Artificial Immune Clonal Selection Algorithms: A Comparative Study of CLONALG, opt-IA, and BCA with Numerical Optimization Problems

Khaled A. Al-Sheshtawi
Information Technology Dept., King AbdulAziz University, Jeddah, Kingdom of Saudi Arabia.

H. M. Abdul-Kader
Information System Dept., Faculty of Computers & Information, Menoufia University, Egypt.

Nabil A. Ismail
Computer Science Dept., Faculty of Computers & Information, Menoufia University, Egypt.

Abstract
This paper presents a comparative study the performance of three important Clonal Selection Algorithms (CSAs): CLONALG, opt-IA, and BCA with numerical optimization problems. Four possible versions of CLONALG have been tested. The experimental results show a global better performance of BCA with respect to CLONALG and opt-IA.

Key words:
Clonal Selection Algorithms, CLONALG, opt-IA, BCA, numerical optimization

1. Introduction

Clonal Selection Algorithms (CSAs) are a special class of Immune algorithms (IA) which are inspired by the Clonal Selection Principle [8] of the human immune system to produce effective methods for search and optimization. In this paper three CSAs are analyzed: CLONal selection ALGorithm (CLONALG) [3], optimization Immune Algorithm (opt-IA) [9], and B-Cell Algorithm (BCA) [7] which use a simplified model of the Clonal Selection Principle. To analyze experimentally the overall performance of those three algorithms, we will test them on a robust set of problems belonging to numerical optimization problems. Each individual of the population is a candidate solution belonging to the fitness landscape of a given computational problem. Using the cloning operator, an immune algorithm produces individuals with higher affinities (higher fitness function values), by introducing blind perturbation (by means of a hypermutation operator) and selecting their improved mature progenies. [2]

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 and antigen are represented by string attributes $m = m_L,...,m_1$, 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 with respect to the Ag (the given objective function). Next, cloning operator selects n Abs that will be cloned independently and proportionally to their antigenic affinities, generating the clone
population P^{cl}. 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 of the population p^{hyp} CLONALG creates randomly d new antibodies that will replace the d lowest fit Abs in the current population. [2]

In this paper we use the CLONALG version for optimization tasks with respect to the following two equations:

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

where a 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. The first potential mutation has been proposed in [3], the original mutation law used by CLONALG; while the second potential mutation has been introduced in [4]. The setting the mutation rates and the parameter p is crucial for the algorithm performance [2]. In the optimization version of CLONALG the affinity proportionate cloning is not useful [2]; we use the same law defined in [3]:

$$N_c = \sum_{i=1}^{n} round(\beta * N) \tag{2}$$

where N_c represents the total number of clones created at each generation, in this way, each antibody (or B cell)

Manuscript received April 5, 2010
Manuscript revised April 20, 2010

produces the same number of clones. Moreover, we assign $N = n$, so all Abs from the population will be selected for cloning in step 1.4 of the algorithm.

The experimental study was conducted using two versions of CLONALG, $CLONALG_1$ and $CLONALG_2$, with different selection scheme in step 1.8 of the algorithm and using the two *potential mutations* above defined (equations 1):

$CLONALG_1$: at generation (t), each Ab will be substituted 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. The pseudo code of CLONALG for optimization [3] is outlined as follows:

Input: Ab, Ngen, n, d, L, β
Output: Ab, f
1. for t = 1 to Ngen,
 1.1 f := decode(Ab);
 1.2 Ab_n := select(Ab, f, n);
 1.3 C := clone(Ab_n, β, f);
 1.4 C* := hypermut (C, f);
 1.5 f* := decode(C*);
 1.6 Ab_n := select(C*, f*, n);
 1.7 Ab := insert (Ab, Ab_n);
 1.8 Ab_d := generate(d, L); Randomly generate d antibodies of length L
 1.9 Ab := replace(Ab, Ab_d, f);
end;
2. f := decode(Ab); Function decode is supposed to decode and evaluate for these decoded values.

