

# Real Coded Clonal Selection Algorithm for Unconstrained Global Optimization using a Hybrid Inversely Proportional Hypermutation Operator

Vincenzo Cutello  
Department of Mathematics  
and Computer Science  
University of Catania  
V.le A. Doria, 6  
95125, Catania, Italy  
vctl@dm.unict.it

Giuseppe Nicosia  
Department of Mathematics  
and Computer Science  
University of Catania  
V.le A. Doria, 6  
95125, Catania, Italy  
nicosia@dm.unict.it

Mario Pavone  
Department of Mathematics  
and Computer Science  
University of Catania  
V.le A. Doria, 6  
95125, Catania, Italy  
mpavone@dm.unict.it

## ABSTRACT

Numerical optimization of given objective functions is a crucial task in many real-life problems. This paper introduces a new immunological algorithm for continuous global optimization problems, called *opt-IMMALG*; it is an improved version of a previously proposed clonal selection algorithm, using a real-code representation and a new Inversely Proportional Hypermutation operator.

We evaluate and assess the performance of *opt-IMMALG* and several others algorithms, namely *opt-IA*, *PSO*, *arPSO*, *DE*, and *SEA* with respect to their general applicability as numerical optimization algorithms. The experiments have been performed on 23 widely used benchmark problems.

The experimental results show that *opt-IMMALG* is a suitable numerical optimization technique that, in terms of accuracy, outperforms the analyzed algorithms in this comparative study. In addition it is shown that *opt-IMMALG* is also suitable for solving large-scale problems.

## Categories and Subject Descriptors

I.2.8 [Problem Solving, Control Methods, and Search]: Heuristic methods.

## General Terms

Algorithms.

## Keywords

Artificial Immune Systems, Immune Algorithms, Clonal Selection Algorithms, Hypermutation Operator, Aging Operator, Global Optimization.

## 1. CLONAL SELECTION ALGORITHM

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC'06 April 23-27, 2006, Dijon, France

Copyright 2006 ACM 1-59593-108-2/06/0004 ...\$5.00.

Clonal Selection Algorithms are a special class of Immune algorithms (IA) which are inspired by the Clonal Selection Principle [1] to produce effective mechanisms for search and optimization [2, 3]. In this paper we present an improved version of the immune algorithm, proposed in [4], which is inspired by the clonal selection principle, and use it for unconstrained global numerical optimization. We call this new algorithm *opt-IMMALG* (*optimization IMMune ALG*).

The algorithm is population based, like any typical evolutionary algorithm. 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), introducing blind perturbation (by means of a hypermutation operator) and selecting their improved mature progenies. *opt-IMMALG* considers only two entities: antigens (Ag's) and B cells. The Ags are the problem to tackle or the function to optimize, whereas the B cells (or B cell receptors in immunological terminology), are the candidate solutions that have solved/approximated the problem, i.e. a population of points of the search space. At each time step  $t$ , we have a population  $P^{(t)}$  of size  $d$ . The initial population of candidate solutions at time  $t = 0$  is randomly generated using uniform distribution in the relative domains of each function. All genes of each B cell receptor  $\vec{x} = (x_1, x_2, \dots, x_n)$ , are randomly initialized using the formula:  $x_i = B_{l_i} + \beta \times (B_{u_i} - B_{l_i})$ , where  $\beta \in [0, 1]$  is a random number with uniform distribution,  $B_{l_i}$  and  $B_{u_i}$  are the lower and the upper bounds of the real coded variable  $x_i$ , respectively. The function  $Evaluate(P)$  computes the fitness function value of each B cell  $\vec{x} \in P$ .

The implemented IA uses three immune operators: cloning, hypermutation and aging. The cloning operator 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 inversely proportional to objective function value.

The proposed *opt-IMMALG* uses an *Inversely Proportional Hypermutation* operator, where the number of mutations is inversely proportional to the fitness value, thus as the fitness function value of the current B cell increases,

```

opt-IMMALG( $d, dup, \rho, \tau_B, T_{max}$ )
   $FFE \leftarrow 0$ ;
   $N_c \leftarrow d \times dup$ ;
   $t \leftarrow 0$ ;
   $P_d^{(t)} \leftarrow \text{Init\_Population}(d)$ ;
  Evaluate( $P_d^{(t)}$ );
   $FFE \leftarrow FFE + d$ ;
  while ( $FFE < T_{max}$ ) do
     $P_{N_c}^{(clo)} \leftarrow \text{Cloning}(P_d^{(t)}, dup)$ ;
     $P_{N_c}^{(hyp)} \leftarrow \text{Hypermutation}(P_{N_c}^{(clo)}, \rho)$ ;
    Evaluate( $P_{N_c}^{(hyp)}$ );
     $FFE \leftarrow FFE + N_c$ ;
    ( ${}^a P_d^{(t)}, {}^a P_{N_c}^{(hyp)}$ ) = Aging( $P_d^{(t)}, P_{N_c}^{(hyp)}, \tau_B$ );
     $P_d^{(t+1)} \leftarrow (\mu + \lambda)\text{-Selection}({}^a P_d^{(t)}, {}^a P_{N_c}^{(hyp)})$ ;
     $t \leftarrow t + 1$ ;
  end\_while

