

An Immunological Algorithm for Global Numerical Optimization

Vincenzo Cutello, Giuseppe Narzisi, Giuseppe Nicosia, and Mario Pavone

Department of Mathematics and Computer Science,
University of Catania,
V.le A. Doria 6, Catania 95125, Italy
{vctl, narzisi, nicosia, mpavone}@dmi.unict.it

Abstract. Numerical optimization of given objective functions is a crucial task in many real-life problems. The present article introduces an immunological algorithm for continuous global optimization problems, called OPT-IA. Several biologically inspired algorithms have been designed during the last few years and have shown to have very good performance on standard test bed for numerical optimization.

In this paper we assess and evaluate the performance of OPT-IA, FEP, IFEP, DIRECT, CEP, PSO, and EO with respect to their general applicability as numerical optimization algorithms. The experimental protocol has been performed on a suite of 23 widely used benchmarks problems. The experimental results show that OPT-IA is a suitable numerical optimization technique that, in terms of accuracy, generally outperforms the other algorithms analyzed in this comparative study. The OPT-IA is also shown to be able to solve large-scale problems.

Keywords: Artificial Immune Systems, Clonal Selection Algorithms, Immune Algorithm, Aging operator, Global Numerical Optimization.

1 The Immunological Algorithm

Clonal Selection Algorithms (CSAs) [1] are a special class of Immune algorithms (IAs) [1, 2] which are inspired by the Clonal Selection Principle [3] of the human immune system to produce effective methods for search and optimization. In this research paper an immune algorithm inspired by the Clonal Selection Principle, OPT-IA [4, 5, 6, 7, 8], is applied to global numerical optimization.

The OPT-IA algorithm uses a population of candidate solutions, i.e. points of the search space (B cell or B cell receptor according to immunological terminology). At each time step t , we have a population $P_d^{(t)}$ of size d . The initial population of candidate solutions, time $t = 0$, is generated uniformly at random in the relative domains of each function (see table 1) The function $Evaluate(P)$ computes the fitness function value of each B cell $\mathbf{x} \in P$. The implemented IA uses three immune operators, cloning, hypermutation and aging. The cloning operator, simply, clones each B cell dup times producing an intermediate population $P_{N_c}^{(clo)}$ of size $d \times dup = N_c$.

The hypermutation operator acts on the B cell receptor of $P_{N_c}^{(clo)}$. The number of mutations M is determined by *mutation potential*. Our IA uses an *Inversely Proportional Hypermutation* operator, where the number of mutations is inversely proportional to the fitness value, that is, it decreases as the fitness function of the current B cell increases. Two different mutation potential are used, they are defined by the following equations:

$$\alpha = e^{(-\rho * f)}, \quad \alpha = \left(\frac{1}{\rho}\right) e^{(-f)} \tag{1}$$

where α represents the mutation rate, and f is the fitness function value normalized in $[0, 1]$. The number of mutations of a clone with fitness function value f is equal to $\lfloor L * \alpha \rfloor$ where L is the length of the clone receptor, that is $L = l \times n$, with l being the number of bits used to code each variable and n the dimension of the function. The first potential mutation was proposed in [10], while the second potential mutation was introduced in [11]. Figure 1 shows the pseudo-code of the proposed Immune Algorithm.

```

opt-IA( $d, dup, \rho, \tau_B, T_{MAX}$ )
1.  $FFE \leftarrow 0$ ;
2.  $N_c \leftarrow d \times dup$ ;
3.  $t \leftarrow 0$ ;
4.  $P_d^{(t)} \leftarrow \text{Population\_Initialization}(d)$ ;
5. Evaluate( $P_d^{(t)}$ );
6.  $FFE \leftarrow FFE + d$ ;
7. while ( $FFE < T_{MAX}$ ) do
8.    $P_{N_c}^{(clo)} \leftarrow \text{Cloning}(P_d^{(t)}, dup)$ ;
9.    $P_{N_c}^{(hyp)} \leftarrow \text{Hypermutation}(P_{N_c}^{(clo)}, \rho)$ ;
10.  Evaluate( $P_{N_c}^{(hyp)}$ );
11.   $FFE \leftarrow FFE + N_c$ ;
12.  ( ${}^a P_d^{(t)}, {}^a P_{N_c}^{(hyp)}$ ) = Aging( $P_d^{(t)}, P_{N_c}^{(hyp)}, \tau_B$ );
13.   $P_d^{(t+1)} \leftarrow (\mu + \lambda)\text{-Selection}({}^a P_d^{(t)}, {}^a P_{N_c}^{(hyp)})$ ;
14.   $t \leftarrow t + 1$ ;
15. end\_while
    
```

