An Artificial Immune Network for Multimodal Function Optimization on Dynamic Environments

Fabricio Olivetti de França
LBiC/DCA/FEEC
State University of Campinas (Unicamp)
PO Box 6101, 13083-970
Campinas/SP, Brazil
Phone: +55 19 3788-3885
olivetti@dca.fee.unicamp.br

Fernando J. Von Zuben
LBiC/DCA/FEEC
State University of Campinas (Unicamp)
PO Box 6101, 13083-970
Campinas/SP, Brazil
Phone: +55 19 3788-3885
vonzuben@dca.fee.unicamp.br

Leandro Nunes de Castro
Research and Graduate Program in Computer Science,
Catholic University of Santos, Brazil, Phone/Fax: +55 13 3226 0500
lnunes@unisantos.br

ABSTRACT
Multimodal optimization algorithms inspired by the immune system are generally characterized by a dynamic control of the population size and by diversity maintenance along the search. One of the most popular proposals is denoted opt-aiNet (artificial immune network for optimization) and is extended here to deal with time-varying fitness functions. Additional procedures are designed to improve the overall performance and the robustness of the immune-inspired approach, giving rise to a version for dynamic optimization, denoted dopt-aiNet. Firstly, challenging benchmark problems in static multimodal optimization are considered to validate the new proposal. No parameter adjustment is necessary to adapt the algorithm according to the peculiarities of each problem. In the sequence, dynamic environments are considered, and usual evaluation indices are adopted to assess the performance of dopt-aiNet and compare with alternative solution procedures available in the literature.

Categories and Subject Descriptors
I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search – *heuristic methods.*

General Terms: Algorithms.

Keywords: Immune network, opt-aiNet, multimodal optimization, dynamic optimization.

1. INTRODUCTION
Among the many possible applications of artificial immune systems (AIS) [6], optimization has been receiving particular attention over the last few years. Within optimization, the emphasis has been on the use of AIS to perform multimodal and dynamic function optimization [5],[7],[8],[10],[15]-[17].

The immune system of higher animals (e.g., humans) has a natural capacity of dealing with complex dynamic environments in which multiple disease-causing agents are trying to break through the body barriers and promote damage. Based on some of the biological immune mechanisms of host defense, simple, evolutionary-like immune algorithms have been devised and studied in the context of global, multimodal and dynamic optimization [5], [7], [8], [10], [15]-[17].

Despite these seminal works, a broader investigation of the usefulness of immune algorithms in the context of multimodal and dynamic function optimization has not been performed yet, and the results available in the literature are still limited. The present paper corroborates with all the previous works cited above in the direction of helping to assess the performance and usefulness of AIS for optimization. In particular, the optimization version of an immune network model, called opt-aiNet [7], is improved and extended to deal with multimodal, dynamic environments.

The modified algorithm, termed here dopt-aiNet (opt-aiNet for dynamic environments) is applied to two sets of problems. First, a large set with 18 numeric benchmark functions, commonly used by the evolutionary computation (EC) community, is used to assess the potential of the approach to find global optimal solutions in spaces of very high dimension. Then, the same version of the dopt-aiNet algorithm is applied to a set of dynamic functions. In both cases the results are directly compared with results from the EC community and those of other immune algorithms.

This paper is organized as follows. Section 2 presents the original opt-aiNet [7]. Section 3 develops new concepts to improve its capabilities on broader domains still maintaining its efficiency on higher dimensions, and without losing its main feature of identifying multiple solutions. In Section 4 it is shown several experimental results and test cases to demonstrate the performance of the algorithm on solving benchmark problems. Section 5 shows the application to dynamic environments without any changes on the main algorithm structure. Section 6 concludes the paper and gives further remarks.

2. THE OPT-AINET ALGORITHM
The opt-aiNet adaptation procedure was first conceived by de Castro and Timmis [7] as a multimodal function optimization algorithm developed by taking inspiration from some evolutionary properties of the human immune system. It is basically a mutation-based evolutionary search procedure with a population with dynamic size allo-

cation. Each individual of the population corresponds to a cell in the immune system, and is encoded as a real-valued vector in an Euclidean shape-space [13]. Each cell generates a certain number of clones (identical offspring), which are subjected to a mutation rate inversely proportional to the fitness of the parent cell [1]. From time to time new cells can be generated at random and the affinity of every cell with each other, that is, the measure of how similar they are, is indirectly calculated based on their Euclidean distance, and those that are too similar to each other are left out of the next generation. The original opt-aiNet algorithm, as introduced by de Castro and Timmis [7], is summarized in the pseudocode below. The algorithm requires as input parameters the number of clones to be generated (Nc), the initialization range, the suppression threshold (σ_s), and the function to be optimized (*f*).

```
Function [C] = opt-aiNet(Nc,range,σs,f)
C = random(range)
While stopping criterion is not met do
        fit = f(C)
        C' = clone(C,Nc)
        C' = mutate(C',fit*)
        best_clone = best(C')
        If best_clone is better than c then
                c = best_clone
        End
        Avg = average(f(C))
        If the average error doesn't stagnate then
                return to the beginning of the loop
        else
                suppress(C)
                C = [C; random(range)]
        End
End
```

Algorithm 1. Original opt-aiNet algorithm.