1.2. Opt-IA Algorithm

The opt-IA algorithm [2] 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 $x \in P$. 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 pseudo code of opt-IA [2] is outlined as follows:

$opt - IA\ (d, dup, \rho, \tau_B, T_{MAX})$
1. $FFE \leftarrow 0$;
Initialise Fitness Function Evaluations (FFE) with 0
2. $N_c \leftarrow d \times dup$;
N_c represents the total number of clones created at each generation
3. $t \leftarrow 0$;
Initialise the time with 0
4. $(P_d^{(t)}) \leftarrow Population\ Initialisation\ (d)$
Initialise the population $(P_d^{(t)})$ of size d
5. Evaluate $(P_d^{(t)})$
Evaluate(P) computes the fitness function value of each B cell x P.
6. $FFE \leftarrow FFE + d$;
Initialise FFE with population size d and FFE
7. *while (FFE < TMAX) do*
The evolution cycle ends if a maximum number of Fitness Function Evaluations (FFE) is reached.
8. $P_{N_c}^{(clo)} \leftarrow Cloning\ (P_d^{(t)}, dup)$;
The cloning operator, simply, clones each B cell *dup* times producing an intermediate population $(P_{N_c}^{(clo)})$ of size $d \times dup = N_c$.
9. $P_{N_c}^{(hyp)} \leftarrow Hypermutation\ (P_{N_c}^{(clo)}, \rho)$;
The hypermutation operator acts on the B cell receptor of $P_{N_c}^{(clo)}$. The number of mutations M is determined by *mutation potential*. Opt- 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.

The first potential mutation was proposed in [3], while the second potential mutation was introduced in [7].

Two different mutation potential are used, they are defined by the following equations:

$$\alpha = e^{(-\rho * f)}, \qquad \alpha = \left(\frac{1}{\rho}\right) e^{(-f)} \qquad (4)$$

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.

10. *Evaluate* $(P_{N_c}^{(hyp)})$
Evaluate the population
11. $FFE \leftarrow FFE + N_c$;
Increase FFE with the total number of clones created at each generation
12. $(P_d^{(t)}, P_{N_c}^{(hyp)}) = Aging\ (P_d^{(t)}, P_{N_c}^{(hyp)}, \tau_B)$
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 opt-IA algorithm and the other Immune and Evolutionary Algorithms. The parameter τ_B is 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 $a\ P_a^{(t)} d$ and $a\ P_{N_c}^{(hyp)}$. In this way, we obtain the new population $P_a^{(t+1)}$, of d B cells, for the next generation $t + 1$.

13. $(P_a^{(t+1)}) \leftarrow (\mu + \lambda) - Selection\ (a\ P_a^{(t)},\ a\ P_{N_c}^{(hyp)}, \tau_B)$

If $d' < d$ B cells survived, the $(\mu + \lambda) - Selection\ operator$ creates $d - d'$ new B cells (Birth phase).

14. $t \leftarrow t + 1$; The next generation.

1.3. BCA Algorithm

Work in [6] proposed the B-cell algorithm (BCA) which is inspired by the clonal selection process. An important feature of the BCA is its use of a unique mutation operator, called a *contiguous somatic hypermutation*. The representation employed in the BCA is binary shape space, with each cell employing this encoding representing a candidate solution. Each B-cell within the population is evaluated by the objective function, $g(x)$. After evaluation by the objective function, the vector within a B-cell $\underline{v}$ is cloned to produce a clonal pool, C. for each B-cell there exists a clonal pool C within the population all the adaptation takes place within C. The size of C is typically the same size as the population P (population size) (but this does not have to be the case). In order to maintain diversity within the search, a single clone is selected at random and each element in the vector undergoes a random change: subject to a certain probability. This is likened by the authors to the metadynamics of the immune system (a technique also employed in aiNET [8]), but within the BCA a separate random clone is produced, rather than utilising an existing one. Each B-cell $\underline{v}' \in C$ is then subjected to a novel contiguous somatic hypermutation mechanism. The BCA uses a distance function as its stopping criterion: when it is within a certain prescribed distance from the optimum, the algorithm is considered to have converged. If the optimum is unknown, then a measure of how far the optimum located so far is employed, and if no progress is made over a certain number of iterations, the search is terminated [7].

