

Clonal Selection Algorithms: A Comparative Case Study using Effective Mutation Potentials

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

Department of Mathematics and Computer Science
University of Catania
V.le A. Doria 6, 95125 Catania, Italy
{vct1,narzisi,nicosia,mpavone}@dmi.unict.it

Abstract. This paper presents a comparative study of two important Clonal Selection Algorithms (CSAs): CLONALG and opt-IA. To understand deeply the performance of both algorithms, have been faced four important classes of problems: toy problems (ones-counting and trap functions), pattern recognition, numerical optimization problems (23 functions) and NP-Complete problem (the 2D HP model for protein structure prediction problem). Two possible versions of CLONALG have been implemented and tested. The experimental results show a global better performance of opt-IA respect to CLONALG. Considering the results obtained, we can claim that the CSAs represent a new effective class of Evolutionary Algorithms to perform searching, learning and optimization tasks.

Keywords: Clonal Selection Algorithms, CLONALG, opt-IA, ones-counting, trap functions, pattern recognition, numerical optimization, NP-Complete problems, 2D HP Protein Structure Prediction.

1 Introduction

Clonal Selection Algorithms (CSAs) are a special class of Immune algorithms (IA) which are inspired by the Clonal Selection Principle [1–3] of the human immune system to produce effective methods for search and optimization. In this research paper two well known CSAs are analyzed: CLONal selection ALGORITHM (CLONALG) [4] and optimization Immune Algorithm (opt-IA) [5]. To analyze experimentally the overall performance of those two algorithms, they will be tested on a robust set of problems belonging to four different classes: toy problems, pattern recognition, numerical optimization problems and NP-complete problems. Those algorithms use a simplified model of the Clonal Selection Principle. Both algorithms are population based. Each individual of the population is a candidate solution belonging to the combinatorial fitness landscape of a given computational problem. Using the cloning operator, an immune algorithm produces individuals with higher affinities (higher fitness function values), introducing blind perturbation (by means of a hypermutation operator) and selecting their improved mature progenies.

1.1 CLONALG

CLONALG is characterized by two populations: a population of antigens Ag and a population of antibodies Ab (denoted with $P^{(t)}$). 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 $S, m \in S^L$. The Ab population is the set of current candidate solutions, and the Ag is the environment to be recognized. After a random initialization of the first population $P^{(0)}$, the algorithm loops for a predefined maximum number of generations (N_{gen}). In the first step, it determines the fitness function values of all Abs in relation to the Ag. Next, it selects n Abs that will be cloned independently and proportionally to their antigenic affinities, generating the clone population P^{clo} . Hence, the higher the fitness, the higher the number of clones generated for each of the n Abs. The hypermutation operator performs an affinity maturation process inversely proportional to the fitness values generating the matured clone population P^{hyp} . After computing the antigenic affinity (i.e., the fitness function) of the population P^{hyp} , CLONALG creates randomly d new antibodies that will replace the d lowest fit Abs in the current population.

In this paper we use the CLONALG version for optimization tasks (except for pattern recognition where we will use the other version proposed in [4]), varying the same parameters (N, n, β, d) plus ρ (not studied in [4]) that controls the shape of the mutation rate with respect to the following two equations:

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

where α represents the mutation rate, and f is the fitness function value normalized in $[0,1]$. The number of mutations of the clone with fitness function value f is equal to $\lfloor L * \alpha \rfloor$ where L is the length of the clone receptor. The first potential mutation has been proposed in [4], the original mutation law used by CLONALG; while the second potential mutation has been introduced in [6]. We will show how the mutation rates and the parameter ρ are crucial to set in order to find the better performance of the algorithm. In the optimization version of CLONALG the affinity proportionate cloning is not useful; we use the same law defined in [4]: $N_c = \sum_{i=1}^n \text{round}(\beta * N)$; where N_c represents the total number of clones created at each generation, in this way, each antibody (or B cell) produces the same number of clones. Moreover, we assign $N = n$, so all Abs from the population will be selected for cloning in step 4 of the algorithm. For the pseudo-code of CLONALG see [4].

The experimental study was conducted using two versions of CLONALG, CLONALG₁ and CLONALG₂, with different selection scheme in step 8 of the algorithm and using the two potential mutations above defined (equations 1):

CLONALG₁: each Ab at generation t will be substituted at the next generation ($t + 1$) by the best individual of its set of $\beta * N$ mutated clones.

CLONALG₂: the population at the next generation ($t + 1$) will be formed by the n best Ab's of the mutated clones at time step t .

1.2 opt-IA

The opt-IA algorithm uses only two entities: antigens (Ag) and B cells (or Ab) like CLONALG. At each time step t , we have a population $P^{(t)}$ of size d . The initial population of candidate solutions, time $t = 0$, is generated randomly. 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^{clo} of size $d \times dup$, where each cloned B cell has the same age of its parent.

The hypermutation operator acts on the B cell receptor of P^{clo} . The number of mutations M is determined by *mutation potential*. We tested our IA using inversely proportional hypermutation operators, hypermacromutation operator, and combination of hypermutation operators and hypermacromutation. The two hypermutation operators and the Hypermacromutation perturb the receptors using different mutation potentials, depending upon a parameter c . In particular, the two implemented operators try to mutate each B cell receptor M times without using probability mutation p_m , typically used in Genetic Algorithms.

In the *Inversely Proportional Hypermutation* the number of mutations is inversely proportional to the fitness value, that is it decrease as the fitness function of the current B cell increases. So at each time step t , the operator will perform at most $M_i(f(\mathbf{x})) = ((1 - \frac{E^*}{f(\mathbf{x})}) \times (c \times \ell)) + (c \times \ell)$ mutations, where E^* is the optimum of the problem and ℓ is the string length. In this case, $M_i(f(\mathbf{x}))$ has the shape of an hyperbola branch. In the *Hypermacromutation* the number of mutations is independent from the fitness function f and the parameter c . In this case, we choose at random two sites in the string, i and j such that $(i + 1) \leq j \leq \ell$ the operator mutates at most $M_m(\mathbf{x}) = j - i + 1$ directions, in the range $[i, j]$.