At the beginning, Nc cells are generated at random on the given domain range. The main loop starts, and after the population is evaluated each individual generates Nc clones that are mutated following Eq. 1.

$$c' = c + \alpha N(0,1), \qquad \alpha = (1/\beta)\exp(-f^*). \tag{1}$$

where c' is a mutated cell, $N(0,1)$ is a Gaussian random variable of mean zero and standard deviation $\sigma = 1$, β is a parameter that controls the decay of the inverse exponential function, and f^* is the fitness of an individual normalized in the interval [0,1].

After every clone is mutated, the highest fitness cell within each clone survives to the next iteration. Whenever the algorithm stagnates, that is, when the average error does not improve significantly after an iterative step, the affinity among cells is calculated and those cells with affinity less than a pre-specified suppression threshold σ_s are eliminated from the population, thus avoiding redundancy. Then, new cells are generated randomly and introduced into the population.

3. DOPT-AINET: A MODIFIED OPT-AINET

In the original proposal of opt-aiNet [7], the authors applied the algorithm to a number of simple bi-dimensional functions with multiple optimal solutions. Although the performance was good on these simple functions, some problems have been identified in the original algorithm [15],[16], such as the need to perform a high number of function evaluations to find a good set of solutions. After some experimentation with dynamic environments, we also detected some aspects to be improved in the algorithm. In summary, four modifications are proposed here: 1) the use of a separate memory subpopulation; 2) a line search procedure to optimize and thus automatically set β; 3) two new mutation operator schemes; 4) a cell line suppression mechanism; and 5) a limited population size. These modifications are described in the following.

3.1 Current and Memory Subpopulations

To avoid a performance decay due to an excessive population growth, two separated populations were created: 1) the *current subpopulation*, which corresponds to the population used on the original algorithm and in which newly created cells are placed; 2) and the *memory subpopulation* [5], which corresponds to those cells that have not evolved during a certain period of time, and thus will be assumed to have converged to local optimal solutions. To determine if a cell should go to the memory subpopulation, i.e., should become a memory cell, each cell receives a rank value that decrements whenever a mutation does not improve its objective function value, and increment otherwise. When this rank reaches zero, the cell is then moved to the memory subpopulation, where it will be treated with special mutation operators to be described bellow. Additionally, when a cell becomes a memory cell, it receives a new rank that follows this same process, but when this rank reaches zero the memory cell does not suffer any mutation whatsoever.

3.2 Line Search

The traditional mutation applied to the algorithm requires a parameter β (Eq. 1) that is a user-defined step size of the direction determined by the Gaussian random vector. This parameter sometimes requires a pre-analysis of the function landscape in order to be set properly. A very small value may lead to slow convergence, and, by contrast, a very high value may lead to a situation where the resultant mutated cell never converges to an optimum solution. Therefore, to take the best of each mutation for each Gaussian random vector generated, a line search procedure, called *golden section* [4], is performed to choose the best step size possible (β).

The golden section method basically divides the search space into two and determines which area is most promising, sub-dividing it and thus generating new search intervals. The process repeats until this uncertainty interval reaches a given threshold length. The key point is how it divides the search space using a ratio called *golden number* or *golden ratio*, which is a proportion number found on many geometric figures and nature structures.

This procedure is guaranteed to reach a global optimum solution in a certain direction given that the interval being searched has no discontinuity, is convex and unimodal. These restrictions stand as a problem on the use of this method together with opt-aiNet, since the algorithm does not have any information about the function to be optimized. To partially solve this problem, the initial segment to be searched can be divided into four new segments and the golden section method applied to each of these segments separately; the best result found is used as β, thus reducing the chance of failure. Even if this procedure fails to find an optimal β, this will still probably lead to a better value than the static parameter on the original algorithm and also, as there is a population of cells, an eventual fail on this search may not lead to total failure on the whole process. This procedure was chosen mainly because it has a fast convergence

rate and does not require any information about the search space; a pre-requisite when the algorithm is run for every mutation operation.

3.3 New Mutation Operators

While testing the simple Gaussian mutation proposed by de Castro and Timmis [7], it was possible to note that a few network cells sometimes converged to a value that is not a peak (local optima) and that could not be further improved by this type of mutation. Although an optimal search of β does alleviate this problem, two new mutation operators are proposed and applied to both populations.

3.3.1 One-Dimensional Mutation

The one-dimensional mutation (Algorithm 2) performs similarly to the traditional Gaussian mutation but only for one direction at a time, thus making a finer search on the area surrounding the cell. Additionally, it is performed the search on the unitary vectors **1** (vectors in which every element is 1) and **–1**. This method has the drawback of presenting a slow convergence for high dimensional spaces. Algorithm 2 describes the one-dimensional mutation, where D is a matrix of size $(n+2) \times n$ containing an Identity Matrix of size n, a row filled with 1's and another row filled with –1's. Parameter β is the step size obtained via golden section (Section 3.2).

```
Function [C] = one-dimensional(C,f)
D = [identity(n); 1; -1]
For each vector element "c" of matrix C do,
        For each row "d" of matrix D do,
                % calculate the step size β that
                optimizes the cell in direction d
                beta = golden_section(c,d,f)
                c' = c + d*β
                If c' is better than c then
                        c = c'
                End
        End
End
```

Algorithm 2. One-dimensional mutation.