```

**Table 1: Pseudo-code of opt-IMMALG.**

the number of mutations performed by opt-IMMALG decreases. Two different mutation potentials are used. They are defined by the following equations:  $\alpha = e^{(-\rho \times f)}$ , and  $\alpha = \left(\frac{1}{\rho}\right) e^{(-f)}$  where  $\alpha$  represents the mutation rate and  $f$  is the fitness function value normalized in  $[0, 1]$ . The number of mutations of a clone (i.e., a candidate solution), with fitness function value  $f$ , is equal to  $\lceil n \times \alpha \rceil$ , where  $n$  is the length of the clone receptor, i.e. the dimension of the function. The first potential mutation was proposed in [5], while the second was introduced in [6]. Once the *mutation potential* on each  $\vec{x}$  cloned B cell is determined, the perturbation operator randomly choose a variable  $x_i$ , with  $i \in \{1, \dots, n\}$ , and replaces it with  $x_i^{new} = ((1 - \beta) \times x_i) + (\beta \times x_{random})$ , where  $x_{random} \neq x_i$  is a randomly chosen variable and  $\beta \in [0, 1]$  is a random number obtained with uniform distribution. It is important to highlight that we do not use any additional information concerning the problem. For example, the global optima is not considered when normalizing the fitness function value. Instead, the best current fitness value is decreased by a percentage  $\theta$ .

The aging operator, used by the algorithm, eliminates old B cells, in the populations  $P_d^{(t)}$  and  $P_{N_c}^{(hyp)}$ , maintaining high diversity in the current population in order to avoid premature convergence. The maximum number of generations the B cells are allowed to remain in the population is determined by the parameter  $\tau_B$ . When a B cell is  $\tau_B + 1$  old it is erased from the current population; the only exception the algorithm makes is: when generating a new population the selection mechanism does not allow the elimination of the B cell with the best fitness function value (*elitist aging*). During the cloning expansion, a cloned B cell adopts the age of its parent. A cloned B cell which successfully mutates, in the hypermutation phase, will have an age equal to 0. In this way, new B cells are given an equal opportunity to effectively explore the computational landscape. After the application of the immune operators the best surviving B cells are selected from the populations  ${}^a P_d^{(t)}$  and  ${}^a P_{N_c}^{(hyp)}$ . 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 have survived, the  $(\mu + \lambda)\text{-Selection operator}$  randomly selects  $d - d'$  "old"

B cells from  ${}^a P_d^{(t)} \sqcup {}^a P_{N_c}^{(hyp)}$ . The evolution cycle ends when the maximum number of fitness function evaluations,  $T_{max}$ , is reached. Table 1 shows the pseudo-code of the proposed clonal selection algorithm.

## 2. NUMERICAL OPTIMIZATION

Numerical optimization problems are fundamental to every field of engineering and science, since analytical optimal solutions are difficult to obtain, even for relatively simple application problems. A global minimization problem can be formalized as a pair  $(S, f)$ , where  $S \subseteq R^n$  is a bounded set on  $R^n$  and  $f : S \rightarrow R$  is an  $n$ -dimensional real-valued function. The goal is to find a point  $x_{min} \in S$  such that  $f(x_{min})$  is a global minimum on  $S$ , i.e.  $\forall x \in S : f(x_{min}) \leq f(x)$ . However, a generic objective function is difficult to optimize, because it could have numerous local optima the algorithm could get trapped in. The difficulty increases proportionally with the dimension of the problem. In this paper we consider the following numerical minimization problem:  $\min(f(\vec{x}))$ ,  $\vec{B}_l \leq \vec{x} \leq \vec{B}_u$  where  $\vec{x} = (x_1, x_2, \dots, x_n)$  is the variable vector in  $R^n$ ,  $f(\vec{x})$  denotes the objective function to minimize and  $\vec{B}_l = (B_{l_1}, B_{l_2}, \dots, B_{l_n})$ ,  $\vec{B}_u = (B_{u_1}, B_{u_2}, \dots, B_{u_n})$  represent, respectively, the lower and the upper bounds of the variables, such that  $x_i \in [B_{l_i}, B_{u_i}]$ .