The aging operator eliminates old B cells, in the populations $P^{(t)}$, $P^{(hyp)}$ and/or $P^{(macro)}$, to avoid premature convergence. The parameter τ_B sets the maximum number of generations allowed to B cells to remain in the population. When a B cell is $\tau_B + 1$ old it is erased by the current population, no matter what its fitness value is. We call this strategy, *static pure aging*. We can also define a *stochastic aging* where the elimination process is based on a stochastic law. The probability to remove a B cell is governed by exponential negative law with parameter τ_B , using the function $P_{die}(\tau_B) = (1 - e^{-\ln(2)/\tau_B})$ [2]. 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. Such a scheme intends to give an equal opportunity to each new B cell to effectively explore the landscape. The best B cells which “survived” the aging operator, are selected from the populations $P^{(t)}$, $P^{(hyp)}$ and/or $P^{(macro)}$, in such a way each B cell receptor is *unique*, i.e. each B cell receptor is different from all other receptors. In this way, we obtain the new population $P^{(t+1)}$, of d B cells, for the next generation $t + 1$. If only $d' < d$ B cells survived, the $(\mu + \lambda)$ -*Selection operator* creates $d - d'$ new B cells (*Birth phase*). The boolean function *Termination.Condition()* returns true if a

```

opt-IA( $\ell, d, dup, \tau_B, c, h, hm$ )
1.  $t := 0$ 
2.  $P^{(t)} := \text{Initial\_Pop}()$ 
3. Evaluate( $P^{(0)}$ )
4. while ( $\neg \text{Termination\_Condition}()$ ) do
5.    $P^{(clo)} := \text{Cloning}(P^{(t)}, dup)$ 
6.   if ( $H$  is TRUE) then
7.      $P^{(hyp)} := \text{Hypermutation}(P^{(clo)}, c, \ell)$ 
8.     Evaluate( $P^{(hyp)}$ )
9.   if ( $M$  is TRUE) then
10.     $P^{(macro)} := \text{Hypermacro}(P^{(clo)})$ 
11.    Evaluate ( $P^{(macro)}$ )
12.    Aging( $P^{(t)}, P^{(hyp)}, P^{(macro)}, \tau_B$ )
13.     $P^{(t+1)} := (\mu + \lambda)\text{-Selection}(P^{(t)}, P^{(hyp)}, P^{(macro)})$ 
14.     $t := t + 1$ 
15.end\_while

```

Fig. 1. Pseudo-code of opt-IA.

solution is found, or a maximum number of fitness function evaluations (T_{max}) is reached. Figure 1 shows the pseudo-code of the proposed Immune Algorithm. The boolean variables H, M control, respectively, the hypermutation and the hypermacro-mutation operator.

2 Toy problems

Toy problems play a central role in understanding the dynamics of algorithms [7]. In fact, toy problems can be used to show the main differences between different algorithms. In this section we test and study the dynamic of CLONALG and opt-IA for two classical toy problems: *one-counting* and *trap functions*.

2.1 Ones-Counting problem

The ones-counting problem (or ones-max problem), is simply defined as the problem to maximize the number of 1 of bit-string \mathbf{x} of length ℓ : $f(\mathbf{x}) = \sum_{i=1}^{\ell} x_i$, with $x_i \in \{0, 1\}$. In this work we set $\ell = 100$. The ones-max problem is a classical test to assess if an evolutionary algorithm is able to reach an optimal solution starting from a randomly initialized population.

All the experimental results reported in this sections have been averaged over 100 independent runs, and we fixed the max number of fitness function evaluations (T_{max}) to 10^4 . Figure 2 shows the population average fitness versus generation for CLONALG and opt-IA on the first 100 generations. For CLONALG we show both versions (CLONALG₁ and CLONALG₂) using the two possible mutation rates defined in section 1.1. The convergence speed of CLONALG is

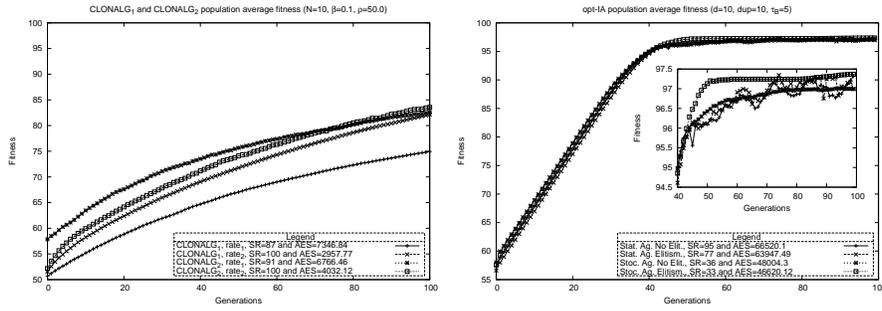


Fig. 2. Population average fitness for CLONALG (left plot) and opt-IA (right plot) (rate₁: $\alpha = e^{(-\rho * f)}$, rate₂: $\alpha = \left(\frac{1}{\rho}\right) e^{(-f)}$).

inferior respect to opt-IA but its SR is superior. In about 40 generations, opt-IA reaches a fitness value of $\simeq 95$ but from now on the aging process is more intensive refraining the convergence speed. For opt-IA we show versions with the usage of the static or stochastic aging coupled with an elitist or no-elitist strategy (i.e., the best candidate solution is always maintained from a generation to another). The better results are obtained using static aging.