Fig. 1. Pseudo-code of OPT-IA

Aging Operator. By inspecting the pseudo-code of OPT-IA, one can see that an “aging” operator is used. The aging operator eliminates old B cells, in the populations $P_d^{(t)}$ and $P_{N_c}^{(hyp)}$ to avoid premature convergence and to increase diversity in the current population. This operator is the main difference between our algorithm and the other IAs and Evolutionary Algorithms. The parameter τ_B sets the maximum number of generations B cells are allowed to remain in the population. When a B cell is $\tau_B + 1$ old it is erased from the current population, no matter what its fitness value is. During the cloning expansion, a cloned B cell takes the age of its parent. After the hypermutation phase, a cloned B cell which successfully mutates, will be considered to have age equal to 0. In this way, new B cells are given an equal opportunity to effectively explore the given computational landscape. The best B cells which “survived” the aging operator,

are selected from the populations ${}^a P_d^{(t)}$ and ${}^a P_{N_c}^{(hyp)}$. In this way, we obtain the new population $P_d^{(t+1)}$, of d B cells, for the next generation $t + 1$. If $d' < d$ B cells survived, the $(\mu + \lambda)$ -Selection operator creates $d - d'$ new B cells (*Birth phase*).

The evolution cycle ends if a maximum number of Fitness Function Evaluations (FFE) is reached.

2 Numerical Optimization

Numerical optimization problems are fundamental for every field of engineering and science. The task is that of finding global optima of a generic objective function. However, often, the objective function is difficult to optimize because of numerous local optima. Moreover, this difficulty increases proportionally with the problem dimension.

In this paper we consider the following numerical minimization problem:

$$\min(f(\mathbf{x})), \quad \mathbf{B}_l \leq \mathbf{x} \leq \mathbf{B}_u \quad (2)$$

where $\mathbf{x} = (x_1, x_2, \dots, x_n)$ is the variable vector in \mathcal{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 bound of the variables, such that $x_i \in [B_{l_i}, B_{u_i}]$.

We used binary string representation: real values x_i are coded using bitstrings of length $l = 32$. The mapping from the binary string $\mathbf{b} = \langle b_1, b_2, \dots, b_l \rangle$ into a real number x consists of two steps:

- (i) convert the bitstring $\mathbf{b} = \langle b_1, b_2, \dots, b_l \rangle$ from base 2 to base 10 using the equation: $\sum_{i=1}^l b_i * 2^i = x'$;
- (ii) finding the corresponding real value: $x = B_{l_i} + \frac{x'(B_{u_i} - B_{l_i})}{2^l - 1}$, where B_{l_i} and B_{u_i} are the lower and upper bounds of the variables.

Test Functions. We selected twentythree functions from three categories [12]. This relative large set is necessary in order to reduce biases in evaluating algorithms. Table 1 lists the 23 functions and their key properties (for a complete description of all the functions and the parameters involved see [12]). These functions can be divided into three categories of different complexities:

- unimodal functions ($f_1 - f_7$), which are relatively easy to optimize, but the difficulty increases as the problem dimension increases;
- multimodal functions ($f_8 - f_{13}$), with many local minima, they represent the most difficult class of problems for many optimization algorithms;
- multimodal functions which contain only few local optima ($f_{14} - f_{23}$).

Some functions possess unique features: f_6 is a discontinuous step function having a single optimum; f_7 is a noisy quartic function involving a uniformly distributed random variable within $[0, 1)$. Optimizing unimodal functions is not a major issue, so in this case the convergence rate is of main interest. However, for multimodal functions the quality of the final results is more important since it reflects the algorithm's ability in *escaping* from local optima.

Table 1. The 23 benchmark functions used in our experimental study; n is the dimension of the function; f_{min} is the minimum value of the function; $S \subseteq \mathcal{R}^n$ are the variable bounds (for a complete description of all the functions and the parameters involved see [12])