3.3.2 Gene Duplication

In nature, a process called *gene duplication*, where some genes are duplicated during the process of chromosome reading, sometimes occur. This is known to play an important role in the evolution of species [9], [12].

A new mutation based on these observations was developed, where a randomly chosen element (coordinate) x_i is copied into another element x_j whenever it minimizes the objective function (minimization problem). Algorithm 3 describes the operator.

```
Function [x] = gene_duplication(x)
Dup = x[rand(1,n)]
x' = x;
For each element of x do
        x'[j] = Dup;
        If x' is better than  x then
                x[j] = x'[j]
        Else
                x'[j] = x[j]
        End
End
```

Algorithm 3. Gene duplication.

The two new mutations proposed work better if the one-dimensional mutation precedes gene duplication, because the former results in an improvement of at least one element of the vector, and the latter can benefit from this improvement.

3.4 Cell Line Suppression

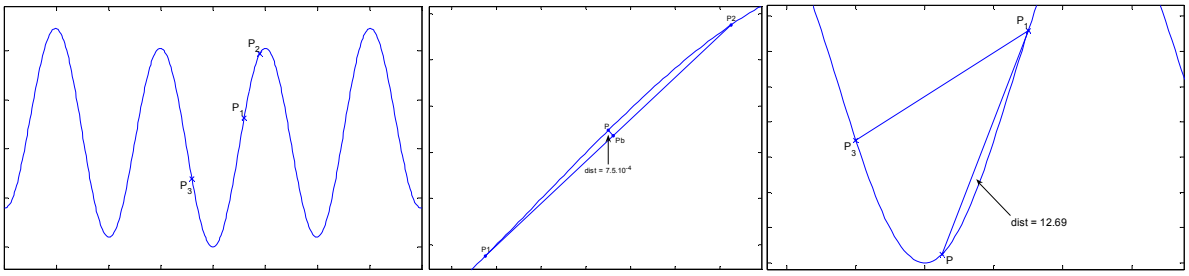
The use of Euclidean distance to indirectly measure the affinity among cells in opt-aiNet may require some pre-analysis of the function to be optimized in order to adjust the threshold parameter (σ_s). In some cases, it is very hard to determine an appropriate threshold value. A new algorithm is then proposed to decrease the probability of having more than one cell located at each peak of the fitness landscape, thus reducing redundancy. The cell line suppression algorithm is described as follows (see Figure 1)

```
Function [X] = cell_line_suppress(X, σs)
  For each pair of cells x1,x2 do
        P1 = [x1 f(x1)]
        P2 = [x2 f(x2)]
        P = [x1+0.5(x2-x1)  f(x1+0.5(x2-x1))]
        v = P2 – P1
        w = P – P1
        c1 = w.v  %dot product of vectors w and v
        c2 = |v|  %norm of vector v
        b = c1/c2  %projection of P on P2P1
        If c1 <= 0, % nearest point is on P1
                d = dist(P,P1)
        Else If c2 <= c1,  %nearest point is on P2
                d = dist(P,P2)
        Else,    %nearest point is at the point
                 Pb where PPb ⊥ P2P1
                Pb = P1 + b.v
                d = dist(P,Pb)
        End
        If d < σs then
             eliminate the cell with worst objec-
        tive function value.
        End

End
```

Algorithm 4. Cell line suppression.

First the line segment $\overline{P_1P_2}$ is built from $[x_1\ f(x_1)]$ and $[x_2\ f(x_2)]$, its middle point is taken and its corresponding function value is calculated forming point P = [(x1+x2)/2 f((x1+x2)/2)]. Next, the direction vectors **v** and **w** of lines $\overline{P_1P_2}$ and $\overline{PP_1}$, respectively, are calculated. Then, to measure the distance from point P to the line segment $\overline{P_1P_2}$, it is first calculated the nearest point from P to P_1P_2, that is done by projecting this point into the line segment. If this projection is outside the line segment, then either the point P_1 (when $c_1 \leq 0$) or the point P_2 (when $c_1/c_2 \geq 1$) is taken, whatever is closer. After that, the Euclidean distance from these two points is calculated and if it is less than the threshold σ_s, one of the two cells are eliminated. This threshold is much easier to set, since the difference from the distance of cells that must be eliminated and those that must not is too high. Experimentally a value $\sigma_s = 0.5$ is a good suggestion.



(a) (b) (c)

Figure 1. Cell Line Suppression example. In (a) three points must be evaluated, (b) P_1 and P_2 hold a large similarity (small distance value) and the distance is measured by the projection point P_b, while in (c) P_1 and P_3 are too far apart (large distance value) and on this case the projection point is outside the line segment, so the point P_1 is taken to measure the distance as it is the nearest point between P_1 and P_3.