2.2 Trap Functions

The trap functions [8] [9], simply, take as input the number of 1's of bit strings of length ℓ :

$$f(x) = \hat{f}(u(x)) = \hat{f}\left(\sum_{k=1}^{\ell} x_k\right) \quad (2)$$

There are two trap functions: *simple trap function* and *complex trap function*. The definition of the simple and complex trap functions are the following:

$$\hat{f}(u) = \begin{cases} \frac{a}{z}(z-u), & \text{if } u \leq z \\ \frac{b}{\ell-z}(u-z), & \text{otherwise.} \end{cases}, \quad \hat{f}(u) = \begin{cases} \frac{a}{z_1}(z_1-u), & \text{if } u \leq z_1 \\ \frac{b}{\ell-z_1}(u-z_1), & \text{if } z_1 < u \leq z_2 \\ \frac{b(z_2-z_1)}{\ell-z_1} \left(1 - \frac{1}{\ell-z_2}(u-z_2)\right) & \text{otherwise.} \end{cases} \quad (3)$$

There are many choice for parameters a, b and z , we will use the parameter values used in [8]: $z \approx (1/4)\ell$; $b = \ell - z - 1$; $1.5b \leq a \leq 2b$; a a multiple of z . The simple trap function is characterize by a global optimum (for a bit string of all 0's) and a local optimum (for a bit string of all 1's) that are the complement *bit-wise* of each other. The complex trap function is more difficult to investigate, in fact there are two directions to get trapped. We note that for $z_2 = \ell$ the complex trap function becomes the simple trap function. In this case the values of parameter z_2 are determined by the following equation $z_2 = \ell - z_1$. The tables of the next section report the experimental results labelling the trap function with the following syntax: $S(\text{type})$ and $C(\text{type})$; where S and C mean

respectively Simple and Complex trap function, while *type* varying with respect the parameter values used by simple and complex trap functions: type I ($\ell = 10, z = 3, a = 12, b = 6$), type II ($\ell = 20, z = 5, a = 20, b = 14$), type III ($\ell = 50, z = 10, a = 80, b = 39$), type IV ($\ell = 75, z = 20, a = 80, b = 54$), type V ($\ell = 100, z = 25, a = 100, b = 74$). For the complex trap function $z_1 = z$ and $z_2 = \ell - z_1$.

Table 1. Best results obtained by CLONALG (both versions) with population size $N = 10$, varying $\beta \in \{0.1, 0.2, \dots, 1.0\}$, $\rho \in \{1.0, 2.0, \dots, 10.0\}$ and $d \in \{1, 2, 3, 4, 5\}$.

Trap	CLONALG ₁						CLONALG ₂						T_{max}
	$\left(\frac{1}{\rho}\right) e^{-f}$			$e^{-\rho * f}$			$\left(\frac{1}{\rho}\right) e^{-f}$			$e^{-\rho * f}$			
	SR	AES	(β, ρ)	SR	AES	(β, ρ)	SR	AES	(β, ρ)	SR	AES	(β, ρ)	
S(I)	100	1100.4	(.5,3)	100	479.7	(.8,2)	100	725.3	(.9,4)	100	539.2	(.7,2)	10^5
S(II)	100	27939.2	(.8,8)	100	174563.4	(.1,4)	30	173679.8	(.1,6)	31	172191.2	(.1,4)	2×10^5
S(III)	0	-	-	0	-	-	0	-	-	0	-	-	3×10^5
S(IV)	0	-	-	0	-	-	0	-	-	0	-	-	4×10^5
S(V)	0	-	-	0	-	-	0	-	-	0	-	-	5×10^5
C(I)	100	272.5	(.7,3)	100	251.3	(.9,4)	100	254.0	(.3,3)	100	218.4	(.5,4)	10^5
C(II)	100	17526.3	(1,8)	10	191852.7	(.2,1)	29	173992.6	(.1,6)	24	172434.2	(.1,4)	2×10^5
C(III)	0	-	-	0	-	-	0	-	-	0	-	-	3×10^5
C(IV)	0	-	-	0	-	-	0	-	-	0	-	-	4×10^5
C(V)	0	-	-	0	-	-	0	-	-	0	-	-	5×10^5

Table 2. Best results obtained by opt-IA with population size $d = 10$, duplication parameter $dup = 1$, varying $c \in \{0.1, \dots, 1.0\}$ and $tau_B \in \{1, \dots, 15, 20, 25, 50, 100, 200, \infty\}$.

Trap	Inv			Macro			Inv+Macro			T_{max}
	SR	AES	(τ_B, c)	SR	AES	(dup, τ_B)	SR	AES	(τ_B, c)	
S(I)	100	504.76	(5, 0.3)	100	1495.9	(1, 1)	100	477.04	(15, 0.2)	10^5
S(II)	97	58092.7	(20, 0.2)	28	64760.25	(1, 1)	100	35312.29	(100, 0.2)	2×10^5
S(III)	0	-	-	23	19346.09	(4, 13)	100	20045.81	$(2 \times 10^5, 0.1)$	3×10^5
S(IV)	0	-	-	28	69987	(10, 12)	100	42089	(25, 0.2)	4×10^5
S(V)	0	-	-	27	139824.41	(7, 1)	100	80789.94	(50, 0.2)	5×10^5
C(I)	100	371.15	(10, 0.2)	100	737.78	(5, 3)	100	388.42	(10, 0.2)	10^5
C(II)	100	44079.57	(10, 0.2)	100	27392.18	(5, 3)	100	29271.68	(5, 0.2)	2×10^5
C(III)	0	-	-	54	115908.61	(4, 7)	24	149006.5	(20, 0.1)	3×10^5
C(IV)	0	-	-	7	179593.29	(2, 9)	2	154925	(15, 0.4)	4×10^5
C(V)	0	-	-	2	353579	(1, 15)	0	-	-	5×10^5

Experimental results. All the experimental results reported in this sections have been averaged over 100 independent runs. Table 1 shows the best results obtained by CLONALG (both versions) in terms of Success Rate (*SR*) and Average number of Evaluations to Solutions (*AES*), the population size has been set to the

minimal value $N = 10$. The third column in table 1 reports the best parameter values that allowed the hypermutation operators to reach the best results. The last column of the tables reports the maximum number of evaluations allowed, T_{max} , for each kind of trap function.