Test function	n	S	f_{min}
$f_1(\mathbf{x}) = \sum_{i=1}^n x_i^2$	30	$[-100, 100]^n$	0
$f_2(\mathbf{x}) = \sum_{i=1}^n x_i + \prod_{i=1}^n x_i $	30	$[-10, 10]^n$	0
$f_3(\mathbf{x}) = \sum_{i=1}^n \left(\sum_{j=1}^i x_j \right)$	30	$[-100, 100]^n$	0
$f_4(\mathbf{x}) = \max_i \{ x_i , 1 \leq i \leq n\}$	30	$[-100, 100]^n$	0
$f_5(\mathbf{x}) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	30	$[-30, 30]^n$	0
$f_6(\mathbf{x}) = \sum_{i=1}^n (x_i + 0.5)^2$	30	$[-100, 100]^n$	0
$f_7(\mathbf{x}) = \sum_{i=1}^n ix_i^4 + random[0, 1]$	30	$[-1.28, 1.28]^n$	0
$f_8(\mathbf{x}) = \sum_{i=1}^n -x_i \sin(\sqrt{ x_i })$	30	$[-500, 500]^n$	-12569.5
$f_9(\mathbf{x}) = \sum_{i=1}^n [x_i^2 - 10 \cos(2\pi x_i) + 10]$	30	$[-5.12, 5.12]^n$	0
$f_{10}(\mathbf{x}) = -20 \exp\left(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2}\right) - \exp\left(\frac{1}{n} \sum_{i=1}^n \cos 2\pi x_i\right) + 20 + e$	30	$[-32, 32]^n$	0
$f_{11}(\mathbf{x}) = \frac{1}{4000} \sum_{i=1}^n x_i^2 - \prod_{i=1}^n \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	30	$[-600, 600]^n$	0
$f_{12}(\mathbf{x}) = \frac{\pi}{4} \{10 \sin^2(\pi y_1) + \sum_{i=1}^{n-1} (y_i - 1)^2 [1 + 10 \sin^2(\pi y_{i+1})] + (y_n - 1)^2\} + \sum_{i=1}^n u(x_i, 10, 100, 4),$ $y_i = 1 + \frac{1}{4}(x_i + 1)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & \text{if } x_i > a, \\ 0, & \text{if } -a \leq x_i \leq a, \\ k(-x_i - a)^m, & \text{if } x_i < -a. \end{cases}$	30	$[-50, 50]^n$	0
$f_{13}(\mathbf{x}) = 0.1 \{\sin^2(3\pi x_1) + \sum_{i=1}^{n-1} (x_i - 1)^2 [1 + \sin^2(3\pi x_{i+1})] + (x_n - 1) [1 + \sin^2(2\pi x_n)]\} + \sum_{i=1}^n u(x_i, 5, 100, 4)$	30	$[-50, 50]^n$	0
$f_{14}(\mathbf{x}) = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^j (x_i - a_{ij})^6} \right]$	2	$[-65.536, 65.536]^n$	1
$f_{15}(\mathbf{x}) = \sum_{i=1}^{11} \left[a_i - \frac{x_i(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4} \right]^2$	4	$[-5, 5]^n$	0.0003075
$f_{16}(\mathbf{x}) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	2	$[-5, 5]^n$	-1.0316285
$f_{17}(\mathbf{x}) = \left(x_2 - \frac{5.1}{4\pi^2} x_1^2 + \frac{5}{\pi} x_1 - 6 \right)^2 + 10 \left(1 - \frac{1}{8\pi} \right) \cos x_1 + 10$	2	$[-5, 10] \times [0, 15]$	0.398
$f_{18}(\mathbf{x}) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	2	$[-2, 2]^n$	3
$f_{19}(\mathbf{x}) = -\sum_{i=1}^4 c_i \exp\left[-\sum_{j=1}^4 a_{ij}(x_j - p_{ij})^2\right]$	4	$[0, 1]^n$	-3.86
$f_{20}(\mathbf{x}) = -\sum_{i=1}^4 c_i \exp\left[-\sum_{j=1}^6 a_{ij}(x_j - p_{ij})^2\right]$	6	$[0, 1]^n$	-3.32
$f_{21}(\mathbf{x}) = -\sum_{i=1}^5 \left[(\mathbf{x} - a_i)(\mathbf{x} - a_i)^T + c_i \right]^{-1}$	4	$[0, 10]^n$	-10.1422
$f_{22}(\mathbf{x}) = -\sum_{i=1}^7 \left[(\mathbf{x} - a_i)(\mathbf{x} - a_i)^T + c_i \right]^{-1}$	4	$[0, 10]^n$	-10.3909
$f_{23}(\mathbf{x}) = -\sum_{i=1}^{10} \left[(\mathbf{x} - a_i)(\mathbf{x} - a_i)^T + c_i \right]^{-1}$	4	$[0, 10]^n$	-10.53

3 Experimental Results

3.1 Experimental Setup

The performance of the proposed IA is assessed by carrying out optimization on the 23 functions listed in table 1 and comparing results with some well-known algorithms for global optimization. For each test function 50 independent runs are performed. At each generation we compute the mean value of the best fit individuals for all 50 runs in order to plot the evolution curves. Moreover the

Table 2. Parameters used by OPT-IA for each function (f_1, \dots, f_{23})

