83 ± 145.477	0.0024 ± 0.0011 369.88 ± 136.87 859.03 ± 99.76
<i>f</i> ₁₀	0 ± 0	0 ± 0	0.3123 ± 0.0426 9.2373 ± 0.4785 14.309 ± 0.3706	0.1589 ± 0.0207 6.6861 ± 0.3286 9.4114 ± 0.4581
<i>f</i> ₁₁	0 ± 0	0 ± 0	0.5984 ± 0.1419 78.692 ± 11.766 368.90 ± 41.116	0.1631 ± 0.0314 28.245 ± 4.605 85.176 ± 12.824

We report the *mean* of the best individuals on all runs and the *standard deviation* (*mean* ± *sd*). The best results are highlight in boldface. The showed results are obtained using *n* = {50, 100, 200} dimensional search space

and their memetic versions, especially for function f_5 . In a both tables, it is significant the difference in solution quality obtained by OPT-IMMALG* for function f_5 compared to the other algorithms. No one of the compared algorithms has been able to reach comparable solutions to OPT-IMMALG* on this function. Moreover, both tables indicate that both variants of OPT-IMMALG outperform the other algorithms when the function dimension increases. Finally, it is important to highlight that the two variants of OPT-IMMALG were ran using smaller population size, in particular for high dimensions ($n = \{100, 200\}$).

4.7 IA versus swarm intelligence algorithms

Recently, artificial immune systems have been related to swarm systems, since many immunological algorithms operate in a very similar manner: the design of distributed systems, which display emergent behaviour at a system level, based on low-level interactions between agents and the environment. Therefore, some swarm intelligence algorithms, proposed in Karaboga and Baturk [29], have been taken into account and compared with only OPT-IMMALG*, since the latter seems to have best performances than the other variants: *particle swam optimization* (PSO); *particle swarm inspired evolutionary algorithm* (PS-EA), and *artificial bee colony* (ABC). For this kind of experiments we have used the same experimental protocol used in Karaboga and Baturk [29], that is: the problem dimension has been set $n = \{10, 20, 30\}$, whilst the termination criterion has been fixed to 500 (for $n = 10$), 750 (for $n = 20$), and 1000 (for $n = 30$) generations.

Similarly to Karaboga and Baturk [29] in this comparison, shown in Table 23, we considered only the following functions: f_5, f_9, f_{10} and f_{11} of the benchmark in Table 1, with the addition of the following new function:

$$H(x) = (418.9829 \times n) + \sum_{i=1}^n -x_i \sin\left(\sqrt{|x_i|}\right) \tag{11}$$

From Table 23, it is possible to affirm that OPT-IMMALG* outperforms all swarm system algorithms on all used functions, except for function H . The best performances of OPT-IMMALG* over the used swarm intelligent algorithms are also confirmed for an increasing problem dimension. Conversely, for the function H , PS-EA reaches better solutions with increasing problem dimension. Finally, the only results reported for ABC₂ were obtained using a different experimental protocol (see Karaboga and Baturk [29]): the termination criterion was increased to 1000 (for $n = 10$), 1500 (for $n = 20$) and 2000 (for $n = 30$), respectively. Although OPT-IMMALG* has been tested with a smaller number of generations is possible to notice that the results are comparable, and often outperform ABC₂. This experiment displays as OPT-IMMALG* reaches competitive solutions, close to global optima, in less time than artificial bee colony (ABC) algorithm.

4.8 IA versus LeGO and PSwarm

In this section we present the comparisons between OPT-IMMALG* and two of the best optimization algorithms present in literature, as LEGO [5] and PSWARM [41]. For this kind of comparison we have used a different set of functions taken from Cassioli et al. [5], which includes 8 functions with $n = 10$ as dimensionality of the variables, except for the function mgw_{20} , where $n = 20$. These functions represent a subset of the widest benchmark proposed in Vaz and Vicente [41], which can be downloaded from <http://www.norg.uminho.pt/aivaz/>