The results show clearly that, in terms of problem solving ability, facing toy problems is not an easy game. The cases *III*, *IV* and *V* for simple and complex trap functions remain no solved. Moreover, the better result are obtained using mutation rate $(1/\rho) e^{(-f)}$, respect to the ones-counting problem, where the better performance is obtained using $e^{(-\rho*f)}$.

Table 2 shows results obtained with opt-IA using a population size $d = 10$, a minimal duplication parameter $dup = 1$, and varying the parameter $c \in \{0.1, \dots, 1.0\}$ and $\tau_B \in \{1, \dots, 15, 20, 25, 50, 100, 200, \infty\}$. If we compare the results of opt-IA using only the inversely proportional hypermutation operator with the results obtained by CLONALG for population size of 10 Ab's we note how CLONALG outperforms opt-IA. Using the hypermacromutation operator, opt-IA obtains $SR > 0$ for all cases of the simple and complex trap function. Finally, the usage of coupled operators (Inv+Macro) is the key feature to effectively face the trap functions as shown in the third column of table 2. The results obtained with this setting are comparable with the results in [8], where the authors, in their theoretical and experimental research work, use only cases C(I), C(II) and C(III) for the complex trap function.

3 Pattern recognition

In this section we consider the simple pattern recognition task to learn ten binary characters. Each character is represented as a bitstring of length $L = 120$ corresponding to a resolution of 12×10 bits for each picture. The original characters are depicted on figure 4, those characters are the same used in [4]. The fitness measure is the standard Hamming distance for bitstrings.

Experimental results. Figure 3 shows the opt-IA dynamic for each input pattern to be learned. The algorithm is able to recognize all the characters in only 90 generations. This is not true for CLONALG, the overall convergence happens after 250 generations. This is visible in figure 4, where the representations of the antibodies after 200 generations of the algorithms contain a bit of noisy.

4 Numerical Optimization

Numerical optimization problems are fundamental for every field of engineering, science, and business. The task is that of global optimization of a generic objective function. However, often, the objective function is difficult to optimize because the function possesses numerous local optima which could trap the algorithm. Moreover this difficulty increases with the increase of the problem dimension. In this paper we consider the following numerical minimization

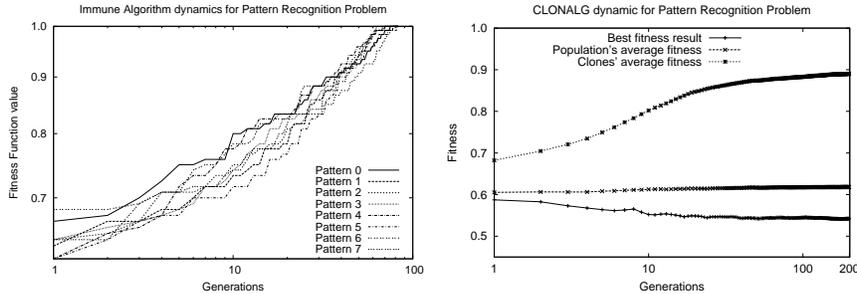


Fig. 3. opt-IA dynamics for each pattern using $d = 10$, $dup = 2$ and $\tau_B = 5$ (left plot). CLONALG population, clones and best Ab average fitness using mutation rate $\alpha = (1/\rho) e^{(-f)}$ on 100 independent runs using the same setting of the parameters in [4]: $N = 10, n = 5, m = 8, \beta = 10, d = 0$ (right plot). For both plots, fitness values in axis y are normalized in $[0, 1]$ and axis x is in log scale.



Fig. 4. CLONALG results on pattern recognition. From left to right: patterns to be learned; initial Abs set; Abs set after 200 generations.

problem:

$$\min(f(\mathbf{x})), \quad L \leq \mathbf{x} \leq U \quad (4)$$

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 $L = (l_1, l_2, \dots, l_n)$, $U = (u_1, u_2, \dots, u_n)$ represent, respectively, the lower and the upper bound of the variables, such that $x_i \in [l_i, u_i]$.

Test Functions. Twentythree functions from three categories are selected [10], covering a broader range. Table 3 lists the 23 functions and their key properties (for a complete description of all the functions and the parameters involved see [10]). These function 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 a 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.

We used binary string representation: each real value x_i is coded using bit-strings of length $L = 22$ corresponding to a precision of six decimal places.

Experimental results. In table 4 we report results obtained with CLONALG and opt-IA respect to one of the best evolutionary algorithms for numerical optimization in literature: Fast Evolutionary Programming (FEP) [10]. FEP is based on Conventional Evolutionary Programming (CEP) but uses a new mutation operator based on Cauchy random numbers that help the algorithm to escape from local optima. In the experiments of this section, opt-IA uses the same mutation potentials above defined for CLONALG (equation 1) for the inversely proportional hypermutation operator. Parameters for CLONALG and opt-IA are setted respectively as follow: $N = n = 50$, $d = 0$, $\beta = 0.1$ and $d = 20$, $dup = 2$, $\tau_B = 20$. If we compare the two versions of CLONALG, we can see that for unimodal functions ($f_1 - f_7$) CLONALG₂ is in general more effective than CLONALG₁. Otherwise, for multimodal functions ($f_8 - f_{23}$), CLONALG₁ has a better performance. This is in agreement with the type of selection scheme used by the two versions. Since CLONALG₁ at each generation replaces each Ab by the best individual of its set of $\beta * N$ mutated clones, it is able to maintain more diversity in the population. On the other hand, CLONALG₂ focuses the search on the global optimum, with the consequence of a higher probability to be trapped in a local optimum.

Considering the two versions of opt-IA, the four versions of CLONALG, and the results obtained by FEP, opt-IA outperforms CLONALG and FEP on 11 functions over 23, analogously to FEP, while CLONALG performs better only in 3 functions (see boldface results in table 4).