To test the robustness and effectiveness of opt-IMMALG we have used a well-known benchmark: twenty-three functions from three different categories [7]. This relatively large set is necessary in order to reduce biases in evaluating algorithms. For a complete description of all the functions and the parameters involved see [7]. These functions can be divided into three categories with different complexities: unimodal functions ( $f_1 - f_7$ ), which are relatively easy to optimize, but the difficulty increases as the dimensions of the problems increase; multimodal functions ( $f_8 - f_{13}$ ), which have many local minima, represent the most difficult class of problems for many optimization algorithms; multimodal functions which contain only few local optima ( $f_{14} - f_{23}$ ). It is interesting to note that some functions have unique features:  $f_6$  is a discontinuous step function having a single optimum;  $f_7$  is a noisy function involving a uniformly distributed random variable within  $[0, 1]$ . In unimodal functions the convergence rate is our main interest, as the optimization is not a "hard" problem. Obviously, for multimodal functions the quality of the final results is more important because it reflects the ability of the designed algorithm to *escape* from local optima.

## 3. EXPERIMENTAL RESULTS

To understand the search ability of opt-IMMALG, we have used a set of experiments on the 23 functions reported in [7], varying the parameter values of  $\theta$  and  $\rho$ . The values used for the  $\rho$  parameter are  $\{50, 75, 100, 125, 150, 175, 200\}$  for the mutation rate  $\alpha = \left(\frac{1}{\rho}\right) e^{(-f)}$  and  $\{4, 5, 6, 7, 8, 9, 10, 11\}$  for  $\alpha = e^{(-\rho \times f)}$ . The parameter  $\theta$  represents the percentage used to decrease the best fitness function value obtained in each generation to normalize the fitness in the range  $[0, 1]$ . Preliminary result show good performances of opt-IMMALG using  $\theta = 95\%$ .

For each test function 50 independent runs were performed using the following parameters:  $d = 100$ ,  $dup = 2$  and  $\tau_B = 15$ . At each generation we compute the mean value

of the best fit individuals for all 50 runs and the standard deviation which is used to indicate the consistency of the algorithm.

There are several algorithms designed for numerical optimization. The performance of opt-IMMALG is compared with some other well-established evolutionary algorithms, such as *Differential Evolution*, *Particle Swarm Optimization*, and *Simple Evolutionary Algorithm* [8].

In the following, we present the results obtained by opt-IMMALG, and compare them with the ones obtained by OPT-IA, proposed in [4], which uses binary string representation. In table 2 we report the best results obtained by two different kinds of IAs, using the same mutation potentials. The best results for each function and mutation rate are displayed in boldface.

opt-IMMALG achieves better results on 18 of the 23 functions. In particular, with the mutation rate  $e^{(-\rho \times f)}$ , the proposed IA is more effective on multimodal functions with many local minima, as it outperforms the other algorithms on all the 6 functions. This reflects opt-IMMALG's ability in *escaping* from local optima.

To assess the real robustness and performances of our proposed Immune Algorithm, we have compared opt-IMMALG with three other well-known evolutionary algorithms, using a different experimental protocol proposed in [8]: *Differential Evolution* (DE), *Particle Swarm Optimization* (PSO) and *simple Evolutionary Algorithm* (SEA). In addition to the classical PSO, the *attractive and repulsive PSO* (arPSO) was proposed in [8], which is a modification scheme for PSO to avoid a premature convergence. The authors of [8] tested each proposed algorithm on all benchmark functions [7], except for the functions 19 and 20. For each experiment the maximum number of fitness function evaluations  $T_{max}$  was fixed to  $5 \times 10^5$ , for 30 and less dimension of the functions, performing for each instance 30 independent runs. Moreover, the authors tested the algorithms using 100 dimension on the functions  $f1 - f13$ . In this case,  $T_{max}$  was set to  $5 \times 10^6$ .

The results obtained in 30 dimensions by opt-IMMALG (see table 3) are comparable to results obtained by DE. Moreover, the real coded immune algorithm outperforms PSO, arPSO, and SEA.

Table 4 shows the experimental results with dimension 100. Results have been averaged over 30 independent runs and  $T_{max} = 5 \times 10^6$ , "mean best" indicates the mean best fitness values found in the last generation, "std dev" stands for standard deviation. In the cases of 30 and 100 variables opt-IMMALG is outperforming PSO, arPSO, and SEA and is comparable to the state-of-art algorithm for numerical optimization, DE.