Function	$\alpha = e^{(-\rho * f)}$				$\alpha = \left(\frac{1}{\rho}\right) e^{(-f)}$			
	d	dup	τ_B	ρ	d	dup	τ_B	ρ
f_1	10	2	5	10	10	2	10	150
f_2	10	2	10	10	10	2	10	150
f_3	20	2	20	10	20	2	10	150
f_4	10	2	10	10	20	2	20	150
f_5	10	2	10	10	20	2	20	150
f_6	20	2	20	10	20	2	50	150
f_7	10	2	10	10	20	2	20	150
f_8	20	2	20	10	20	2	20	150
f_9	20	2	20	10	20	2	5	150
f_{10}	20	2	20	10	10	2	10	150
f_{11}	20	2	20	10	10	2	10	150
f_{12}	20	2	20	10	10	2	10	150
f_{13}	20	2	20	10	20	2	5	150
f_{14}	10	5	5	10	20	2	20	150
f_{15}	20	2	20	10	20	2	20	150
f_{16}	10	2	5	6	10	2	20	100
f_{17}	10	2	15	7	10	2	15	125
f_{18}	10	2	10	8	10	2	15	100
f_{19}	10	2	10	9	10	2	15	100
f_{20}	10	2	10	8	20	2	20	150
f_{21}	10	2	25	6	10	2	10	150
f_{22}	10	2	5	7	10	2	15	125
f_{23}	10	2	5	7	10	2	10	100

standard deviation is used to indicate the consistency of the algorithm. Table 2 summarizes the key parameters setting of OPT-IA for each test function and for each mutation potential used.

There are several algorithms designed for numerical optimization. We start comparing OPT-IA with one of the best evolutionary algorithms for numerical optimization in literature: Fast Evolutionary Programming (FEP) [12] and his improved version IFEP. FEP is based on Conventional Evolutionary Programming (CEP [13]) but uses a new mutation operator based on Cauchy random numbers that helps the algorithm to escape from local optima. The performance of OPT-IA is further compared with some other well-established evolutionary algorithms such as CEP with three different mutation operators (Gaussian Mutation Operator GMO, Cauchy Mutation Operator CMO and Mean Mutation Operator MMO) [13], Particle Swarm Optimization (PSO) [14] and Evolutionary Optimization (EO) [14]. Moreover comparison is made, when possible, with a global search algorithm for bound constrained optimization based on Lipschitz constant estimation, DIRECT [15, 16].

3.2 Comparison with FEP and DIRECT

Unimodal functions ($f_1 - f_7$). Unimodal functions are not the most challenging test problems. There are more efficient algorithms which are specifically designed to optimize them. The aim in this case is to get a picture of the convergence rate of the algorithms. In table 3 we report the optimization results obtained by OPT-IA with respect to those obtained by FEP and DIRECT. All results

Table 3. Comparison between FEP, OPT-IA and DIRECT on functions $f_1 - f_7$

Fun.	T_{max}	Direct	FEP	opt-IA	
				$\alpha = e^{(-\rho * f)}$	$\alpha = \left(\frac{1}{\rho}\right) e^{(-f)}$
		Min Found	Mean Best Std Dev	Mean Best Std Dev	Mean Best Std Dev
f_1	150,000	n.a.	5.7×10^{-4} 1.3×10^{-4}	9.23×10^{-12} 2.44×10^{-11}	1.7×10^{-8} 3.5×10^{-15}
f_2	200,000	n.a.	8.1×10^{-3} 7.7×10^{-4}	0.0 0.0	7.1×10^{-8} 0.0
f_3	500,000	n.a.	1.6×10^{-2} 1.4×10^{-2}	0.0 0.0	1.9×10^{-10} 2.63×10^{-10}
f_4	500,000	n.a.	0.3 0.5	1.0×10^{-2} 5.3×10^{-3}	4.1×10^{-2} 5.3×10^{-2}
f_5	2×10^6	27.89	5.06 5.87	3.02 12.2	28.4 0.42
f_6	150,000	n.a.	0.0 0.0	0.2 0.44	0.0 0.0
f_7	300,000	8.9×10^{-3}	7.6×10^{-3} 2.6×10^{-3}	3.0×10^{-3} 1.2×10^{-3}	3.9×10^{-3} 1.3×10^{-3}