The BCA pseudo code [7] is outlined as follows:

1. *Initialisation*: create an initial random population of individuals P.
2. *Antigenic Presentation:* for $\underline{v} \in P$:
   - *2.1. Clonal Selection and Expansion:*
     - evaluate g(v);
     - clone each B-cell;
     - clone v and place in clonal pool C;
   - *2.2. Metadynamics:*
     - randomly select a clone c in C;
     - randomise the vector;
   - *2.3. Affinity Maturation:*
     - For all c in C, apply the contiguous somatic hypermutation operator;
     - evaluate each clone by applying g(v);
     - if a clone has higher affinity than its parent B-cell v, then $\underline{v} = \underline{c}$;
3. *Cycle:* repeat until a certain stopping criterion is met.

Table 1: The 16 Benchmark Functions Used in Our Experimental Study; n is the Dimension of the Function; f_{min} is the Minimum Value of the Function; $S \subseteq \mathbb{R}^n$ are the Variable Bounds (For Complete Description of All the Functions and Their Related Parameters Involved see [1])

Test Function
Sphere Model $f_1(x) = \sum_{i=1}^{n} x_i^2$ n 30, S $[-100, 100]^n$, f_{min} 0
Schwefel's Problem 2.22 $f_2(x) = \sum_{i=1}^{n} x_i^2 \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$ n 30, S $[-10, 10]^n$, f_{min} 0
Schwefel's Problem 1.2 $f_3(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$ n 30, S $[-100, 100]^n$, f_{min} 0
Schwefel's Problem 2.21 $f_4(x) = \max_i \{ \lvert x_i \rvert, 1 \le i \le n \}$ n 30, S $[-10, 10]^n$, f_{min} 0
Generalised Rosenbrock $f_5(x) = \sum_{i=1}^{n-1} [100 (x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$ n 30, S $[-30, 30]^n$, f_{min} 0
Step $f_6(x) = \sum_{i=1}^{n} (\lvert x_i + 0.5 \rvert)^2$ n 30, S $[-100, 100]^n$, f_{min} 0
Quartic Function i.e. Noise

$f_7(x) = \sum_{i=1}^{n} ix_i^4 + random[0,1)$
n 30, S $[-1.28, 1.28]^n$, f_{min} 0
Generalised Rastrigin's Function
$f_9(x) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10]$
n 30, S $[-5.12, 5.12]^n$, f_{min} 0
Ackley's Function
$f_{10}(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos 2\pi x_i\right) + 20 + e$
n 30, S $[-32, 32]^n$, f_{min} 0
Generalised Griewank Function
$f_{11}(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$
n 30, S $[-600, 600]^n$, f_{min} 0
Generalised Schwefel's 2.26
$f_{12}(x) = \sum_{i=1}^{n} -x_i \sin\left(\sqrt{\lvert x_i\rvert}\right)$
n 30, S $[-500, 500]^n$, f_{min} -12569.5
Generalised Penalised Function
$f_{13}(x) = \frac{\pi}{n}\left\{10\sin^2(\pi y_i) + \sum_{i=1}^{n-1}(y_i-1)^2[1+10\sin^2(\pi y_{i+1})] + (y_n-1)^2\right\} + \sum_{i=1}^{n} u(x_i,10,100,4)$, $y_i = 1 + \frac{1}{4}(x_i+1)$
$u(x_i,a,k,m) = \begin{cases} k(x_i-a)^m, & x_i > a \\ 0, & -a \le x_i \le a \\ k(-x_i-a)^m, & x_i < -a \end{cases}$
n 30, S $[-100, 100]^n$, f_{min} 0
Six-Hump Camel-Back Function
$f_{15}(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$
n 30, S $[-5, 5]^n$, f_{min} 0
Branin Function
$f_{16}(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$
n 30, S $[-5, 10] \times [0, 15]$, f_{min} 0
Goldstein-Price Function
$f_{17}(x) = [1 + (x_1+x_2+1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times [30 + (2x_1 - 3x_2)^2 \times (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$
n 2, S $[-2, 2]^n$, f_{min} 3
Shekel's Foxholes Function
$f_{18}(x) = \left[\frac{1}{500} + \sum_{j=1}^{25}\frac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6}\right]$
n 2, S $[-65.536, 65.536]^n$, f_{min} 1

2. 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 problem:

$$\min(f(x)), \qquad L \le x \le U \tag{5}$$

where $x = (x_1, x_2, ..., x_n)$ is the variable vector in $\Re^n$, $f(x)$ denotes the objective function to minimize, and $L = (l_1, l_2, ..., l_n)$, $U = (u_1, u_2, ..., u_n)$ represent, respectively, the lower and the upper bound of the variables, such that $x_i \in [l_n, u_n]$.