3.5 Limited Population

The last modification proposed was developed to deal with functions with too many optimal solutions. In such cases, as the algorithm tends to find every possible optimum, the population expands along the iterations and may cause an excessive memory usage, resulting in an increase of convergence time. To avoid this, a maximum number of cells is pre-defined. When the population of cells reaches this number, a percentage of the worst fitness cells is eliminated from the current subpopulation.

The resultant dopt-aiNet is summarized in Algorithm 5.

```
Function [C] = dopt-aiNet(Nc,range,σs,f,max_cells)
C = random(range)
While stopping criterion is not met do
        fit = f(C)
        C' = clone(C,Nc)
        C' = mutate(C',f)
        For each cell c from C do,
                If c' is better than c,
                        c.rank = c.rank + 1
                        c = c'
                Else
                        c.rank = c.rank - 1
                End
                If c.rank == 0,
                        Mem = [Mem, c]
                End
        End
        % These two mutations are only applied to
          cells with rank values greater than zero
        C = one-dimensional(C,f)
        C = gene_duplication(C,f)
        Mem = one-dimensional(Mem,f)
        Mem = gene_duplication(Mem,f)
        For each cell m from Mem do,
                If m has improved,
                        m.rank = m.rank + 1
                Else
                        m.rank = m.rank - 1
                End
        End
        Avg = average(f(C))
        If the average error does not stagnate
                return to the beginning of the loop
        else
                cell_line_suppress(C, σs)
                C = [C; random(range)]
        End
        If size(C) > max_cells,
                suppress_fitness(C)
        End
End
```

Algorithm 5. dopt-aiNet: an extended version of opt-aiNet for optimization in dynamic environments.

4. NUMERICAL EXPERIMENTS

To assess the performance of dopt-aiNet for solving global and multimodal optimization problems, some initial experiments were conducted with a large number of multi-dimensional static functions. A good performance on static benchmark functions not only validates the proposal in these circumstances, but also gives an indication that the algorithm may react with sufficient speed so as to cope with dynamic environments. The second set of experiments was thus performed taking into account time-varying functions. In both cases the results of dopt-aiNet were directly compared with those found in the literature.

4.1 Static Environments

The application of dopt-aiNet to static environments was performed by applying it to 18 numeric functions with varying degrees of complexity. The first 11 functions were taken from Leung and Wang [11], and the last 7 functions were taken from Timmis et al. [16]. The performance will be compared with those provided by the algorithms discussed in the cited references. Two aspects are taken into account for comparison: 1) the capability of finding the global optimal solution; and 2) the average number of function evaluations to find the optima. Discussions about the number of optimal solutions found are also included.

$$f_1 = \sum_{i=1}^{N} \left(-x_1 \sin\left(\sqrt{|x_i|} \right) \right) \tag{2}$$

$$f_2 = \sum_{i=1}^{N} \left(x_i^2 - 10\cos(2\pi x_i) + 10 \right) \tag{3}$$

$$f_3 = -20\exp\left(-0.2\sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2} \right) - \exp\left(\frac{1}{N}\sum_{i=1}^{N} \cos(2\pi x_i) \right) \tag{4}$$

$$f_4 = \frac{1}{4000}\sum_{i=1}^{N} x_i^2 - \prod_{i=1}^{N} \cos\left(\frac{x_i}{\sqrt{i}} \right) + 1 \tag{5}$$

$$f_5 = -\sum_{i=1}^{N} \sin(x_i)\sin^{20}\left(\frac{i \times x_i^2}{\pi} \right) \tag{6}$$

$$f_6 = \frac{1}{N}\sum_{i=1}^{N} \left(x_i^4 - 16x_i^2 + 5x_i \right) \tag{7}$$

$$f_7 = \sum_{i=1}^{N-1} \left(100(x_i - x_{i+1})^2 + (x_i - 1)^2 \right) \tag{8}$$

$$f_8 = \sum_{i=1}^{N} x_i^2 \tag{9}$$

$$f_9 = \sum_{i=1}^{N} |x_i| + \prod_{i=1}^{N} |x_i| \quad (10)$$

$$f_{10} = \sum_{i=1}^{N} \left(\sum_{j=1}^{i} x_j \right)^2 \quad (11)$$

$$f_{11} = \max\{|x_i|, i = 1,2,...,N\} \quad (12)$$

$$f_{12} = 2(x - 0.75)^2 + \sin(5\pi x - 0.4\pi) - 0.125 \quad (13)$$

$$f_{13} = -\sum_{j=1}^{5} j \sin((j+1)x + j) \quad (14)$$

$$f_{14} = (x_2 - \frac{5.1}{4\pi^2} x_1^2 + \frac{5}{\pi} x_1 - 6)^2 + 10(1 - \frac{1}{8\pi}) \cos(x_1) + 10 \quad (15)$$

$$f_{15} = \sum_{j=1}^{5} j \sin((j+1)x_1 + j). \sum_{j=1}^{5} j \sin((j+1)x_2 + j) + 0.5[(x_1 + 1.4513)^2 + (x_2 + 0.80032)^2] \quad (16)$$

$$f_{16} = \sum_{j=1}^{5} j \sin((j+1)x_1 + j). \sum_{j=1}^{5} j \sin((j+1)x_2 + j) + [(x_1 + 1.4513)^2 + (x_2 + 0.80032)^2] \quad (17)$$

$$f_{17} = \frac{x_1^4}{4} - \frac{x_1^2}{2} + \frac{x_1}{10} + \frac{x_2^2}{2} \quad (18)$$

$$f_{18} = \sum_{j=1}^{5} j \sin((j+1)x_1 + j). \sum_{j=1}^{5} j \sin((j+1)x_2 + j) \quad (19)$$

The initialization range and the dimension adopted for each of the first 11 functions follow Leung and Wang [11] (Table 1).