Inspecting entries results on table 4 in terms of results obtained by CSAs only, we note that, although CLONALG and opt-IA have equivalent performance for unimodal functions ($f_1 - f_7$), opt-IA outperforms CLONALG on 11 functions for the more difficult class of multimodal functions ($f_8 - f_{23}$), while CLONALG obtains the best results on 5 functions only (results reported in italic in table 4).

5 Protein Structure Prediction: HP model

The Protein Structure Prediction problem (PSP) is simply defined as the problem of finding the 3D conformation of a protein starting from the amino-acids composition. A simplified version of this problem was introduced by Dill in [11] which is called the HP model. It models proteins as two-dimensional *self-avoiding walk chains* of ℓ monomers on the square lattice: two residues cannot occupy the same node of the lattice. Residues are classified into two major classes: H (hydrophobic) and the P (polar). In this model, each H–H topological contact, that is, each lattice nearest-neighbor H–H contact interaction, has energy value $\epsilon \leq 0$, while all other contact interaction types (H–P, P–P) have zero energy. In general, in the HP model the residues interactions can be defined as follows:

Table 3. 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 [10]).

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
$f_3(\mathbf{x}) = \sum_{i=1}^n \left(\sum_{j=1}^i x_j \right)^2$	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
$f_7(\mathbf{x}) = \sum_{i=1}^n ix_i^4 + \text{random}[0, 1]$	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
$f_{14}(\mathbf{x}) = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^2 (x_i - a_{ij})^6} \right]^{-1}$	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

Table 4. Comparison between FEP[10], CLONALG₁, CLONALG₂ and opt-IA on the 23 test functions. Results 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 and T_{max} is the maximum number of fitness function evaluation allowed. In **boldface** overall better results for each function, in *italics* the best results among CLONALG and opt-IA.

Fun. T_{max}	FEP[10]	CLONALG ₁		CLONALG ₂		opt-IA	
		$e(-\rho^*f)$,	$\left(\frac{1}{\rho}\right)e(-f)$,	$e(-\rho^*f)$,	$\left(\frac{1}{\rho}\right)e(-f)$,	$e(-\rho^*f)$,	$\left(\frac{1}{\rho}\right)e(-f)$
		$\rho = 10$	$\rho = 150$	$\rho = 10$	$\rho = 150$	$\rho = 10$	$\rho = 150$
	mean best (std dev)	mean best (std dev)					
f_1 150.000	5.7×10^{-4} (1.3×10^{-4})	9.6×10^{-4} (1.6×10^{-3})	3.7×10^{-3} (2.6×10^{-3})	3.2×10^{-6} (1.5×10^{-6})	5.5×10^{-4} (2.4×10^{-4})	6.4×10^{-8} (2.6×10^{-8})	3.4×10^{-8} (1.3×10^{-8})
f_2 200.000	8.1×10^{-3} (7.7×10^{-4})	7.7×10^{-5} (2.5×10^{-5})	2.9×10^{-3} (6.6×10^{-4})	1.2×10^{-4} (2.1×10^{-5})	2.7×10^{-3} (7.1×10^{-4})	7.4×10^{-5} (4.5×10^{-6})	7.2×10^{-5} (3.4×10^{-6})
f_3 500.000	1.6×10^{-2} (1.4×10^{-2})	2.2×10^4 (1.3×10^{-4})	1.5×10^4 (1.8×10^3)	2.4×10^4 (5.7×10^3)	5.9×10^3 (1.8×10^3)	3.6×10^3 (1.1×10^3)	2.6×10^2 (6.8×10^2)
f_4 500.000	0.30 (0.50)	9.44 (1.98)	4.91 (1.11)	5.9×10^{-4} (3.5×10^{-4})	8.7×10^{-3} (2.1×10^{-3})	1.0×10^{-2} (5.3×10^{-3})	4.9×10^{-3} (3.8×10^{-3})
f_5 2×10^6	5.06 (5.87)	31.07 (13.48)	27.6 (1.034)	4.67×10^2 (6.3×10^2)	2.35×10^2 (4.4×10^{02})	28.6 (0.12)	28.4 (0.42)
f_6 150.000	0.0 (0.0)	0.52 (0.49)	2.0×10^{-2} (1.4×10^{-1})	<i>0.0</i> (<i>0.0</i>)	<i>0.0</i> (<i>0.0</i>)	0.2 (0.44)	<i>0.0</i> (<i>0.0</i>)
f_7 300.000	7.6×10^{-3} (2.6×10^{-3})	1.3×10^{-1} (3.5×10^{-2})	7.8×10^{-2} (1.9×10^{-2})	4.6×10^{-3} (1.6×10^{-3})	5.3×10^{-3} (1.4×10^{-3})	3.4×10^{-3} (1.6×10^{-3})	3.9×10^{-3} (1.3×10^{-3})
f_8 900.000	-12554.5 (52.6)	-11099.56 (112.05)	-11044.69 (186.73)	-12228.39 (41.08)	-12533.86 (43.08)	-12508.38 (155.54)	-12568.27 (0.23)
f_9 500.000	4.6×10^{-2} (1.2×10^{-2})	42.93 (3.05)	37.56 (4.88)	21.75 (5.03)	22.41 (6.70)	19.98 (7.66)	5.68 (1.55)
f_{10} 150.000	1.8×10^{-2} (2.1×10^{-3})	18.96 (2.2×10^{-1})	1.57 (3.9×10^{-1})	19.30 (1.9×10^{-1})	1.2×10^{-1} (4.1×10^{-1})	0.94 (3.56×10^{-1})	4.0×10^{-4} (1.8×10^{-4})
f_{11} 200.000	1.6×10^{-2} (2.2×10^{-2})	3.6×10^{-2} (3.5×10^{-2})	1.7×10^{-2} (1.9×10^{-2})	9.4×10^{-2} (1.4×10^{-1})	4.6×10^{-2} (7.0×10^{-2})	9.1×10^{-2} (1.36×10^{-1})	3.8×10^{-2} (5.5×10^{-2})
f_{12} 150.000	9.2×10^{-6} (3.6×10^{-6})	0.632 (2.2×10^{-1})	0.336 (9.4×10^{-2})	0.738 (5.3×10^{-1})	0.573 (2.6×10^{-1})	0.433 (1.41×10^{-1})	0.364 (5.6×10^{-2})
f_{13} 150.000	1.6×10^{-4} (7.3×10^{-5})	1.83 (2.7×10^{-1})	1.39 (1.8×10^{-1})	1.84 (2.7×10^{-1})	1.69 (2.4×10^{-1})	1.51 (1.01×10^{-1})	1.75 (7.7×10^{-2})
f_{14} 10.000	1.22 (0.56)	1.0062 (4.0×10^{-2})	1.0021 (2.8×10^{-2})	1.45 (0.95)	2.42 (2.60)	1.042 (0.11)	1.21 (0.54)
f_{15} 400.000	5.0×10^{-4} (3.2×10^{-4})	1.4×10^{-3} (5.4×10^{-4})	1.5×10^{-3} (7.8×10^{-4})	8.3×10^{-3} (8.5×10^{-3})	7.2×10^{-3} (8.1×10^{-3})	7.1×10^{-4} (1.3×10^{-4})	7.7×10^{-3} (1.4×10^{-2})
f_{16} 10.000	-1.03 (4.9×10^{-7})	-1.0315 (1.8×10^{-4})	-1.0314 (5.7×10^{-4})	-1.0202 (1.8×10^{-2})	-1.0210 (1.9×10^{-2})	-1.0314 (8.7×10^{-4})	-1.027 (1.0×10^{-2})
f_{17} 10.000	0.398 (1.5×10^{-7})	0.40061 (8.8×10^{-3})	0.399 (2.0×10^{-3})	0.462 (2.0×10^{-1})	0.422 (2.7×10^{-2})	0.398 (2.0×10^{-4})	0.58 (0.44)
f_{18} 10.000	3.02 (0.11)	3.00 (1.3×10^{-7})	3.00 (1.3×10^{-5})	3.54 (3.78)	3.46 (3.28)	3.0 (3.3×10^{-8})	3.0 (0.0)
f_{19} 10.000	-3.86 (1.4×10^{-5})	-3.71 (1.1×10^{-2})	-3.71 (1.5×10^{-2})	-3.67 (6.6×10^{-2})	-3.68 (6.9×10^{-2})	-3.72 (1.5×10^{-4})	-3.72 (1.4×10^{-6})
f_{20} 20.000	-3.27 (5.9×10^{-2})	-3.30 (1.0×10^{-2})	-3.23 (5.9×10^{-2})	-3.21 (8.6×10^{-2})	-3.18 (1.2×10^{-1})	-3.31 (7.5×10^{-3})	-3.31 (5.9×10^{-3})
f_{21} 10.000	-5.52 (1.59)	-7.59 (1.89)	-5.92 (1.77)	-5.21 (1.78)	-3.98 (2.73)	-8.29 (2.25)	-3.73 (0.26)
f_{22} 10.000	-5.52 (2.12)	-8.41 (1.40)	-5.90 (2.09)	-7.31 (2.67)	-4.66 (2.55)	-9.59 (1.72)	-3.79 (0.25)
f_{23} 10.000	-6.57 (3.14)	-8.48 (1.51)	-5.98 (1.98)	-7.12 (2.48)	-4.38 (2.66)	-9.96 (1.46)	-3.86 (0.19)