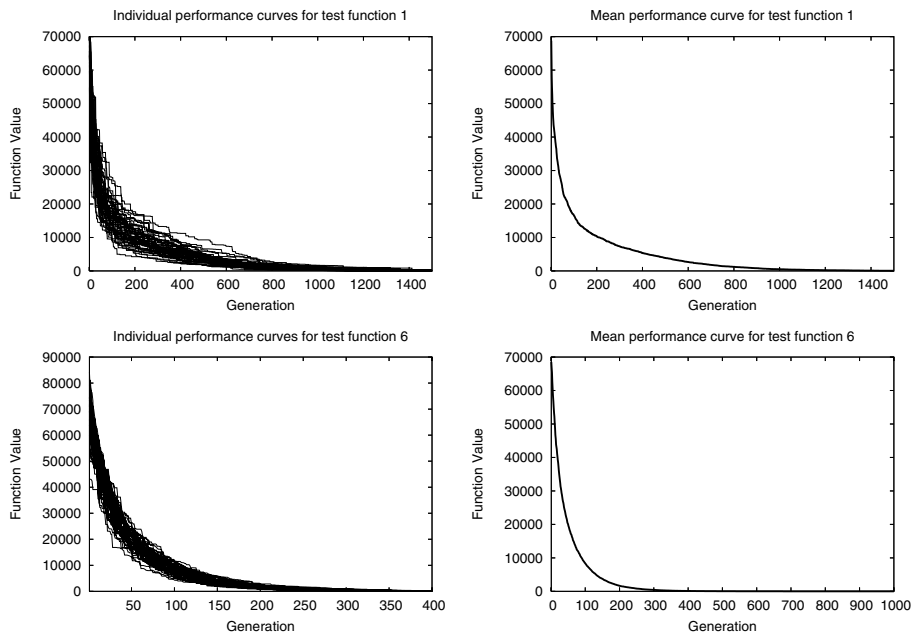


Fig. 2. Evolution curves of OPT-IA algorithm on two unimodal functions (f_1, f_6) over 50 independent runs Individual curves for each run (left plot), mean performance curve (right plot)

have been averaged over 50 independent runs. “Mean Best” indicates the mean best function values found in the last generation, “Std Dev” stands for standard deviation. Second column shows the maximum number of Fitness Function Evaluation allowed (T_{max}); we used the same T_{max} values proposed by X. Yao, Y. Liu

Table 4. Comparison between FEP, OPT-IA and DIRECT on functions $f_8 - f_{13}$

Fun.	T_{max}	Direct	FEP	opt-IA	
				$\alpha = e^{(-\rho * f)}$	$\alpha = \left(\frac{1}{\rho}\right) e^{(-f)}$
		Min Found	Mean Best Std Dev	Mean Best Std Dev	Mean Best Std Dev
f_8	900,000	-4093.0.	-12554.5 52.6	-12508.38 155.54	-12568.27 0.23
f_9	500,000	n.a.	4.6×10^{-2} 1.2×10^{-2}	19.98 7.66	2.66 2.39
f_{10}	150,000	n.a.	1.8×10^{-2} 2.1×10^{-3}	18.98 0.35	1.1×10^{-4} 3.1×10^{-5}
f_{11}	200,000	n.a.	1.6×10^{-2} 2.2×10^{-2}	7.7×10^{-2} 8.63×10^{-2}	4.55×10^{-2} 4.46×10^{-2}
f_{12}	150,000	0.03	9.2×10^{-6} 3.6×10^{-6}	0.137 0.23	3.1×10^{-2} 5.7×10^{-2}
f_{13}	150,000	0.96	1.6×10^{-4} 7.3×10^{-5}	1.51 0.10	3.20 0.13

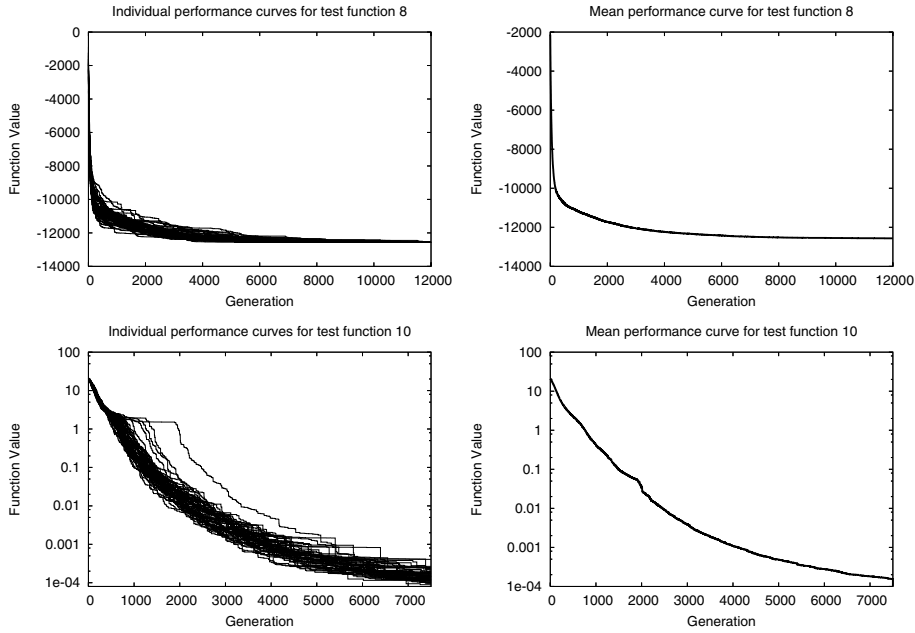


Fig. 3. Evolution curves of OPT-IA algorithm on two multimodal functions with many local minima (f_8, f_{10}) over 50 independent runs. Individual curves for each run (left plot), mean performance curve (right plot).

and G. Lin in [12]. Better results are highlighted in boldface. As it can be seen, OPT-IA is able to always obtain better results than FEP except for function f_3 . For most functions, results for DIRECT are not available because the symmetry of the function implies that the optimum is in the center of the variable bounds,