Test Functions. Sixteen functions from three categories are selected [1], covering a broader range. Table I lists the 16 functions and their key properties (for a complete description of all the functions and the parameters involved see [1]). 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_{11}), 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_{13} and f_{14}).

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 function the quality of the final results is more important since it reflects the algorithm's ability in *escaping* from local optima.

3. Experimental Results

In table 2 below, we report results obtained with CLONALG, opt-IA, and B-Cell. In the experiments of this section, B-Cell uses the same mutation potentials above defined for CLONALG (equation 1) for the inversely proportional hypermutation operator. Parameters for CLONALG, opt-IA, and B-Cell are set respectively as follow: For CLONALG, $N = n = 50$, $d = 0$, $\beta = 0.1$; for opt-IA $d = 20$, $dup = 2$, $T_B = 20$; and for BCA $dup = 2$,, n

= 50, *mutation factor = 0.5*. If we compare the two versions of CLONALG, we can see that for unimodal functions *(f_1— f_7)* $CLONALG_2$ is in general more effective than $CLONALG_1$. Otherwise, for multimodal functions (f_8 –f_{16}), $CLONALG_1$ has a better performance. This is in agreement with the type of selection scheme used by the two versions. Since $CLONALG_1$ at each generation replaces each Ab by the best individual of its set of β N mutated clones, it is able to maintain more diversity in the population. On the other hand, $CLONALG_2$ 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 version of B-Cell, the four versions of CLONALG, and the results obtained by FEP [1], opt-IA outperforms CLONALG, B-Cell, and FEP on 8 functions over 16, analogously to FEP, while CLONALG performs better only in 3 functions (results reported in **boldface** in table II). By inspecting the entries on the table II in, we note that, opt-IA outperforms CLONALG on 11 functions over 16 benchmark functions while CLONALG obtains the best results on 7 functions only (results reported in *italic* in table II below). By inspecting the entries on the table II below, we note that, BCA outperforms opt-IA on 13 functions over 16 benchmark functions while opt-IA obtains the best results on only 3 functions (results are reported in underline in table 2).

Table 2: the Best Results Among FEP [1], CLONALG1, CLONALG2, Opt-IA, and B-Cell on 16 Functions. Results Have Been Averaged Over 30 Independent Runs, "Mean Best" Indicates the Mean Best Function Values Found in the Last Generation, "Std. Dev." Stands for Standard Deviation, and Tmax is the Maximum Number of Fitness Function Evaluation Allowed. In boldface Overall Better Results for Each Function, in italics.