$e_{HH} = -|\epsilon|$ and $e_{HP} = e_{PH} = e_{PP} = \delta$. When $\epsilon = 1$ and $\delta = 0$ we have the typical interaction energy matrix for the standard HP model [11]. The native conformation is the one that maximizes the number of contacts H–H, i.e. the one that minimizes the free energy function. Finding the global minimum of the free energy function for the protein folding problem in the 2D HP model is NP-hard [12]. The input for the algorithms is a protein sequence of $s \in \{H, P\}^\ell$ where ℓ represents the number of amino-acids. The candidate solution is a sequence of *relative directions* [13] $r \in \{F, R, L\}^{\ell-1}$, where each r_i is a relative direction with respect to the previous direction (r_{i-1}), with $i = 2, \dots, \ell - 1$ (i.e., there are $\ell - 2$ relative directions) and r_1 the non relative direction. We obtain an overall sequence r of length $\ell - 1$.

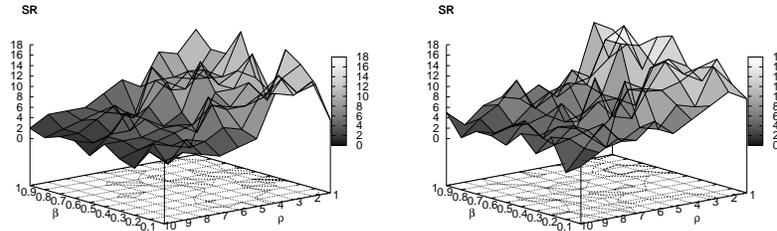


Fig. 5. CLONALG₁: SR as a function of the values β and ρ for mutation rate $\alpha = e^{(-\rho*f)}$ (left plot) and mutation rate $\alpha = \left(\frac{1}{\rho}\right) e^{(-f)}$ (right plot) on seq2 instance.

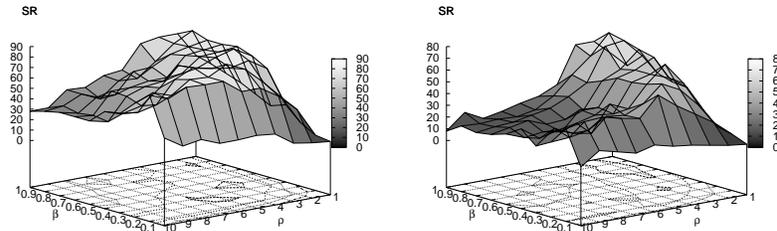


Fig. 6. CLONALG₂: SR as a function of the values β and ρ for mutation rate $\alpha = e^{(-\rho*f)}$ (left plot) and mutation rate $\alpha = \left(\frac{1}{\rho}\right) e^{(-f)}$ (right plot) on seq2 instance.