Table 5. Comparison between FEP, OPT-IA and DIRECT on functions $f_{14} - f_{23}$

Fun.	T_{max}	Direct	FEP	opt-IA	
				$\alpha = e^{(-\rho * f)}$	$\alpha = \left(\frac{1}{\rho}\right) e^{(-f)}$
		Min Found	Mean Best Std Dev	Mean Best Std Dev	Mean Best Std Dev
f_{14}	10,000	1.0.	1.22 .56	1.02 7.1×10^{-2}	1.21 0.54
f_{15}	400,000	1.2×10^{-3}	5.0×10^{-4} 3.2×10^{-4}	7.1×10^{-4} 1.3×10^{-4}	7.7×10^{-3} 1.4×10^{-2}
f_{16}	10,000	-1.031	-1.031 4.9×10^{-7}	-1.03158 1.5×10^{-4}	-1.02 1.1×10^{-2}
f_{17}	10,000	0.398	0.398 1.5×10^{-7}	0.398 2.0×10^{-4}	0.450 0.21
f_{18}	10,000	3.01	3.02 0.11	3.0 0.0	3.0 0.0
f_{19}	10,000	-3.86	-3.86 1.4×10^{-5}	-3.72 1.1×10^{-4}	-3.72 1.1×10^{-2}
f_{20}	20,000	-3.30	-3.27 5.9×10^{-2}	-3.31 7.4×10^{-2}	-3.31 5.9×10^{-3}
f_{21}	10,000	-6.84	-5.52 1.59	-9.11 1.82	-5.36 2.20
f_{22}	10,000	-7.09	-5.52 2.12	-9.86 1.88	-5.34 2.11
f_{23}	10,000	-7.22	-6.57 3.14	-9.96 1.46	-6.03 2.66

the point from which DIRECT starts the search. Figure 2 shows performance curves of OPT-IA for the two unimodal functions f_1 and f_6 .

Multimodal functions with many local minima ($f_8 - f_{13}$). Function $f_8 - f_{13}$ are multimodal function with many local minima. The number of local minima increases exponentially as the function dimension increases. The fitness landscape of these functions is generally very *rugged* and difficult to optimize. Table 4 summarizes the final results obtained by OPT-IA, FEP and DIRECT. FEP has a better performance on 4 of 6 test problems. In particular, for functions f_9, f_{12} and f_{13} FEP perform significantly better than OPT-IA, except for function f_{11} where results are comparable. On the other hand, OPT-IA performs significantly better than FEP on function f_8 and f_{10} . By comparing results between OPT-IA and DIRECT, we can see that DIRECT is unable to approach the optimum, while on functions f_{12} and f_{13} DIRECT shows a bit better performance.

Multimodal functions with only a few local minima ($f_{14} - f_{23}$). The final results of OPT-IA, FEP and DIRECT on functions $f_{14} - f_{23}$ are summarized in table 5. In this case OPT-IA shows a better performance on 6 of the 10 test functions. Moreover, for functions f_{16} and f_{17} , OPT-IA is able to obtain the same optimum results of FEP in terms of mean best values found, but FEP shows more consistency in terms of standard deviation. Instead for function f_{15} results are of the same order. Inspecting all the experimental results (reported in the previous tables) it is possible to note an overall better performance of OPT-IA algorithm using a minimum population size ($d \in \{10, 20\}$) while FEP uses a larger population size greater of an order of magnitude.

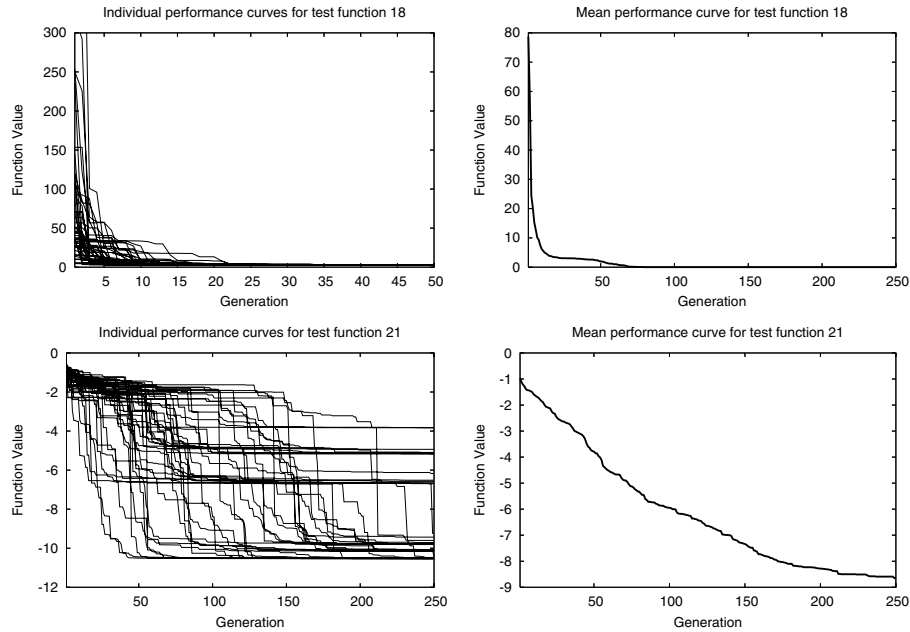


Fig. 4. Evolution curves of OPT-IA algorithm on two multimodal functions with a few local minima (f_{18} , f_{21}) over 50 independent runs. Individual curves for each run (left plot), mean performance curve (right plot).