For each function 1,000 iterations of dopt-aiNet were run in 30 independent experiments and the results compared with the following methods (see [11] and [16] for details):

Table 1. Initialization range for the benchmark functions.

Function	Initialization Range	Problem Dimension (N)
f_1	$[-500, 500]^N$	30
f_2	$[-5.12, 5.12]^N$	30
f_3	$[-32, 32]^N$	30
f_4	$[-600, 600]^N$	30
f_5	$[0, \pi]^N$	30
f_6	$[-5, 5]^N$	100
f_7	$[-5, 10]^N$	30
f_8	$[-100, 100]^N$	30
f_9	$[-10, 10]^N$	30
f_{10}	$[-100, 100]^N$	30
f_{11}	$[-100, 100]^N$	30

- *Orthogonal Genetic Algorithm with Quantization* (OGA/Q) [11]: an enhanced GA with quantization.
- *Conventional Genetic Algorithm* (CGA): the ordinary genetic algorithm used for comparison with OGA/Q.
- *Fast Evolution Strategy* (FES) [18]: evolution strategy with Cauchy mutation.
- *Enhanced Simulated Annealing* (ESA) [14]: a simulated annealing where large steps are used on high temperatures and small steps at low temperatures.
- *Particle Swarm Optimization* (PSO) [2]: a swarm based metaheuristic.
- *Evolutionary Optimization* (EO) [2]: following the same framework of evolutionary algorithms, it uses only mutation and selection to evolve a population.
- *B-Cell Algorithm* (BCA) [16]: another immune system based algorithm, but this one has a fixed population size, bit-string representation and uses a contiguous somatic hypermutation.
- *Hybrid GA algorithm* (HGA) [16]: a genetic algorithm with hybrid search technique to help improving the solutions.

Tables 2, 3 and 4 summarize the performance of the algorithms when applied to some of the benchmark functions from Leung and Wang [11].

Table 2. Performance comparison between dopt-aiNet and opt-aiNet when applied to four benchmark functions from [11].

Function	Known Global Value	Mean Objective Function Value		Mean no. of Function Evaluations ± std	
		dopt-ainet	opt-aiNet	dopt-ainet	opt-aiNet
f_2	0	**0**	153.54±13.58	3379.3±1040.8	5500000
f_4	0	**0**	340±61.94	7276±2072.5	5500000
f_7	0	**0**	0.2192±0.085	81296±5801.8	5500000
f_8	0	**0**	**0**	6182.6±1693.4	3109986±362220

Table 3. Performance comparison among dopt-aiNet, OGA/Q and CGA when applied to eleven benchmark functions from [11].

Function	Known Global Value	Mean Objective Function Value			Mean no. of Function Evaluations ± std		
		dopt-ainet	OGA/Q	CGA	dopt-ainet	OGA/Q	CGA
f_1	−12569.5	**−18286**	−12569.5	−8444.75	4168.7±4250.9	302166	458653
f_2	0	**0**	**0**	22.97	3379.3±1040.8	224710	335993
f_3	0	**0**	4.440×10^{-6}	2.69	5563.7±1112.3	112421	336481
f_4	0	**0**	**0**	1.26	7276±2072.5	134000	346971
f_5	−99.27	**−99.27**	−92.83	−83.27	2318.7±1901.4	302773	338417
f_6	−78.33	**−78.33**	−78.30	−59.05	428460±34992	245930	268286
f_7	0	**0**	0.752	150.79	81296±5801.8	167863	1651448
f_8	0	**0**	**0**	4.96	6182.6±1693.4	112559	181445
f_9	0	**0**	**0**	0.79	406150±22774	112612	170955
f_{10}	0	**0**	**0**	18.83	10113±3050.1	112576	203143
f_{11}	0	**0**	**0**	2.62	119840±6052.8	112893	185373

Table 4. Performance of dopt-aiNet, FES and ESA when applied to some benchmark functions from [11].

Function	Known Global Value	Mean Objective Function Value					Mean no. of Function Evaluations ± std				
		dopt-ainet	FES	ESA	PSO	EO	dopt-ainet	FES	ESA	PSO	EO
f_1	−12569.5	**−18286**	−12556.4	---	---	---	4168.7±4250.9	900030	---	---	---
f_2	0	**0**	0.16	---	47.1345	46.4689	3379.3±1040.8	500030	---	250000	250000
f_3	0	**0**	0.012	---	---	---	5563.7±1112.3	150030	---	---	---
f_4	0	**0**	0.037	---	0.4498	0.4033	7276±2072.5	200030	---	250000	250000
f_6	−78.33	**−78.33**	---	---	11.175	9.8808	428460±34992			250000	250000
f_7	0	**0**	---	17.1	---	---	81296±5801.8	---	188227	---	---

Table 5. Performance of dopt-aiNet, opt-aiNet, BCA and HGA when applied to the last 7 benchmark functions from [16].