## 4. CONCLUSION

In this paper we have introduced a new immune algorithm based on the clonal selection principle, called opt-IMMALG (*optimization Immune Algorithm*), for global optimization. This algorithm is an improved version of OPT-IA [4]. The main features of the opt-IMMALG algorithm are the following: cloning operator, inversely proportional hypermutation operator and aging operator. The cloning operator explores the neighbourhood 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 [9].

The binary string representation has been replaced by a real-coded one and a new Inversely Proportional Hypermutation operator was introduced. We have tested the performance of opt-IMMALG on 23 well-known benchmark problems, from three categories of different complexity: unimodal functions, multimodal functions with many local minima, and multimodal functions with a few local optima. The experimental studies show that the proposed clonal selection algorithm is an effective numerical optimization algorithm.

## 5. ADDITIONAL AUTHORS

Additional authors: Giuseppe Narzisi (Department of Mathematics and Computer Science, University of Catania, email: narzisi@dmi.unict.it)

## 6. REFERENCES

- [1] 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).
- [2] Nicosia G., Cutello V., Bentley P. J., Timmis J., "Artificial Immune Systems", *Third Int. Conf. on AIS, ICARIS 2004*, Catania, Italy, September 13-16, Springer (2004).
- [3] Nicosia G., "Immune Algorithms for Optimization and Protein Structure Prediction", Department of Mathematics and Computer Science, University of Catania, Italy, (2004).
- [4] Cutello V., Narzisi G., Nicosia G., and Pavone M.; "An Immunological Algorithm for Global Numerical Optimization," in *Proc. of the Seventh Int. Conf. on Artificial Evolution (EA'05)*, Lille, France, October 26-28, (2005). To appear.
- [5] De Castro L.N., Von Zuben F.J.: "Learning and optimization using the clonal selection principle". *IEEE Trans. on Evol. Comp.*, vol 6, no 3, pp. 239-251, (2002).
- [6] De Castro L. N., Timmis J.,: "An Artificial Immune Network for Multimodal Function Optimization", *CEC'02, Congress on Evol. Comp.*, IEEE Press, (2002).
- [7] Yao X., Liu Y. and Lin G.M.: "Evolutionary programming made faster", *IEEE Trans. on Evol. Comp.*, vol 3, pp. 82-102, (1999).
- [8] Versterstrøm, J. and Thomsen R.: "A Comparative Study of Differential Evolution, Particle Swarm Optimization, and Evolutionary Algorithms on Numerical Benchmark Problems." *Congress on Evol. Comp., CEC04*, vol. 1, pp. 1980-1987, (2004).
- [9] 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, LNCS*, vol. 2723, pp. 171-182.

Table 2: Real vs. Binary representation: comparison of opt-IMMARG, which used real representation, with opt-IA, which used a binary string representation. Results have been averaged over 50 independent runs.