Table 23 Comparison between OPT-IMMALG* and some Swarm Intelligence algorithms

Algorithm	f_{11}	f_9	f_5	f_{10}	H
<i>10 variables</i>					
PSO	0.079393	2.6559	4.3713	9.8499×10^{-13}	161.87
	0.033451	1.3896	2.3811	9.6202×10^{-13}	144.16
PS- EA	0.222366	0.43404	25.303	0.19209	0.32037
	0.0781	0.2551	29.7964	0.1951	1.6185
OPT-IMMALG*	0.0	0.0	0.0	0.0	1.27×10^{-4}
	0.0	0.0	0.0	0.0	1.268×10^{-14}
ABC ₁	0.00087	0.0	0.034072	7.8×10^{-11}	1.27×10^{-9}
	0.002535	0.0	0.045553	1.16×10^{-9}	4×10^{-12}
ABC ₂	0.000329	0.0	0.012522	4.6×10^{-11}	1.27×10^{-9}
	0.00185	0.0	0.01263	5.4×10^{-11}	4×10^{-12}
<i>20 variables</i>					
PSO	0.030565	12.059	77.382	1.1778×10^{-6}	543.07
	0.025419	3.3216	94.901	1.5842×10^{-6}	360.22
PS- EA	0.59036	1.8135	72.452	0.32321	1.4984
	0.2030	0.2551	27.3441	0.097353	0.84612
OPT-IMMALG*	0.0	0.0	0.0	0.0	237.5652
	0.0	0.0	0.0	0.0	710.4036
ABC ₁	2.01×10^{-8}	1.45×10^{-8}	0.13614	1.6×10^{-11}	19.83971
	6.76×10^{-8}	5.06×10^{-8}	0.132013	1.9×10^{-11}	45.12342
ABC ₂	0.0	0.0	0.014458	0.0	0.000255
	0.0	0.0	0.010933	1×10^{-12}	0
<i>30 variables</i>					
PSO	0.011151	32.476	402.54	1.4917×10^{-6}	990.77
	0.014209	6.9521	633.65	1.8612×10^{-6}	581.14
PS- EA	0.8211	3.0527	98.407	0.3771	3.272
	0.1394	0.9985	35.5791	0.098762	1.6185
OPT-IMMALG*	0.0	0.0	0.0	0.0	2766.804
	0.0	0.0	0.0	0.0	2176.288
ABC ₁	2.87×10^{-9}	0.033874	0.219626	3×10^{-12}	146.8568
	8.45×10^{-10}	0.181557	0.152742	5×10^{-12}	82.3144
ABC ₂	0.0	0.0	0.020121	0.0	0.000382
	0.0	0.0	0.021846	0.0	1×10^{-12}

For each function is showed the *mean* of the best candidate solutions on 30 independent runs (in the first line of each table entry), and *standard deviation* (in the second line). The best results are highlighted in boldface

[pswarm/](#). Table 24 presents the comparison among OPT-IMMALG*, LEGO and PSWARM, showing for each function the *minimum*, *median* and *maximum* found, except for PSWARM because only the minimal value found is known. Precisely, the results for PSWARM were taken from Vaz and Vicente [41], where is given the *optimality gap*. For this kind of comparison, as proposed in Vaz and Vicente [41] and Cassioli et al. [5], 30 independent runs were performed for any test function, with T_{max} fixed to 10^4 . Thus, also the *mean* of the best solutions obtained on the 30 runs was included into the table. For LEGO algorithm we show the best solutions found from 5000 points accepted (first relative line), and from 5000 refused ones.

Since for these experiments a small value of objective function evaluations was set, then a smaller population size (respect the previous comparisons) for OPT-IMMALG* has been used: $d = 40$.