Function	Known Global Value	Mean Objective Function Value				Mean no. of Function Evaluations ± std			
		dopt-ainet	opt-aiNet	BCA	HGA	dopt-ainet	opt-aiNet	BCA	HGA
f_{12}	−1,12	−1,12	−1,12	−1,08±04	−1,12	103.4±26.38	6717±538	3016±2252	6081±4471
f_{13}	−12,06	−12,06	−12,03	−12,03	−12,03	110±0	41419±25594	1219±767	3709±2397
f_{14}	0,4	0,4	0,39	0,4	−0,4	302.4±99.19	6346±4656	4921±31587	30583±28378
f_{15}	−186,73	−186,73	−180,83	−186,73	−186,73	1742.7±1412.3	363528±248161	46433±31587	78490±6344
f_{16}	−186,73	−186,73	−173,16	−186,73	−186,73	1227.6±976.21	346330±255980	426360±32809	76358±11187
f_{17}	−0,35	−0,35	−0,26	−0,91	0,99	442.8±141.82	54703±29701	2862±351	12894±9235
f_{18}	−186,73	−186,73	−186,73	−186,73	−186	349.2±67.15	50875±45530	14654±5277	52581±19095

Table 6. Number of peaks found by dopt-aiNet in 1000 iterations.

Function	Number of peaks found
f_1	20
f_2	18
f_3	33
f_4	40
f_5	20
f_6	23
f_7	20
f_8	22
f_9	20
f_{10}	20
f_{11}	30

Table 5 compares the performance of dopt-aiNet with opt-aiNet (as presented by Timmis et al. [16]), BCA and HGA on the last 7 functions used by Timmis et al. [16]. Table 6 depicts the number of local optima found by dopt-aiNet, stressing its capability of locating multiple optimal solutions. As can be observed, dopt-aiNet presented a good performance, both in terms of number and quality of solutions, on most problems. Most works from the literature do not present the standard deviation for the algorithms discussed. However, it is possible to infer that even the worst case of dopt-aiNet is superior to the mean of most results from the literature.

4.2 Dynamic Environments

Motivated by the good results obtained for the static benchmark problems, dopt-aiNet was tested on dynamic environments. There are some other recent papers that successfully applied immune algorithms on time-varying problems [8],[10],[17]. However, in all cases the test functions used were simpler than the ones to be investigated here.

The tests were made based on one of Angeline's experiments [3], in which the optima suffer a displacement at every n iterations, as specified. In Angeline [3], three movements (linear, circular and Gaussian) and two adjusting parameters (τ and f) were defined, the amount of displacement (τ) and the update frequency (f), that means that at every f iterations the function is updated.

The linear movement displaces the optimum by a constant rate at every iteration update, simply adding Δk to each variable. The update rule for Δk is as follows:

$$\Delta k = \Delta k + \tau \tag{20}$$

where τ is the amount of displacement.

The circular movement displaces the optimum in cycles of 25 units of time regarding update:

$$\begin{aligned} \Delta k &= \Delta k + \tau \mathbf{sin} \frac{2.\pi.t}{25} \quad \text{for } k \text{ even} \\ \Delta k &= \Delta k + \tau \mathbf{cos} \frac{2.\pi.t}{25} \quad \text{for } k \text{ odd} \end{aligned} \tag{21}$$

where t is the number of times the function was displaced so far.

Finally, the Gaussian movement displaces the optimum at a Gaussian random rate, updating Δk as follows:

$$\Delta k = \Delta k + N(1,0) \tag{22}$$

As the optima keep moving along the iterations, there is no need for a memory subpopulation, as the cells will always have a direction to improve. Thus, the mutation operators described in Sections 3.3.1 and 3.3.2 were run at each iteration on the current subpopulation.

For this set of experiments, it were tested the linear, circular and Gaussian movements with $\tau = 0.1$ and update rate at every iteration. A total of 1,000 iterations were run to study the behavior of dopt-aiNet when dealing with a moving target on some of the functions studied above. It was calculated the maximum, minimum and average objective function values and the mean error; that is, the distance from the best current point to the optimum of each function.

Four functions were used in these experiments: Sphere, Rosenbrock, Rastrigin, and Griewank, as described below:

$$Sphere = \sum_{i=1}^{N} x_i^2 \tag{23}$$

$$Rosenbrock = \sum_{i=1}^{N-1} \left(100(x_i - x_{i+1})^2 + (x_i - 1)^2\right) \tag{24}$$

$$Griewank = \frac{1}{4000} \sum_{i=1}^{N} x_i^2 - \prod_{i=1}^{N} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{25}$$

$$Rastrigin = \sum_{i=1}^{N} \left(x_i^2 - 10\mathbf{cos}(2\pi x_i) + 10\right) \tag{26}$$

The initialization range and problem dimension are summarized in Table 7. All functions have the global optimum equals to 0.