|          | $\alpha = e^{(-\rho * f)}$   |   | $\alpha = \left(\frac{1}{\rho}\right) e^{(-f)}$  |   |
|----------|--|---|--|---|
|          | opt-IMMARG   | OPT-IA  | opt-IMMARG   | OPT-IA  |
|          | Mean Best<br>(Std Dev)   | Mean Best<br>(Std Dev)  | Mean Best<br>(Std Dev)   | Mean Best<br>(Std Dev)  |
| $f_1$    | <b>0.0</b><br>(0.0)  | $9.23 \times 10^{-12}$<br>$2.44 \times 10^{-11}$              | <b><math>4.663 \times 10^{-19}</math></b><br>( <b><math>7.365 \times 10^{-19}</math></b> ) | $1.7 \times 10^{-8}$<br>$3.5 \times 10^{-15}$   |
| $f_2$    | <b>0.0</b><br>(0.0)  | <b>0.0</b><br>(0.0)   | <b><math>3.220 \times 10^{-17}</math></b><br>( <b><math>1.945 \times 10^{-17}</math></b> ) | $7.1 \times 10^{-8}$<br>(0.0)   |
| $f_3$    | <b>0.0</b><br>(0.0)  | <b>0.0</b><br>(0.0)   | 3.855<br>(5.755)   | <b><math>1.9 \times 10^{-10}</math></b><br>( <b><math>2.63 \times 10^{-10}</math></b> ) |
| $f_4$    | <b>0.0</b><br>(0.0)  | $1.0 \times 10^{-2}$<br>( $5.3 \times 10^{-3}$ )              | <b><math>8.699 \times 10^{-3}</math></b><br>( <b><math>3.922 \times 10^{-2}</math></b> )   | $4.1 \times 10^{-2}$<br>( $5.3 \times 10^{-2}$ )  |
| $f_5$    | 16.29<br>(13.96)   | <b>3.02</b><br>( <b>12.2</b> )                                | <b>22.32</b><br>( <b>11.58</b> )   | 28.4<br>(0.42)  |
| $f_6$    | <b>0.0</b><br>(0.0)  | 0.2<br>(0.44)   | <b>0.0</b><br>(0.0)  | <b>0.0</b><br>(0.0)   |
| $f_7$    | <b><math>1.995 \times 10^{-5}</math></b><br>( <b><math>2.348 \times 10^{-5}</math></b> )   | $3.0 \times 10^{-3}$<br>( $1.2 \times 10^{-3}$ )              | <b><math>1.143 \times 10^{-4}</math></b><br>( <b><math>1.411 \times 10^{-4}</math></b> )   | $3.9 \times 10^{-3}$<br>( $1.3 \times 10^{-3}$ )  |
| $f_8$    | <b>-12535.15</b><br>( <b>62.81</b> )   | -12508.38<br>(155.54)   | -12559.69<br>(34.59)   | <b>-12568.27</b><br>( <b>0.23</b> )   |
| $f_9$    | <b>0.596</b><br>( <b>4.178</b> )   | 19.98.0<br>(7.66)   | <b>0.0</b><br>(0.0)  | 2.66<br>(2.39)  |
| $f_{10}$ | <b>0.0</b><br>(0.0)  | 18.98<br>(0.35)   | <b><math>1.017 \times 10^{-10}</math></b><br>( <b><math>5.307 \times 10^{-11}</math></b> ) | $1.1 \times 10^{-4}$<br>( $3.1 \times 10^{-5}$ )  |
| $f_{11}$ | <b>0.0</b><br>(0.0)  | $7.7 \times 10^{-2}$<br>( $8.63 \times 10^{-2}$ )             | <b><math>2.066 \times 10^{-2}</math></b><br>( <b><math>5.482 \times 10^{-2}</math></b> )   | $4.55 \times 10^{-2}$<br>( $4.46 \times 10^{-2}$ )                                      |
| $f_{12}$ | <b><math>1.770 \times 10^{-21}</math></b><br>( <b><math>8,774 \times 10^{-24}</math></b> ) | 0.137<br>(0.23)   | <b><math>7.094 \times 10^{-21}</math></b><br>( <b><math>5.621 \times 10^{-21}</math></b> ) | $3.1 \times 10^{-2}$<br>( $5.7 \times 10^{-2}$ )  |