Table 24 Comparison between OPT-IMMALG*, LEGO [5] and PSWARM [41] algorithms

Algorithm	Min	Mean	Median	Max
<i>ack</i> (global minimum at $f(x) = 0$)				
PSWARM	0.217164	n. a.	n. a.	n. a.
LEGO	2.04	4.74	4.85	5.36
	4.59	6.06	6.03	7.97
OPT-IMMALG*	0	0	0	0
<i>em</i> ₁₀ (global minimum at $f(x) = -9.660152$)				
PSWARM	-8.275452	n. a.	n. a.	n. a.
LEGO	-8.88	-5.16	-5.24	-0.488
	-8.68	-4.12	-4.18	0.002
OPT-IMMALG*	-6.3086	-5.22	-5.225	-4.446
<i>fx</i> ₁₀ (global minimum at $f(x) = -10.2088$)				
PSWARM	-2.131509	n. a.	n. a.	n. a.
LEGO	-10.21	-1.97	-1.48	-1.28
	-10.21	-1.58	-1.48	-1.15
OPT-IMMALG*	-2.2	-0.4676	-0.3887	-0.3784
<i>mgw</i> ₁₀ (global minimum at $f(x) = 0$)				
PSWARM	1.1078×10^{-2}	n. a.	n. a.	n. a.
LEGO	4.4×10^{-16}	1.9×10^{-2}	8.9×10^{-3}	3.63
	4.4×10^{-16}	4.0×10^{-2}	3.2×10^{-2}	3.63
OPT-IMMALG*	0	4.23×10^{-6}	1.13×10^{-6}	2.56×10^{-5}
<i>mgw</i> ₂₀ (global minimum at $f(x) = 0$)				
PSWARM	5.3904×10^{-2}	n. a.	n. a.	n. a.
LEGO	-1.3×10^{-15}	7.4×10^{-2}	2.5×10^{-2}	7.80
	-1.3×10^{-15}	8.4×10^{-2}	4.4×10^{-2}	9.42
OPT-IMMALG*	0	7.28×10^{-4}	3.64×10^{-5}	1.6686×10^{-2}
<i>ml</i> ₁₀ (global minimum at $f(x) = -0.965$)				
PSWARM	-0.965	n. a.	n. a.	n. a.
LEGO	-1.7×10^{-22}	1.7^{-22}	-1.0×10^{-132}	8.6×10^{-19}
	-8.3×10^{-81}	1.6^{-74}	1.9×10^{-279}	8.2×10^{-71}
OPT-IMMALG*	-7.917×10^{-2}	-3.088×10^{-3}	0	0
<i>rg</i> ₁₀ (global minimum at $f(x) = 0$)				
PSWARM	0	n. a.	n. a.	n. a.
LEGO	6.96	57.54	57.71	127.40
	9.95	81.15	80.59	224.90
OPT-IMMALG*	0	0	0	0
<i>sal</i> ₁₀ (global minimum at $f(x) = 0$)				
PSWARM	0.399873	n. a.	n. a.	n. a.
LEGO	2.1×10^{-16}	14.47	15.10	20.90
	1.2×10^{-14}	18.65	18.90	26.60
OPT-IMMALG*	0	0.113	9.987×10^{-2}	0.19987

For each function is showed the *minimum* value found, the *mean* on all independent runs, and the *median* and *max* values found. The best results are highlighted in boldface

Inspecting these results is possible to see as OPT-IMMALG* outperforms the other two optimization algorithms on 5 functions over 8, whilst in the remaining 2 functions over 3, that is *fx*₁₀ and *ml*₁₀, OPT-IMMALG* does not exhibit the worst solutions. Moreover, analyzing the obtained solutions on function *em*₁₀ by all three algorithms, is possible to see that, although OPT-IMMALG* is not able to reach a comparable minimal point with respect to the other two algorithms, it shows instead better performances with respect to the *mean* value of best solutions found, showing thus in the overall a better search strategy. We think that

Table 25 Results obtained by OPT-IMMALG using large dimensional search space, $n = \{1000, 5000\}$

	f_1	f_5	f_9	f_{10}	f_{11}
$T_{max} = 10^4$					
$n = 1000$	1.93×10^{-1}	$1.01 \times 10^{+3}$	2.29×10^{-2}	1.21×10^{-3}	1.27×10^{-2}
	2.44×10^{-2}	$2.94 \times 10^{+2}$	5.09×10^{-3}	7.76×10^{-5}	1.7×10^{-3}
$n = 5000$	16	$9.11 \times 10^{+3}$	1.83	2.76×10^{-3}	3.26×10^{-1}
	28.6	$3.56 \times 10^{+3}$	8.13	2.31×10^{-3}	5.61×10^{-1}
$T_{max} = 10^5$					
$n = 1000$	3.35×10^{-3}	$9.54 \times 10^{+2}$	7.06×10^{-4}	3.76×10^{-8}	6.66×10^{-12}
	2.22×10^{-2}	$1.54 \times 10^{+2}$	4.72×10^{-3}	2.63×10^{-7}	4.56×10^{-11}
$n = 5000$	3.52	$5.95 \times 10^{+3}$	3.64×10^{-1}	8.14×10^{-4}	8.99×10^{-2}
	5.14	$1.98 \times 10^{+3}$	$6, 34 \times 10^{-1}$	1.59×10^{-3}	3.33×10^{-1}