Table 7. Initialization range for the dynamic functions.

Function	Initialization Range	Problem Dimension (N)
Rastrigrin	$[-5.12, 5.12]^N$	30
Griewank	$[-600, 600]^N$	30
Rosenbrock	$[-100, 100]^N$	30
Sphere	$[-1.28, 1.28]^N$	30

Table 8 summarizes the performance of dopt-aiNet when applied to the four dynamic problems described above, and Figure 2 illustrates the behavior of the algorithm during the iterative search. The numbers within parentheses in Table 8 correspond to the iteration when dopt-aiNet first identified the global optimum and the maximum, minimum, mean and error values corresponds to the period between this iteration to the end. Through the error value it can be inferred that after the global optimum is found the oscillation around this point is minimal indicating the capacity of the algorithm to pursue the moving target.

As can be observed from Figure 2, for all functions, with the exception of Rosenbrock, dopt-aiNet reached the region surrounding the global optimum very quickly and oscillated around it while the environment was moving. From the four functions tested, Rosenbrock was the one with higher learning time. This is because it has a large plateau near the global optimum, making the search more complicated at the neighborhood of the optimum.

Table 8. Max, min, and mean objective function value, together with the average error for 1,000 iterations.

Function		Linear	Circular	Gaussian
Sphere	Max	46.76	0.75	0.05
	Min	4.16×10^{-16}	8.32×10^{-16}	8.12×10^{-19}
	Mean	0.05±1.47	0.13±0.13	7.11×10^{-4}
	Error	0.02±0.22	0.32±0.18 (100)	0.02±0.02 (100)
Rosenbrock	Max	2.59	43.79	0.92
	Min	0.14	0.21	0.06
	Mean	0.74±0.58	8.49±7.44	0.14±0.19
	Error	0.50±0.17 (400)	0.57±0.24 (300)	0.22±0.17 (400)
Rastrigin	Max	39.56	74.05	55.11
	Min	9.4×10^{-9}	0	0
	Mean	0.35±1.47	16.05±15.46	0.48±2.62
	Error	0.03±0.16	0.38±0.58	0.03±0.16
Griewank	Max	1.48	0.02	3.8
	Min	0	0	0.01
	Mean	0.003±0.06	0.006±0.005	0.04±0.23
	Error	0.13±1.76	0.33±0.17 (100)	7.57±5.79

5. CONCLUSION

This work proposed several improvements to the opt-aiNet algorithm that not only helped to enhance its capabilities of quickly finding local optimal solutions and maintaining the cell diversity but

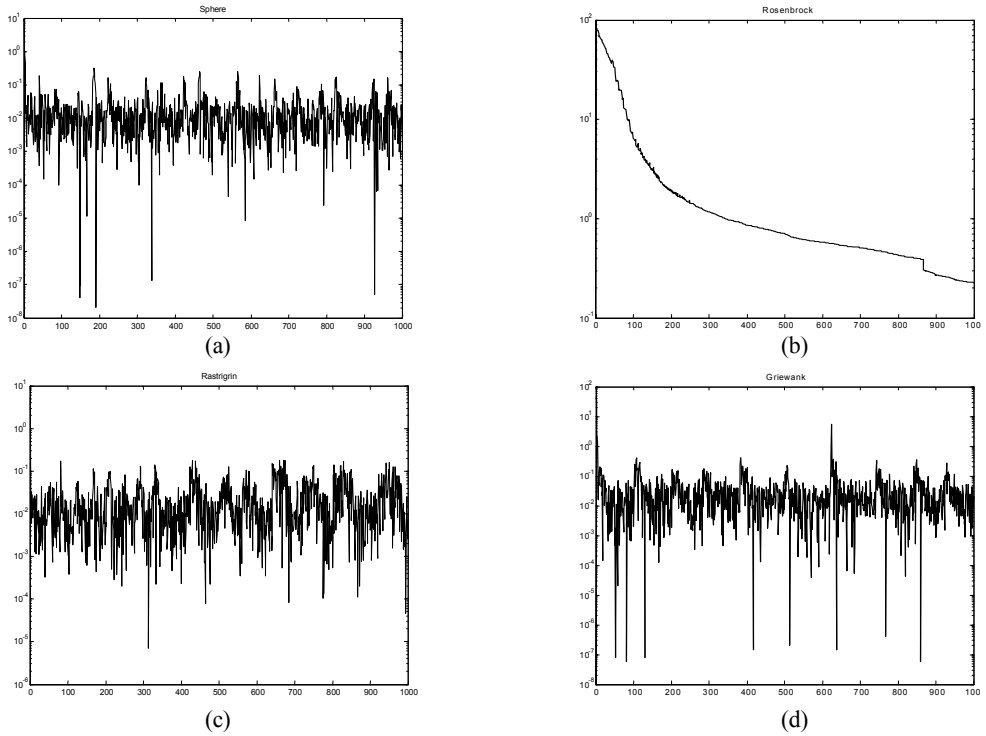


Figure 2. Performance of the dopt-aiNet on dynamic environments (iterations × log(fitness)). Linear displacement, with step size $\tau = 0.1$ and update frequency $f = 1$. (a) Sphere. (b) Rosenbrock. (c) Rastrigin. (d) Griewank.

also made it capable of dealing with dynamic environments (function with moving optima). The new procedures include: 1) the use of a modified golden section method as a local search procedure for an optimal mutation rate; 2) two new mutation operators to fine-tune each cell; and 3) a new suppress algorithm based on the approximation of a line between two points and the corresponding objective function curve, what provides a better measure of the affinity among cells.