3.3 Comparison with IFEP

The analyses performed in [12] show that Cauchy mutation performs better when the current search point is far away from the global optimum, while Gaussian mutation is better when search points are in the neighborhood of the global optimum. Based on those observations in [12] was proposed a version of FEP which uses both Cauchy and Gaussian mutations. This improved version was called IFEP. IFEP differs from FEP only in the step of the creation of the offsprings: two new offspring are generated instead of one, the first one using Cauchy mutation and the second one using Gaussian mutation, the better one is chosen. Table 6 shows the comparison between OPT-IA and IFEP on the same functions used in [12]: unimodal functions f_1, f_2 , multimodal functions f_{10}, f_{11} with many local minima and multimodal functions f_{21}, f_{22} with only few local minima. OPT-IA has again a better performance on all the functions except for function f_{11} where the results are comparable.

3.4 Comparison with CEP, PSO and EO

Finally, we compare the immune algorithm with some other well-known biologically inspired algorithms: CEP, PSO and EO. Since the optimization results obtained with these algorithms are available in literature only for some of the

Table 6. Comparison between OPT-IA and IFEP on functions $f_1, f_2, f_{10}, f_{11}, f_{21}, f_{22}$ and f_{23}

Function	T_{max}	opt-IA	IFEP
f_1	150,000	1.70×10^{-8}	4.16×10^{-5}
f_2	200,000	7.15×10^{-5}	2.44×10^{-2}
f_{10}	150,000	1.11×10^{-4}	4.83×10^{-3}
f_{11}	200,000	8.36×10^{-2}	4.54×10^{-2}
f_{21}	10,000	-8.29	-6.46
f_{22}	10,000	-9.59	-7.10
f_{23}	10,000	-9.96	-7.80

Table 7. Comparison between OPT-IA and CEP with three different mutation operators (GMO,CMO,MMO) in terms of mean best values found

Function	T_{max}	opt-IA	CEP		
			GMO	CMO	MMO
f_1	150,000	1.70×10^{-8}	3.09×10^{-7}	3.07×10^{-7}	9.81×10^{-7}
f_2	250,000	7.15×10^{-5}	1.99×10^{-3}	5.87×10^{-3}	3.23×10^{-3}
f_3	250,000	260.12	17.60	5.78	11.80
f_4	250,000	0.001	5.18	0.66	1.88
f_5	250,000	29	86.70	114.0	63.8
f_7	250,000	5.85×10^{-03}	12.20	9.42	9.53
f_9	250,000	24.0	120.0	4.73	9.52
f_{10}	150,000	1.11×10^{-4}	9.10	1.3×10^{-3}	7.49×10^{-4}
f_{11}	250,000	8.36×10^{-2}	2.52×10^{-7}	2.2×10^{-6}	6.99×10^{-7}

Table 8. Performance Comparison among OPT-IA, PSO, and EO on functions f_1, f_5, f_9, f_{11}

Fun.	T_{max}	opt-IA		PSO		EO	
		Mean Best	Std Dev	Mean Best	Std Dev	Mean Best	Std Dev
f_1	250,000	1.70×10^{-8}	3.5×10^{-15}	11.75	1.3208	9.8808	0.9444
f_5	250,000	29.0	0.0	1911.598	374.2935	1610.39	293.5783
f_9	250,000	24.0	7.69	47.1354	1.8782	46.4689	2.4545
f_{11}	250,000	8.36×10^{-2}	4.32×10^{-2}	0.4498	0.0566	0.4033	0.0436

23 test functions considered in this work, comparisons will be made accordingly. Results shown in tables 7 and 8 indicate that while OPT-IA maintain a consistent performance, CEP, PSO and EO appear unable to reach the optima for most of the listed functions, although they make a considerable computational effort. An overall better performance of OPT-IA is evident.