We performed 50 independent runs for each test function using different maximum objective function evaluation $T_{max} = \{10^4, 10^5\}$. We fixed $\rho = 9$ for $n = 1000$ and $\rho = 11.5$ for $n = 5000$. The mean of the best individuals on all runs (in the first line of each table entry), and standard deviation (in the second line) are presented

finding better solutions also in the mean, can be useful on any real optimization task, where often you need to find a good alternative solution to the optimal ones.

4.8.1 IA for high dimensional search spaces

The final set of experiments that complete this exhaustive study of the performances of the proposed IA, consists of tackling global numerical optimization problems with very high dimensions ($n = 1000$ and $n = 5000$). We present only the results obtained by OPT-IMMALG using the potential mutation of Eq. 5. Table 25 shows the results obtained by OPT-IMMALG on large dimension values using different T_{max} values: 10^4 and 10^5 . As we expected, the proposed algorithm exhibits more obstacles in reaching optimal solutions for the given functions, once is increased the dimensionality of the function. However, by increasing the number of objective function evaluations, the algorithm begins to reach acceptable solutions, and then showing better performances. This makes us think that giving more time for the evolution, the algorithm performs well also on large-scale dimensions.

5 Conclusion

In this research paper we presented an extensive comparative study illustrating the performance of two optimization immunological algorithms with 39 state-of-the-art optimization algorithms (deterministic and nature inspired methodologies): FEP; IFEP; three versions of CEP; two versions of PSO and ARPSO; PS- EA; two version of ABC; EO; SEA; HGA; immunological inspired algorithms, as BCA and two versions of CLONALG; CHC algorithm; Generalized Generation Gap ($G3 - 1$); hybrid steady-state RCMA (SW- 100), Family Competition (FC); CMA with crossover Hill Climbing (RCMA- XHC); eleven variants of DE and two its memetic versions; artificial bee colony (ABC); learning for global optimization (LEGO); and PSWARM.

Two different versions are given to solve the global numerical optimization problem: OPT-IMMALG01 based on binary-code representation, and OPT-IMMALG that is based on real values. Moreover, two variant of OPT-IMMALG are presented in this work.

The main features of the designed immunological algorithm can be summarized as: (1) the cloning operator, which explores the neighbourhood of a given solution, (2) the inversely

proportional hypermutation operator, which perturbs each candidate solution as a function of its objective function value (inversely proportionally), and (3) the *aging operator*, that eliminates the oldest candidate solutions from the current population in order to introduce diversity and thus avoiding local minima during the search process.

For our experiments, we have used a large set of test-beds and numerical functions from Cassioli et al. [5], Timmis and Kelsey [40], Vaz and Vicente [41] and Yao et al. [43]. Furthermore the dimensionality of the problems was varied from small to high dimensions (5000 variables). Our results suggest that the proposed immunological algorithm is an effective numerical optimization algorithm (in terms of solution quality) particularly for the most challenging highly dimensional search spaces. In particular, increasing the dimension of the solutions space improves the performances of IA. Moreover, experimental results indicate that our IA using real values coding reaches better solutions than the binary-code version.

All experimental comparisons show that OPT-IMMALG is comparable, and often outperforms, all 39 state-of-the-art optimization algorithms.

Acknowledgments The anonymous reviewers provided helpful feedback that measurably improved the manuscript.