These modifications were tested and showed to successfully solve several benchmark static problems with large dimensions. On dynamic environments, the behavior of the algorithm was also encouraging. In the most difficult situations, dopt-aiNet was able to track the optima while it changed its position through time.

For future research it must be studied a single suitable mutation to deal with tracking the optima once this is reached. Also some other types of dynamic environments must be investigated, for instance, when some optima disappear and others appear somewhere else, or when the intensity of peaks and valleys varies. Also, some tests should be performed on discrete optimization problems.

6. ACKNOWLEDGEMENTS

This work has been supported by grants from Fapesp, CNPq, and Capes.

7. REFERENCES

[1] Allen, D. *et al.*, *Timing, Genetic Requirements and Functional Consequences of Somatic Hypermutation During B-cell Development*, *Imm. Rev.*, 96, pp. 5-22, 1987.

[2] Angeline, P. J. *Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences*, in Proc. Evol. Prog. VII, V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, Eds. Berlin, Germany: Springer-Verlag, 1998, pp. 601–610.

[3] Angeline, P. J. *Tracking extrema in dynamic environments*. In Proceedings of the 6th International Conference on Evolutionary Programming (Indianapolis,Indiana, USA, Apr. 13-16 1997), P. J. Angeline, R. G. Reynolds, J. R. McDonnell, and R. C. Eberhart, Eds., vol. 1213 of Lecture Notes in Computer Science,Springer Verlag.

[4] Bazaraa, M. S., Sherali, H. D., and Shetty, C. M., *Nonlinear Programming: Theory and Algorithms*, 2nd Ed., Wiley, 1993.

[5] de Castro , L. N. & Von Zuben, F. J., *Learning and Optimization Using the Clonal Selection Principle* , IEEE Transactions on Evolutionary Computation, Special Issue on Artificial Immune Systems, **6** (3), pp. 239-251, 2002.

[6] de Castro, L. N. and Timmis, J. *Artificial Immune Systems: A New Computational Intelligence Approach*, Springer-Verlag.

[7] de Castro, L. N. and Timmis, J. *An Artificial Immune Network for Multimodal Function Optimization*. Proceedings of the IEEE Congress on Evolutionary Computation (CEC'02), Vol. 1, pp. 699-674, May, Hawaii, 2002.

[8] Gaspar, A., Collard, P. *From GAs to Artificial Immune Systems: Improving Adaptation in Time Dependent Optimization*. Proceedings of the Congress on Evolutionary Computation. pp. 1859-1866, Peter J. Angeline and Zbyszek Michalewicz and Marc Schoenauer and Xin Yao and Ali Zalzal (Eds).

[9] Holland, P. W. H., Garcia-Fernandez, J., Williams, N. A. and Sidow, A. (1994). *Gene duplications and origins of vertebrate development*. Development Supplement, pp. 125-133.

[10] Kelsey, J., Timmis, J., Hone, A. *Chasing Chaos*. In R. Sarker, R. Reynolds, H. Abbass, T. Kay-Chen, R. McKay, D. Essam, and T. Gedeon, editors. Proceedings of the Congress on Evolutionary Computation, pp. 413-419, Canberra. Australia, December. IEEE.

[11] Leung, Y. and Wang, Y. *An Orthogonal Genetic Algorithm with Quantization for Global Numerical Optimization*, IEEE Trans. Evol. Comput. Vol. 5, No. 1, pp. 41-53, 2001.

[12] Ohno, S. *Evolution by Gene Duplication*. Allen and Unwin, London, 1970

[13] Perelson, A. S., *Immune Network Theory*, *Imm. Rev.*, **110**, pp. 5-36, 1989.

[14] Siarry , P., Berthiau, G., Durbin, F. and Haussy, J. *Enhanced simulated annealing for globally minimizing functions of many-continuous variables*. Neural Networks, vol. 3, pp. 467-483, 1990.

[15] Timmis, J. and Edmonds, C., *A Comment on opt-AINet: An Immune Network Algorithm for Optimisation*, In D. Kalyanmoy et al, editor, Genetic and Evolutionary Computation, volume 3102 of Lecture Notes in Computer Science, pp. 308-317, Springer, 2004.

[16] 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.

[17] Walker, J., Garrett, S. *Dynamic Function Optimisation: Comparing the Performance of Clonal Selection and Evolutionary Strategies*. Lecture Notes in Computer Science 2787, pp. 273-284. Timmis, J., Bentley, P. and Hart, E.(Eds).

[18] Yao, X. and Liu, Y. *Fast evolution strategies*, in Evolutionary Programming VI, P. J. Angeline, R. Reynolds, J. McDonnell, and R. Eberhart, Eds. Berlin, Germany: Springer-Verlag, 1997, pp. 151–161.