| $f_{13}$ | <b><math>1.687 \times 10^{-21}</math></b><br>( <b><math>5.370 \times 10^{-24}</math></b> ) | 1.51<br>(0.10)  | <b><math>1.122 \times 10^{-19}</math></b><br>( <b><math>2.328 \times 10^{-19}</math></b> ) | 3.20<br>(0.13)  |
| $f_{14}$ | <b>0.998</b><br>( <b><math>1.110 \times 10^{-3}</math></b> )                               | 1.02<br>( $7.1 \times 10^{-2}$ )                              | <b>0.999</b><br>( <b><math>7.680 \times 10^{-3}</math></b> )                               | 1.21<br>(0.54)  |
| $f_{15}$ | <b><math>3.200 \times 10^{-4}</math></b><br>( <b><math>2.672 \times 10^{-5}</math></b> )   | $7.1 \times 10^{-4}$<br>( $1.3 \times 10^{-4}$ )              | <b><math>3.270 \times 10^{-4}</math></b><br>( <b><math>3.651 \times 10^{-5}</math></b> )   | $7.7 \times 10^{-3}$<br>( $1.4 \times 10^{-2}$ )  |
| $f_{16}$ | -1.013<br>( $2.212 \times 10^{-2}$ )   | <b>-1.03158</b><br>( <b><math>1,5 \times 10^{-4}</math></b> ) | -1.017<br>( $2.039 \times 10^{-2}$ )   | <b>-1.02</b><br>( <b><math>1.1 \times 10^{-2}</math></b> )                              |
| $f_{17}$ | 0.423<br>( $3.217 \times 10^{-2}$ )  | <b>0.398</b><br>( <b><math>2.0 \times 10^{-4}</math></b> )    | <b>0.425</b><br>( <b><math>4.987 \times 10^{-2}</math></b> )                               | 0.450<br>(0.21)   |
| $f_{18}$ | 5.837<br>(3.742)   | <b>3.0</b><br>(0.0)   | 6.106<br>(4.748)   | <b>3.0</b><br>(0.0)   |
| $f_{19}$ | <b>-3.720</b><br>( $7.846 \times 10^{-3}$ )  | <b>-3.72</b><br>( $1.1 \times 10^{-4}$ )                      | <b>-3.720</b><br>( $8.416 \times 10^{-3}$ )  | <b>-3.72</b><br>( $1.1 \times 10^{-2}$ )  |
| $f_{20}$ | -3.292<br>( $3.097 \times 10^{-2}$ )   | <b>-3.31</b><br>( <b><math>7.4 \times 10^{-2}</math></b> )    | -3.293<br>( $3.022 \times 10^{-2}$ )   | <b>-3.31</b><br>( <b><math>5.9 \times 10^{-3}</math></b> )                              |
| $f_{21}$ | <b>-10.153</b><br>( <b><math>1.034 \times 10^{-7}</math></b> )                             | -9.11<br>(1.82)   | <b>-10.153</b><br>( <b><math>7.710 \times 10^{-8}</math></b> )                             | -5.36<br>(2.20)   |
| $f_{22}$ | <b>-10.402</b><br>( <b><math>1.082 \times 10^{-5}</math></b> )                             | -9.86<br>(1.88)   | <b>-10.402</b><br>( <b><math>1.842 \times 10^{-6}</math></b> )                             | -5.34<br>(2.11)   |
| $f_{23}$ | <b>-10.536</b><br>( <b><math>1.165 \times 10^{-5}</math></b> )                             | -9.96<br>(1.46)   | <b>-10.536</b><br>( <b><math>7.694 \times 10^{-7}</math></b> )                             | -6.03<br>(2.66)   |