Function	$CLONALG_1$		$CLONALG_2$		Opt-IA		B-Cell	FEP [1]
	$e(-\rho^{*}f)$ $\rho = 10$	$\left(\frac{1}{\alpha}\right)e^{(-f)}$ $\rho = 150$	$e(-\rho^{*}f)$ $\rho = 10$	$\left(\frac{1}{\alpha}\right)e^{(-f)}$ $\rho = 150$	$e(-\rho^{*}f)$ $\rho = 10$	$\left(\frac{1}{\alpha}\right)e^{(-f)}$ $\rho = 150$	Mean best (Std. Dev.)	Mean best (Std. Dev.)
	Mean best (Std. Dev.)	Mean best (Std. Dev.)	Mean best (Std. Dev.)	Mean best (Std. Dev.)	Mean best (Std. Dev.)	Mean best (Std. Dev.)		
f_1 *Sphere Model*	3.65E-4 (9.27E-3)	-2.9E-3 (-5.5E-2)	-2E-4 (1.6E-3)	6.5E-3 (2.7E-2)	*-4.7E-16 (1.01E-14)*	-5.1E-14 (3.76E-13)	1.18E-15 (8.39E-15)	5.7E-4 (1.3E-4)
f_2 *Schwefel's 2.22*	-5E-05 (9.14E-4)	5.47E-4 (4.89E-3)	-2.3E-05 (1.18E-4)	4.1E-4 (2.9E-3)	*2.36E-16 (3.69E-15)*	1.78E-15 (6.56E-15)	1.18E-16 (9.17E-16)	8.1E-3 (7.7E-4)
f_3 *Schwefel's 1.2*	4.04348 (42.8923)	-7.21859 (45.4564)	1.293484 (42.3266)	2.42213 (40.54677)	*2.308061 (35.7351)*	*2.308061 (35.73511)*	-9.33914 (43.18147)	**1.6E-2 (1.4E-2)**
f_4 *Schwefel's 2.21*	-1.7E-3 (1.4E-2)	1.9E-2 (5.6E-2)	3.11E-4 (2.08E-3)	1E-3 (3E-2)	*2.17E-13 (1.77E-12)*	*1.2E-12 (6.79E-12)*	1.99E-13 (1.16E-12)	0.30 (0.50)
f_5 *Generalized Rosenbrock's*	0.979586 (0.051)	0.99699 (0.13862)	*0.933088 (8.8E-2)*	*0.96225 (8.1E-2)*	3.928357 (4.3212)	4.053255 (4.289953)	0.962505 (0.29734)	5.06 (5.87)
f_6 *Step Function*	-2.13E-3 (0.32092)	2.8E-2 (0.31554)	5.4E-2 (0.26445)	3.0E-2 (0.301918)	*2.27E-2 (0.29537)*	*2.27E-2 (0.295378)*	7.0E-2 (0.30350)	**0.0 (0.0)**
f_7 *Quartic Function i.e. Noise*	-4.1E-2 (0.52447)	-1.25E-2 (0.58069)	-0.19514 (0.55957)	2.2E-2 (0.49403)	*-2.8E-2 (0.49217)*	3.36E-2 (0.496706)	1.7E-2 (0.483891)	7.6E-3 (2.6E-3)
f_8 *Generalized Schwefel's Problem 2.26*	420.8771 1.038147	421.0714 (1.18785)	420.9586 (0.78432)	421.0694 (0.849675)	*420.9687 (5.44E-06)*	420.9687 (6.96E-06)	420.9687464 (3.71E-07)	-12554.5 (52.6)
f_9 *Generalized Rastrigin's*	3.4E-4 (1.37E-3)	4.93E-4 (5.1E-3)	-4E-6 (7.89E-5)	-2.7E-4 (3.9E-3)	*4.44E-11 (1.02E-09)*	*4.43E-11 (1.0E-09)*	1.50E-10 (9.91E-10)	4.6E-2 (1.2E-2)
f_{10} *Ackley's Function*	7.25E-4 (5.68E-3)	-3.64E-3 (1.85E-2)	1.7E-4 (5.1E-4)	3.1E-3 (1.3E-2)	*1.18E-16 (4.18E-15)*	-1.2E-14 (1.22E-13)	2.36E-16 (2.34E-15)	1.8E-2 (2.1E-3)
f_{11} *Generalized Griewank*	0.167699 (6.33833)	-0.32172 (4.70729)	*-0.55116 (3.11806)*	*0.519165 (3.783)*	-1.50592 (6.14359)	-0.46549 (2.880417)	1.70E-01 (3.979902)	**1.6E-2 (2.2E-2)**
f_{12} *Generalized Penalized*	0.519165 (3.7823)	0.419819 (6.2983)	0.167699 (6.338338)	0.419819 (6.2983)	*5.0E-2 (2.24054)*	-1.50592 (6.143593)	-0.53174 (6.080786)	**9.2E-6 (3.6E-6)**
f_{13} *Six-Hump Camel-Back Function*	2.0E-2 (0.51178)	2.1E-2 (0.51171)	2.0E-2 (0.511822)	6.2E-2 (0.50835)	*-0.12456 0.496557*	*-2.0E-2 (0.51177)*	4.1E-2 (0.510485)	**5.0E-4 (3.2E-4)**
f_{14} *Branin Function*	4.48812 (5.60715)	4.357803 (5.2559)	*4.485622 (4.427214)*	4.519409 (6.1902)	4.005967 (4.40565)	4.113296 (4.456233)	4.22104 (2.914086)	**-1.03 (4.9E-7)**
f_{15} *Goldstein Price Function*	-1.87274 (0.12807)	*-1.87398 (0.1256)*	*-1.87275 (0.128104)*	-1.87079 (0.12889)	-1.87541 (0.12564)	-1.87541 (0.125646)	-1.87565 (0.125401)	**0.398 (1.5E-7)**
f_{16} *Shekel Foxholes Function*	-31.9974 (4.73E-2)	-31.9912 (5.7E-2)	-31.9936 (2.2E-2)	-32.004 (4.1E-2)	31.9802 (6.9E-3)	-32.0178 (0.173571)	-31.9783341 (4.45E-05)	**1.6E-4 (7.3E-5)**