4 Conclusions

In this paper we have introduced an immune algorithm based on clonal selection principle, called OPT-IA, for global numerical optimization. The main features of OPT-IA are the following: cloning operator, inversely proportional hypermutation operator, and aging operator. The cloning operator explores the neighborhood of each point of the search space. The inversely proportional

hypermutation perturbs each candidate solution inversely proportional to its fitness function value. Finally, the aging operator eliminates the oldest candidate solutions from the current population in order to introduce diversity and to avoid local minima during the evolutionary search process. We tested OPT-IA on 23 well-known benchmark problems. The experimental studies show that the clonal selection algorithm is an effective numerical optimization algorithm in terms of solution quality. The results show that OPT-IA is significant better than the seven tested evolutionary algorithms and one well-known deterministic algorithm (DIRECT).

As future works we plan to perform a deeper statistical analysis of the obtained experimental results and to compare *opt-IA* with the BCA algorithm [17], Evolution Strategies and Differential Evolution [18].

References

- [1] Nicosia G.: “Immune Algorithms for Optimization and Protein Structure Prediction”, Ph.D. Thesis, *University of Catania*, Italy, December 2004.
- [2] De Castro L. N., Timmis J., “Artificial Immune Systems: A New Computational Intelligence Paradigm” London, UK: *Springer-Verlag*, (2002).
- [3] Cutello V., Nicosia G.: “The Clonal Selection Principle for in silico and in vitro Computing”, in *Recent Developments in Biologically Inspired Computing*, L. N. de Castro and F. J. Von Zuben, Eds., (2004).
- [4] Cutello V., Nicosia G., and Pavone M.: “Exploring the capability of immune algorithms: A characterization of hypermutation operators” in *Proc. of the Third Int. Conf. on Artificial Immune Systems (ICARIS'04)*, pp. 263–276 (2004).
- [5] Nicosia G., Cutello V., Pavone M.: “An Immune Algorithm with Hyper-Macromutations for the Dill’s 2D Hydrophobic-Hydrophilic Model”, *Congress on Evolutionary Computation, CEC 2004, IEEE Press*, vol. 1, pp. 1074-1080, (2004).
- [6] Cutello V., Morelli G., Nicosia G., and Pavone M.; “Immune Algorithms with Aging operators for the String Folding Problem and the Protein Folding Problem,” in *Proc. of the Fifth Europ. Conf. on Comp. in Combinatorial Optimization (EVOCOP'05)*, LNCS, vol. 3448, pp. 80-90 (2005).
- [7] Nicosia G., Cutello V., Pavone M.: “A Hybrid Immune Algorithm with Information Gain for the Graph Coloring Problem”, *Genetic and Evolutionary Computation Conference, GECCO 2003*, vol. 2723, pp. 171-182.
- [8] Nicosia G., Cutello V., Bentley P. J., Timmis J., “Artificial Immune Systems”, *Third International Conference, ICARIS 2004, Catania, Italy, September 13-16*, Springer (2004).
- [9] Goldberg D.E.: “The Design of Innovation: Lessons from and for Competent Genetic Algorithms”, *Kluwer Academic Publisher*, vol 7, pp. Boston, (2002).
- [10] De Castro L.N., Von Zuben F.J.: “Learning and optimization using the clonal selection principle”. *IEEE Trans. on Evolutionary Computation*, vol 6, no 3, pp. 239-251, (2002).
- [11] De Castro L. N., Timmis J.:. “An Artificial Immune Network for Multimodal Function Optimization”, *CEC'02, Proceeding of IEEE Congress on Evolutionary Computation*, IEEE Press, (2002).
- [12] Yao X., Liu Y. and Lin G.M.: “Evolutionary programming made faster”, *IEEE Trans. on Evolutionary Computation*, vol 3, pp. 82-102, (1999).

- [13] Chellapilla, K.: "Combining mutation operators in evolutionary programming," *IEEE Trans. Evol. Comput.*, vol. 2, pp. 91-96, (1998).
- [14] Angeline, P. J.: "Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences", in *Proc. Evolutionary Programming VII*, V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, Eds., pp. 601-610, (1998).
- [15] Jones, D. R., Perttunen, C. D. and Stuckman, B. E.: "Lipschitzian optimization without the lipschitz constant." *J. of Optimization Theory and Application*, vol. 79, pp. 157-181, (1993).
- [16] Finkel, D. E.: "DIRECT Optimization Algorithm User Guide." *Technical Report, CRSC N.C. State University*, March 2003. (<ftp://ftp.ncsu.edu/pub/ncsu/crsc/pdf/crsc-tr03-11.pdf>).
- [17] Timmis J. and Kelsey J.: "Immune Inspired Somatic Contiguous Hypermutation for Function Optimisation", *Genetic and Evolutionary Computation Conference, GECCO 2003*, Springer vol. 2723, pp. 207-218.
- [18] Vesterstrom, J. and Thomsen R.: "A Comparative Study of Differential Evolution, Particle Swarm Optimization, and Evolutionary Algorithms on Numerical Benchmark Problems", *Congress on Evolutionary Computation, CEC 2004, IEEE Press*, vol. 2, pp. 1980-1987, (2004).