**Table 3: Performance Comparison among opt-IMMALG, DE, PSO, arPSO and SEA on the first 13 functions with dimension 30.**

|          | opt-IMMALG  |  | DE   | PSO   | arPSO  | SEA   |
|----------|---|--|--|---|--|---|
|          | $\alpha = \left(\frac{1}{\rho}\right) e^{(-f)}$       | $\alpha = e^{(-\rho * f)}$                         |  |   |  |   |
|          | Mean Best<br>Std Dev                                  | Mean Best<br>Std Dev                               | Mean Best<br>Std Dev                               | Mean Best<br>Std Dev                              | Mean Best<br>Std Dev                             | Mean Best<br>Std Dev                              |
| $f_1$    | 0.0<br>0.0  | 0.0<br>0.0   | 0.0<br>0.0   | 0.0<br>0.0  | $6.8 \times 10^{-13}$<br>$5.3 \times 10^{-13}$   | $1.789 \times 10^{-3}$<br>$2.77 \times 10^{-4}$   |
| $f_2$    | 0.0<br>0.0  | 0.0<br>0.0   | 0.0<br>0.0   | 0.0<br>0.0  | $2.089 \times 10^{-2}$<br>$1.48 \times 10^{-1}$  | $1.72 \times 10^{-2}$<br>$1.7 \times 10^{-3}$     |
| $f_3$    | 0.0<br>0.0  | 0.0<br>0.0   | $2.02 \times 10^{-9}$<br>$8.26 \times 10^{-10}$    | 0.0<br>0.0  | 0.0<br>$2.13 \times 10^{-25}$                    | $1.589 \times 10^{-2}$<br>$4.25 \times 10^{-3}$   |
| $f_4$    | $5.604 \times 10^{-4}$<br>$2.184 \times 10^{-3}$      | 0.0<br>0.0   | $3.85 \times 10^{-8}$<br>$9.17 \times 10^{-9}$     | $2.107 \times 10^{-16}$<br>$8.01 \times 10^{-16}$ | $1.418 \times 10^{-5}$<br>$8.27 \times 10^{-6}$  | $1.982 \times 10^{-2}$<br>$2.07 \times 10^{-3}$   |
| $f_5$    | 21.156<br>11.395                                      | 12<br>13.22  | 0.0<br>0.0   | 4.026<br>4.99                                     | $3.55 \times 10^{+2}$<br>$2.15 \times 10^{+3}$   | 31.3189<br>17.4                                   |
| $f_6$    | 0.0<br>0.0  | 0.0<br>0.0   | 0.0<br>0.0   | $4 \times 10^{-2}$<br>$1.98 \times 10^{-1}$       | 18.98<br>63                                      | 0.0<br>0.0  |
| $f_7$    | $3.705 \times 10^{-5}$<br>$5.618 \times 10^{-5}$      | $1.521 \times 10^{-5}$<br>$2.05 \times 10^{-5}$    | $4.939 \times 10^{-3}$<br>$1.13 \times 10^{-3}$    | $1.908 \times 10^{-3}$<br>$1.14 \times 10^{-3}$   | $3.8866 \times 10^{-4}$<br>$4.78 \times 10^{-4}$ | $7.106 \times 10^{-4}$<br>$3.27 \times 10^{-4}$   |
| $f_8$    | $-1.256697 \times 10^{+4}$<br>8.369                   | $-1.256041 \times 10^{+4}$<br>25.912               | $-1.256048 \times 10^{+4}$<br>$2.3 \times 10^{-4}$ | $-7.187 \times 10^{+3}$<br>$6.72 \times 10^{+2}$  | $-8.598 \times 10^{+3}$<br>$2.07 \times 10^{+3}$ | $-1.1669 \times 10^{+4}$<br>$2.34 \times 10^{+2}$ |
| $f_9$    | 0.0<br>0.0  | 0.0<br>0.0   | 0.0<br>0.0   | 49.17<br>16.2                                     | 2.149<br>4.91                                    | $7.1789 \times 10^{-1}$<br>$9.22 \times 10^{-1}$  |
| $f_{10}$ | $4.7369 \times 10^{-16}$<br>$1.20769 \times 10^{-15}$ | 0.0<br>0.0   | $-1.19 \times 10^{-15}$<br>$7.03 \times 10^{-16}$  | 1.4<br>$7.91 \times 10^{-1}$                      | $1.84 \times 10^{-7}$<br>$7.15 \times 10^{-8}$   | $1.0468 \times 10^{-2}$<br>$9.08 \times 10^{-4}$  |
| $f_{11}$ | 0.0<br>0.0  | 0.0<br>0.0   | 0.0<br>0.0   | $2.35 \times 10^{-2}$<br>$3.54 \times 10^{-2}$    | $9.23 \times 10^{-2}$<br>$3.41 \times 10^{-1}$   | $4.63669 \times 10^{-3}$<br>$3.96 \times 10^{-3}$ |
| $f_{12}$ | $1.787 \times 10^{-21}$<br>$5.06 \times 10^{-23}$     | $1.77 \times 10^{-21}$<br>$7.21 \times 10^{-24}$   | 0.0<br>0.0   | $3.819 \times 10^{-1}$<br>$8.4 \times 10^{-1}$    | $8.559 \times 10^{-3}$<br>$4.79 \times 10^{-2}$  | $4.56 \times 10^{-6}$<br>$8.11 \times 10^{-7}$    |
| $f_{13}$ | $1.702 \times 10^{-21}$<br>$4.0628 \times 10^{-23}$   | $1.686 \times 10^{-21}$<br>$1.149 \times 10^{-24}$ | $-1.142824$<br>$4.45 \times 10^{-8}$               | $-5.9688 \times 10^{-1}$<br>$5.17 \times 10^{-1}$ | $-9.626 \times 10^{-1}$<br>$5.14 \times 10^{-1}$ | $-1.142742$<br>$1.34 \times 10^{-5}$              |