References

1. Aiex, R.M., Resende, M.G.C., Ribeiro, C.C.: TTTPLOTS: a perl program to create time-to-target plots. *Optim. Lett.* **1**, 355–366 (2007)
2. Aiex, R.M., Resende, M.G.C., Ribeiro, C.C.: Probability distribution of solution time in GRASP: an experimental investigation. *J. Heuristics* **8**, 343–373 (2002)
3. Angeline, P.J.: Evolutionary optimization versus particle swarm optimization: philosophy and performance differences. In: Porto, V.W., Saravanan, N., Waagen, D., Eiben, A.E. (eds.) *Evolutionary programming*, vol. 7, pp. 601–610. Springer-Verlag, Berlin (1998)
4. Caponetto, R., Fortuna, L., Fazzino, S., Xibilia, M.G.: Chaotic sequences to improve the performance of evolutionary algorithms. *IEEE Trans. Evolut. Comput.* **7**(3), 289–304 (2003)
5. Cassioli, A., Di Lorenzo, D., Locatelli, M., Schoen, F., Sciandrone, M.: *Machine Learning for Global Optimization*. *Comput. Optim. Appl.* doi:10.1007/s10589-010-9330-x accepted August (2010)
6. Chambers, J.M., Cleveland, W.S., Kleiner, B., Tukey, P.A.: *Graphical Models for Data Analysis*. Chapman & Hall, London (1983)
7. Chellapilla, K.: Combining mutation operators in evolutionary programming. *IEEE Trans. Evolut. Comput.* **2**, 91–96 (1998)
8. Cutello, V., Narzisi, G., Nicosia, G., Pavone, M.: An immunological algorithm for global numerical optimization. In: *Proceedings of the of the Seventh International Conference on Artificial Evolution (EA'05)*, vol. 3871, 284–295. LNCS (2005)
9. Cutello, V., Narzisi, G., Nicosia, G., Pavone, M.: Clonal selection algorithms: a comparative case study using effective mutation potentials. In: *Proceedings of the Fourth International Conference on Artificial Immune Systems (ICARIS'05)*, vol. 3627, pp. 13–28. LNCS (2005)
10. Cutello, V., Nicosia, G., Pavone, M.: A hybrid immune algorithm with information gain for the graph coloring problem. In: *Proceedings of Genetic and Evolutionary Computation Conference (GECCO'03)*, vol. 2723, pp. 171–182. LNCS (2003)
11. Cutello, V., Nicosia, G., Pavone, M.: Exploring the capability of immune algorithms: a characterization of hypermutation operators. In: *Proceedings of the Third International Conference on Artificial Immune Systems (ICARIS'04)*, vol. 3239, pp. 263–276. LNCS (2004)
12. Cutello, V., Nicosia, G., Pavone, M.: An immune algorithm with hyper-macromutations for the Dill's 2D hydrophobic–hydrophilic model. In: *Proceedings of Congress on Evolutionary Computation (CEC'04)*, vol. 1, pp. 1074–1080. IEEE Press, New York (2004)
13. Cutello, V., Nicosia, G., Pavone, M.: An immune algorithm with stochastic aging and Kullback entropy for the chromatic number problem. *J. Comb. Optim.* **14**(1), 9–33 (2007)
14. Cutello, V., Nicosia, G., Pavone, M., Narzisi, G.: Real coded clonal selection algorithm for unconstrained global numerical optimization using a hybrid inversely proportional hypermutation operator. In:

- Proceedings of the 21st Annual ACM Symposium on Applied Computing (SAC'06), vol. 2, pp. 950–954 (2006)
15. Cutello, V., Nicosia, G., Pavone, M., Timmis, J.: An immune algorithm for protein structure prediction on lattice models. *IEEE Trans. Evolut. Comput.* **11**(1), 101–117 (2007)
 16. Dasgupta, D.: Advances in artificial immune systems. *IEEE Comput. Intell. Mag.* 40–49 (2006)
 17. Dasgupta, D., Niño, F.: *Immunological Computation: Theory and Applications*. CRC Press, Taylor & Francis Group, Boca Raton (2009)
 18. Davies, M., Secker, A., Freitas, A., Timmis, J., Clark, E., Flower, D.: Alignment-independent techniques for protein classification. *Curr. Proteomics* **5**(4), 217–223 (2008)
 19. De Castro, L.N., Von Zuben, F.J.: Learning and optimization using the clonal selection principle. *IEEE Trans. Evolut. Comput.* **6**(3), 239–251 (2002)
 20. Feo, T.A., Resende, M.G.C., Smith, S.H.: A greedy randomized adaptive search procedure for maximum independent set. *Oper. Res.* **42**, 860–878 (1994)
 21. Finkel, D.E.: DIRECT optimization algorithm user guide. Technical report, CRSC N.C. State University. <http://ftp.ncsu.edu/pub/ncsu/crsc/pdf/crsc-tr03-11.pdf> (March 2003)
 22. Floudas, C.A., Pardalos, P.M. (eds.): *Encyclopedia of Optimization*. Springer, Berlin (2009)
 23. Garrett, S.: How do we evaluate artificial immune systems? *Evolut. Comput.* **13**(2), 145–178 (2005)
 24. Goldberg, D.E.: *The Design of Innovation Lessons from and for Competent Genetic Algorithms*, vol. 7. Kluwer Academic Publisher, Boston (2002)
 25. Goldberg, D.E., Voessner, S.: Optimizing global-local search hybrids. In: *Genetic and Evolutionary Computation Conference (GECCO'99)*, pp. 220–228 (1999)
 26. Hart, W.E., Krasnogor, N., Smith, J.E.: *Recent Advances in Memetic Algorithms*, Series in Studies in Fuzziness and Soft Computing. Springer, Berlin (2005)
 27. <http://www2.research.att.com/~mgrc/ttplots/>
 28. Jones, D.R., Perttunen, C.D., Stuckman, B.E.: Lipschitzian optimization without the Lipschitz constant. *J. Optim. Theory Appl.* **79**, 157–181 (1993)
 29. Karaboga, D., Baturk, B.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *J. Global Optim.* **39**, 459–471 (2007)
 30. Lozano, M., Herrera, F., Krasnogor, N., Molina, D.: Real-coded Memetic algorithms with crossover hill-climbing. *Evolut. Comput.* **12**(3), 273–302 (2004)
 31. Mezura-Montes, E., Velázquez-Reyes, J., Coello Coello C.: A comparative study of differential evolution variants for global optimization. In: *Genetic and Evolutionary Computation Conference (GECCO'06)*, vol. 1, pp. 485–492 (2006)
 32. Noman N., Iba H.: Enhancing differential evolution performance with local search for high dimensional function optimization. In: *Genetic and Evolutionary Computation Conference (GECCO'05)*, pp. 967–974 (2005)
 33. Pardalos, P.M., Resende, M.: *Handbook of Applied Optimization*. Oxford University Press, Oxford (2002)
 34. Price, K.V., Storn, M., Lampien, J.A.: *Differential Evolution: A Practical Approach to Global Optimization*. Springer, Berlin (2005)
 35. Smith, S., Timmis, J.: Immune network inspired evolutionary algorithm for the diagnosis of Parkinsons disease. *Biosystems* **94**(1–2), 34–46 (2008)
 36. Storn, R., Price, K.V.: Differential evolution a simple and efficient heuristic for global optimization over continuous spaces. *J. Global Optim.* **11**(4), 341–359 (1997)
 37. Timmis, J.: Artificial immune systems—today and tomorrow. *Nat. Comput.* **6**(1), 1–18 (2007)
 38. Timmis, J., Hart, E.: Application areas of AIS: the past, present and the future. *J. Appl. Soft Comput.* **8**(1), 191–201 (2008)
 39. Timmis, J., Hart, E., Hone, A., Neal, M., Robins, A., Stepney, S., Tyrrell A.: Immuno-engineering. In: *Proceedings of the international conference on Biologically Inspired Collaborative Computing (IFIP'09)*, vol. 268, pp. 3–17. IEEE Press, New York (2008)
 40. Timmis, J., Kelsey J.: Immune inspired somatic contiguous hypermutation for function optimization. In: *Proceedings of Genetic and Evolutionary Computation Conference (GECCO'03)*, vol. 2723, pp. 207–218. LNCS (2003)
 41. Vaz, A.I.F., Vicente, L.N.: A particle swarm pattern search method for bound constrained global optimization. *J. Global Optim.* **39**, 197–219 (2007)
 42. Versterstrøm, J., Thomsen, R.: A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In: *Congress on Evolutionary Computing (CEC'04)*, vol. 1, pp. 1980–1987 (2004)
 43. Yao, X., Liu, Y., Lin, G.M.: Evolutionary programming made faster. *IEEE Trans. Evolut. Comput.* **3**(2), 82–102 (1999)