Experimental results. In this section we report the results obtained for both versions of CLONALG and opt-IA on 12 instances from the *Tortilla 2D HP*

benchmarks¹. Since opt-IA was well studied for the HP model [5, 14, 15], first of all we made a parameter tuning process for CLONALG in order to choose between CLONALG₁ and CLONALG₂ the version with the best performance for PSP, and also in order to set the best values for parameters β and ρ . In particular, parameter surfaces were determined in order to predict the best delimited region that maximizes SR values and minimize AES value. The maximum number of fitness function evaluation (T_{max}) allowed for this first set of experiments is 10^5 . The results are averaged on 100 independent runs.

Table 5. opt-IA algorithm ($d = 10, dup = 2$). Results have been averaged on 30 independent runs, where *b. f* indicates the best values found, symbol μ stands for mean and symbol σ stands for standard deviation.

Protein			$\tau_B = 1$					$\tau_B = 5$				
No.	ℓ	E^*	SR	AES	<i>b. f.</i>	μ	σ	SR	AES	<i>b. f.</i>	μ	σ
1	20	-9	100	23710	-9	-9	0	100	20352.4	-9	-9	0
2	24	-9	100	69816.7	-9	-9	0	100	39959.9	-9	-9	0
3	25	-8	100	269513.9	-8	-8	0	100	282855.7	-8	-8	0
4	36	-14	100	2032504	-14	-13.93	0.25	73.33	4569496.3	-14	-13.73	0.44
5	48	-23	56.67	6403985.3	-23	-22.47	0.67	6.67	4343279	-23	-21.47	0.62
6	50	-21	100	778906.4	-21	-21	0	100	1135818.9	-21	-21	0
7	60	-36	0	//	-35	-33.73	0.68	0	//	-35	-34.5	0.5
8	64	-42	0	//	-39	-36.13	1.28	0	//	-38	-35.1	1.25
9	20	-10	100	18085.8	-10	-10	0	100	18473.6	-10	-10	0
10	18	-9	100	69210	-9	-9	0	100	130342	-9	-9	0
11	18	-8	100	41724.2	-8	-8	0	100	50151.2	-8	-8	0
12	18	-4	100	87494.5	-4	-4	0	100	74426.5	-4	-4	0

Table 6. CLONALG₁ and CLOANLG₂ using mutation rate $\alpha = e^{(-\rho*f)}$ ($N = n = 10, d = 0$). Results have been averaged on 30 independent runs, where *b. f* indicates the best values found, symbol μ stands for mean and symbol σ stands for standard deviation.

Protein			CLOANLG ₁ ($\beta = 0.4, \rho = 1.0$)					CLOANLG ₂ ($\beta = 0.3, \rho = 5.0$)				
No.	ℓ	E^*	SR	AES	<i>b. f.</i>	μ	σ	SR	AES	<i>b. f.</i>	μ	σ
1	20	-9	100	322563.50	-9	-9	0	100	22379.60	-9	-9	0
2	24	-9	90	2225404.75	-9	-8.9	0.3	100	69283.34	-9	-9	0
3	25	-8	96.67	1686092.38	-8	-7.96	0.17	100	907112.56	-8	-8	0
4	36	-14	0	//	-13	-12.23	0.46	23.33	5189238.50	-14	-13.2	0.47
5	48	-23	0	//	-21	-18.93	0.92	3.33	8101204.50	-23	-20.76	1.02
6	50	-21	0	//	-20	-17.43	0.95	46.67	6019418.50	-21	-20.2	0.87
7	60	-36	0	//	-34	-30.43	1.33	0	//	-35	-32.43	0.98
8	64	-42	0	//	-35	-29.26	1.74	0	//	-39	-33.43	2.21
9	20	-10	100	649403.00	-10	-10	0	100	27391.67	-10	-10	0
10	18	-9	96.67	2143456.50	-9	-8.96	0.18	90	1486671.25	-9	-8.9	0.3
11	18	-8	96.67	742352.56	-8	-7.96	0.18	100	52349.10	-8	-8	0
12	18	-4	100	740468.31	-4	-4	0	100	70247.73	-4	-4	0

¹ http://www.cs.sandia.gov/tech_report/compbio/tortilla-hpbenchmarks.html

From figures 5 and 6 it is obvious that CLONALG₂ has a better behavior respect to CLONALG₁, the best SR found by CLONALG₂ is 85 using mutation rate $\alpha = e^{(-\rho * f)}$, while the best SR found by CLONALG₁ is 18. The worse performance of CLONALG₁ is consequence of the structure of the selection scheme for the creation of the new population of antibody, as explained in section 1.1. In fact, this version is more useful for multimodal optimization problems where is necessary to find the greatest number of peaks of a specific function (maximization), as shown in [4] and as demonstrated from results on numerical optimization in section 4. Again, we want to put in evidence the crucial importance of selecting the better mutation rate for each problem, and the tuning of the parameter to which is correlated (ρ).

Tables 5 and 6 show the best results for CLONALG (both versions) and opt-IA on the 12 PSP instances setting $T_{max} = 10^7$. All the results have been averaged over 30 independent runs. For CLONALG the values of β and ρ have been chosen according to figures 5 and 6 when the best SR is found. The mutation rate $\alpha = e^{(-\rho * f)}$ was used, according to its better performance as shown previously. Best results for opt-IA are obtained using coupled operators, inversely proportional hypermutation and Hypermacromutation. As for the traps, this is again the key feature to effectively face the problem. Both algorithms use the same minimal population dimension ($N = d = 10$). For the simple sequences 1,2,3,9,11 and 12 the algorithms have a similar behavior, but when we consider more difficult instances, like sequences 4,5 and 6, the overall performance of opt-IA is evident. Both algorithms are unable to solve the hard sequences 7 and 8, but although they reach the same minimum values, opt-IA has lower mean and standard deviation, showing a more robust behavior.