**Table 4: Performance Comparison among opt-IMMALG, DE, PSO, arPSO and SEA on the first 13 functions with dimension 100.**

|          | opt-IMMALG   |  | DE  | PSO   | arPSO   | SEA  |
|----------|--|--|---|---|---|--|
|          | $\alpha = \left(\frac{1}{\rho}\right) e^{(-f)}$    | $\alpha = e^{(-\rho * f)}$                           |   |   |   |  |
|          | Mean Best<br>Std Dev                               | Mean Best<br>Std Dev                                 | Mean Best<br>Std Dev                              | Mean Best<br>Std Dev                              | Mean Best<br>Std Dev                              | Mean Best<br>Std Dev                             |
| $f_1$    | 0.0<br>0.0   | 0.0<br>0.0   | 0.0<br>0.0  | 0.0<br>0.0  | $7.4869 \times 10^{+2}$<br>$2.31 \times 10^{+3}$  | $5.229 \times 10^{-4}$<br>$5.18 \times 10^{-5}$  |
| $f_2$    | 0.0<br>0.0   | 0.0<br>0.0   | 0.0<br>0.0  | $1.804 \times 10^{+1}$<br>$6.52 \times 10^{+1}$   | $3.9637 \times 10^{+1}$<br>$2.45 \times 10^{+1}$  | $1.737 \times 10^{-2}$<br>$9.43 \times 10^{-4}$  |
| $f_3$    | 0.0<br>0.0   | 0.0<br>0.0   | $5.87 \times 10^{-10}$<br>$1.83 \times 10^{-10}$  | $3.666 \times 10^{+3}$<br>$6.94 \times 10^{+3}$   | $1.817 \times 10^{+1}$<br>$2.50 \times 10^{+1}$   | $3.68 \times 10^{-2}$<br>$6.06 \times 10^{-3}$   |
| $f_4$    | $7.32 \times 10^{-4}$<br>$2.109 \times 10^{-3}$    | $6.447 \times 10^{-7}$<br>$3.338 \times 10^{-6}$     | $1.128 \times 10^{-9}$<br>$1.42 \times 10^{-10}$  | 5.312<br>$8.63 \times 10^{-1}$                    | 2.4367<br>$3.80 \times 10^{-1}$                   | $7.6708 \times 10^{-3}$<br>$5.71 \times 10^{-4}$ |
| $f_5$    | 97.02<br>54.73                                     | 74.99<br>38.99                                       | 0.0<br>0.0  | $2.02 \times 10^{+2}$<br>$7.66 \times 10^{+2}$    | $2.36 \times 10^{+2}$<br>$1.25 \times 10^{+2}$    | $9.249 \times 10^{+1}$<br>$1.29 \times 10^{+1}$  |
| $f_6$    | 0.0<br>0.0   | 0.0<br>0.0   | 0.0<br>0.0  | 2.1<br>3.52                                       | $4.118 \times 10^{+2}$<br>$4.21 \times 10^{+2}$   | 0.0<br>0.0                                       |
| $f_7$    | $1.763 \times 10^{-5}$<br>$2.108 \times 10^{-5}$   | $1.59 \times 10^{-5}$<br>$3.61 \times 10^{-5}$       | $7.664 \times 10^{-3}$<br>$6.58 \times 10^{-4}$   | $2.784 \times 10^{-2}$<br>$7.31 \times 10^{-2}$   | $3.23 \times 10^{-3}$<br>$7.87 \times 10^{-4}$    | $7.05 \times 10^{-4}$<br>$9.70 \times 10^{-5}$   |
| $f_8$    | $-4.176 \times 10^{+4}$<br>$2.08 \times 10^{+2}$   | $-4.16 \times 10^{+4}$<br>$2.06 \times 10^{+2}$      | $-4.1898 \times 10^{+4}$<br>$1.06 \times 10^{-3}$ | $-2.1579 \times 10^{+4}$<br>$1.73 \times 10^{+3}$ | $-2.1209 \times 10^{+4}$<br>$2.98 \times 10^{+3}$ | $-3.943 \times 10^{+4}$<br>$5.36 \times 10^{+2}$ |
| $f_9$    | 0.0<br>0.0   | 0.0<br>0.0   | 0.0<br>0.0  | $2.4359 \times 10^{+2}$<br>$4.03 \times 10^{+1}$  | $4.809 \times 10^{+1}$<br>9.54                    | $9.9767 \times 10^{-2}$<br>$3.04 \times 10^{-1}$ |
| $f_{10}$ | $1.18 \times 10^{-16}$<br>$6.377 \times 10^{-16}$  | 0.0<br>0.0   | $8.023 \times 10^{-15}$<br>$1.74 \times 10^{-15}$ | 4.49<br>1.73                                      | $5.628 \times 10^{-2}$<br>$3.08 \times 10^{-1}$   | $2.93 \times 10^{-3}$<br>$1.47 \times 10^{-4}$   |
| $f_{11}$ | 0.0<br>0.0   | 0.0<br>0.0   | $5.42 \times 10^{-20}$<br>0.0                     | $4.17 \times 10^{-1}$<br>$6.45 \times 10^{-1}$    | $8.53 \times 10^{-2}$<br>$2.56 \times 10^{-1}$    | $1.89 \times 10^{-3}$<br>$4.42 \times 10^{-3}$   |
| $f_{12}$ | $5.34 \times 10^{-22}$<br>$9.81 \times 10^{-24}$   | $5.3169 \times 10^{-22}$<br>$5.0655 \times 10^{-24}$ | 0.0<br>0.0  | $1.77 \times 10^{-1}$<br>$1.75 \times 10^{-1}$    | $9.219 \times 10^{-2}$<br>$4.61 \times 10^{-1}$   | $2.978 \times 10^{-7}$<br>$2.76 \times 10^{-8}$  |
| $f_{13}$ | $1.712 \times 10^{-21}$<br>$9.379 \times 10^{-23}$ | $1.689 \times 10^{-21}$<br>$9.877 \times 10^{-24}$   | $-1.142824$<br>$2.74 \times 10^{-8}$              | $-3.86 \times 10^{-1}$<br>$9.47 \times 10^{-1}$   | $3.301 \times 10^{+2}$<br>$1.72 \times 10^{+3}$   | $-1.142810$<br>$2.41 \times 10^{-8}$             |