4. Conclusions

In this experimental work we made a comparative study of three Clonal Selection Algorithms, CLONALG, opt-IA, and BCA, on significant test bed, numerical optimization (16 functions). Two possible versions of CLONALG have been 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 BCA was found on all problems tackled. In particular, simulation results on numerical optimization problems show how CSAs (in particular BCA) are effective methods 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 [10]

References

[1] Yao X., Liu Y., Lin G.M., "Evolutionary programming made faster", IEEE Trans. on Evolutionary Computation, vol. 3, pp. 82-102, 1999.

[2] V. Cutello, G. Narzisi, G. Nicosia, M. Pavone, "Clonal Selection Algorithms: A Comparative Case Study using Effective Mutation Potentials", 4th Int. Conference on Artificial Immune Systems, ICARIS 2005, August 14-17, 2005, Banff, Canada. Springer, LNCS 3627:13-28, 2005.

[3] 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.

[4] 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.

[5] V. Cutello, G. Narzisi, G. Nicosia, M. Pavone, "An Immunological Algorithm for Global Numerical Optimization", Artificial Evolution: 7th Int. Conference, Evolution Artificielle, EA 2005, October 26-28, 2005, Lille, France. Springer, LNCS 3871:284-295, 2005.

[6] Kelsey, J and Timmis, "Immune Inspired Somatic Contiguous Hypermutation", In E. Cantú-Paz et al, editor, Genetic and Evolutionary Computation Conference - GECCO 2003, volume 2723 of Lecture Notes in Computer Science, Springer-Verlag, Chicago, USA., July 2003.

[7] Timmis, J., Edmonds, C., and Kelsey, J. "Assessing the Performance of Two Immune Inspired Algorithms and a Hybrid Genetic Algorithm for Function Optimisation", In Proceedings of the Congress on Evolutionary Computation, vol. 1, pp. 1044-1051, 2004.

[8] De Castro, L.N and Timmis, J. "Artificial Immune Systems: A New Computational Intelligence Approach", Springer-Verlag. 2002.

[9] 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.

[10] Cutello V. , Nicosia G.: "An Immunological Approach to Combinatorial Optimization Problems". Proc. of 8th Ibero-American Conference on Artificial Intelligence (IBERAMIA'02), 2002.

Khaled AbdElKhalek Al-Sheshtawi obtained his B.Sc. in Computer Science from King AbdulAziz University, Faculty of Science, Saudi Arabia in 1991 and M.Sc. with Distinction in Computer-based Information Systems from University of Sunderland, School of Computing and Information Systems, United Kingdom in 1998. He is currently a Lecturer in King AbdulAziz University, Information Technology Department since 1999.

Hatem Mohamed Abdul-Kader obtained his B.S. and M.SC. (by research) both in Electrical Engineering from the Alexandria University , Faculty of Engineering , Egypt in 1990 and 1995 respectively. He obtained his Ph.D. degree in Electrical Engineering also from Alexandria University, Faculty of Engineering, Egypt in 2001 specializing in neural networks and its applications. He is currently a Associative professor in Information systems department, Faculty of Computers and Information, Minufiya University, Egypt since 2004. He has worked on a number of research topics and consulted for a number of organizations. He has contributed more than 35+ technical papers in the areas of neural networks, Database applications, Information security and Internet applications.

Nabil A. Ismail is a professor of Computer Science and Engineering. He used to be the Dean of the Faculty of Computers and Information, University of Menoufia, Egypt (2006-2008). Prof. Ismail has obtained his PhD from Durham University, England, in 1983. He is now working as a Professor at the Faculty of Electronic Engineering, Egypt and Al-Baha College of Computer Science, KSA. Prof. Nabil Ismail research interests including computer security, image reconstruction, computer architecture, and constrained-resource ECP.