6 Conclusions

In this experimental work we made a comparative study of two famous Clonal Selection Algorithms, CLONALG and opt-IA, on significant test bed: ones-counting and trap functions (toy problems), pattern recognition, numerical optimization (23 functions) and 2D HP Protein Structure Prediction problem (NP-Complete problem). A robust test bed is important in order to analyze theoretically and experimentally the overall robustness of evolutionary algorithms, as reported in [16]. Two possible versions of CLONALG have been implemented and tested, coupled with two possible mutation potential for the hypermutation operator. The experimental results show a deep influence of the mutation potential for each problem and the setting of the respective parameter. Parameter tuning was made for both algorithms, and an overall better performance of opt-IA was found on all problems tackled. In particular, simulation results on numerical optimization problems show how CSAs (in particular opt-IA) are effective methods also for numerical optimization problems, obtaining comparable results respect to one of the most effective method in literature, Fast Evolutionary Programming. Obviously, the presented clonal selection algorithms can be

applied to any other combinatorial and numerical optimization problem using suitable representations and variable operators [17, 18, 2].

In last years there have been many applications of CSAs to search, learning and optimization problems [19–21]. In particular, this new class of evolutionary algorithms seem to be effective to face protein structure prediction problem [14, 15, 22]. This article and all the abovementioned research works demonstrate as the clonal selection algorithms are mature and effective computational tools [21, 23].

The evolutionary computation scientific community has a new class, *immune algorithms* [4, 17], that along with *genetic algorithms* [24, 25], *evolution strategies* [26], *evolutionary programming* [27] and the *genetic programming* [28] constitutes the overall set of evolutionary algorithms.

References

1. Burnet F.M.: “The Clonal Selection Theory of Acquired Immunity”. Cambridge, UK: *Cambridge University Press*, (1959).
2. 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).
3. De Castro L. N., Timmis J.: “Artificial Immune Systems: A New Computational Intelligence Paradigm” London, UK: *Springer-Verlag*, (2002).
4. 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).
5. Cutello V., Nicosia G., 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).
6. 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).
7. Prugel-Bennett A., Rogers A.: “Modelling GA Dynamics”, *Proc. Theoretical Aspects of Evolutionary Computing*, pp. 59–86, (2001).
8. Nijssen S., Back T.: “An analysis of the Behavior of Simplified Evolutionary Algorithms on Trap Functions”, *IEEE Trans. on Evolutionary Computation*, vol 7(1), pp. 11-22, (2003).
9. Nicosia G., Cutello V., Pavone M., Narzisi G., Sorace G.: “How to Escape Traps using Clonal Selection Algorithms”, *The First International Conference on Informatics in Control, Automation and Robotics (ICINCO) INSTICC Press*, Vol. 1, pp. 322-326, (2004).
10. Yao X., Liu Y., Lin G.M.: “Evolutionary programming made faster”, *IEEE Trans. on Evolutionary Computation*, vol 3, pp. 82-102, (1999).
11. Dill K.A.: “Theory for the folding and stability of globular proteins”, in *Biochemistry*, vol. 24, pp. 1501-1509, (1985).
12. Crescenzi P., Goldman D., Papadimitriou C., Piccolboni A., Yannakakis M.: “On the complexity of protein folding” in *J. of Comp. Bio.*, vol. 5, pp. 423-466, (1998).
13. Krasnogor N., Hart W.E., Smith J., Pelta D.A.: “Protein Structure Prediction with Evolutionary Algorithms”, *GECCO ’99*, vol 2, pp. 1596-1601, (1999).

14. Cutello V., Morelli G., Nicosia G., 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).
15. 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).
16. Goldberg D.E.: "The Design of Innovation: Lessons from and for Competent Genetic Algorithms", *Kluwer Academic Publisher*, vol 7, pp. Boston, (2002).
17. Cutello V. , Nicosia G.: "An Immunological Approach to Combinatorial Optimization Problems". *Proc. of 8th Ibero-American Conf. on Artificial Intelligence (IBERAMIA'02)*, (2002).
18. 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.
19. Garrett S. M.: "Parameter-free, Adaptive Clonal Selection" *Congress on Evolutionary Computing*, Portland Oregon, June (2004).
20. Nicosia G., Cutello V., Bentley P. J., Timmis J.: "Artificial Immune Systems", *Third International Conference, ICARIS 2004*, Catania, Italy, September 13-16, Springer (2004).
21. Nicosia G.: "Immune Algorithms for Optimization and Protein Structure Prediction", PHD Thesis, *University of Catania*", Italy, December 2004.
22. Cutello V., Narzisi G., Nicosia G.: "A Class of Pareto Archived Evolution Strategy Algorithms Using Immune Inspired Operators for Ab-Initio Protein Structure Prediction." *EvoWorkshops 2005*, LNCS, vol. 3449, pp. 54-63, (2005).
23. Garrett S. M.: "A Survey of Artificial Immune Systems: Are They Useful?" *Evolutionary Computation*, vol. 13(2), (2005) (to appear).
24. Holland J.: "Genetic algorithms and the optimal allocation of trials", *SIAM J. Computing*, vol. 2, pp. 88-105, (1973).
25. Holland, J. H.: "Adaptation in Natural and Artificial Systems". Ann Arbor, Michigan: *The University of Michigan Press*, (1975).
26. Rechenberg, I.: "Evolutions strategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution". *Frommann-Holzboog*, Stuttgart, (1973).
27. Fogel, L. J., Owens, A. J., Walsh, M. J. "Artificial Intelligence Through Simulated Evolution". *New York: Wiley Publishing*, (1966).
28. Koza, J. R.: "Evolving a computer program to generate random numbers using the genetic programming paradigm". *Proc. of the Fourth Int. Conf. on GA*, (1991).