Clonal Selection Algorithm With Binary Representation of Solutions for Non-Stationary Optimization Tasks

Krzysztof Trojanowski[1,2] and Grzegorz Matusik[1]

[1] Institute of Computer Science, University of Podlasie, Siedlce, Poland
[2] Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland

Abstract. Non-stationary optimization with the immune based algorithms is studied in this paper. The algorithm works with a binary representation of solutions. A set of different types of binary mutation is proposed and experimentally verified. The mutations differ in the way of calculation of the number of bits to be mutated. Obtained results allow to indicate the leading formulas of calculation.

1 Introduction

Heuristic algorithms inspired by the immune metaphor called clonal selection algorithms (CSA) have been already applied to non-stationary optimization tasks. The obtained results showed that they can be competitive with other evolutionary methods [6]. As in case of other evolutionary approaches CSA also need some additional modifications specific to non-stationary optimization. Usually these algorithms employ floating point representation of solutions and a set of perturbation operators specific to this type representation. The binary representation is hardly even used. Thus it was interesting to do some kind of review of binary mutations and evaluate efficiency of different strategies of such mutation. In this paper we selected one of the versions of clonal selection algorithms and proposed a suite of binary mutation operators. Two versions of the algorithm were tested. One is very close to the original while the other one is equipped with a few mechanisms selected from a set of typical extensions applied to non-stationary optimization tasks. As a test benchmark a MPB [3] generator was selected. These two versions of the clonal selection algorithm as well as a set of mutation operators are experimentally verified with the MPB benchmark in this paper. The results allow to answer the question if the binary representation can be successfully used in non-stationary optimization and what type of mutation operators should be used.

The paper consists of five sections. In Section 2 both versions of the clonal selection algorithm as well as the mutation operators are described. Section 3 briefly presents MPB and applied measure of offline error. Section 4 includes results of experiments while Section 5 – final conclusions.

2 Clonal Selection Algorithm

Our version of the clonal selection algorithm originates from [4]. The pseudo-code of the algorithm is given in Figure 1. The algorithm starts with a population of solutions randomly generated from the search space and performs the process of iterated improvement of the solutions by the execution of the main loop (Figure 1).

```
procedure clonal_selection_optimization

$FFE := 0$; $Nc := d * c$; $t := 0$;
Initialize $P_0(d) = \{\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_d\}$
Evaluate( $P_0(d)$ )
$FFE := FFE + d$;
while ($FFE < FFE_{\max}$) do {
    $P_t(Nc)$=Cloning( $P_t(d), c$ )
    $P_t^H(Nc)$=Hypermutation( $P_t(Nc)$ )
    Evaluate( $P_t(Nc)$ ); $FFE = FFE + Nc$
    Aging( $P_t(d) \cup P_t^H(Nc), \upsilon$ )
    $P_{t+1}(d'')$=Selection( $P_t(d') \cup P_t^H(Nc')$ )
    if ($d'' < d$) {
        Generate($P_{rand}(d - d'')$); Evaluate( $P_{rand}(d - d'')$ ); $FFE = FFE + (d - d'')$
        $P_{t+1}(d) = P_{t+1}(d'') \cup P_{rand}(d - d''))$
    }
    $t = t + 1$;
}
```

Figure 1. Pseudo-code of the clonal selection algorithm

The algorithm has four control parameters: d – the population size, c – the number of clones for each of the solutions in P, υ – the maximum life-time of a solution and $FFE_{\max}$ – the maximum number of fitness function evaluations (FFE) in the experiment.

The main loop consists of five steps. In *Cloning* for each of the solutions a set of c clones is generated. *Hypermutation* mutates clones in $P_t(Nc)$ so a set of new solutions $P_t^H(Nc)$ is created as a result of this step. Then the solutions are evaluated. All the solutions, i.e. both the original ones stored in $P_t(d)$ and the mutated clones from $P_t^H(Nc)$ are aged. Clearly the algorithm eliminates those of the solutions which age is higher than υ. Eventually there follows *Selection* step where a sum of solutions from sets $P_t(d)$ and $P_t^H(Nc)$ undergoes selection procedure. Clearly the best d solutions of the sum build up a new set $P_{t+1}(d)$. If the number of available solutions in $P_t(d') \cup P_t^H(Nc')$ is less than d a set of additional random solutions P_{rand} is generated and $P_{t+1} = P_{t+1} \cup P_{rand}$. After the *Selection* the loop continues from the beginning.

2.1 Two Versions of the Algorithm

The algorithm described above is quite close to the version presented in [4]. It was designed to cope with stationary optimization tasks thus it is expected that it will be outperformed with an ease by recent algorithms devoted to non-stationary optimization. Therefore we extended our tests and build up a new version of the algorithm by introduction of two additional modifications typical for non-stationary optimization heuristics. The first modification is the rule of succession: clearly the original solution has to be replaced only by its mutated clones. This approach has been tested in e.g. [5] where it proved its efficiency. The other modification is concerned with the problem of the convergence of solutions in the population. To uphold the diversity on the level which allows to react immediately to the changes a mechanism which is very similar to the one called *random immigrants* was applied. However there is a difference lying in the rules of selection of solutions to remove. In our version the population loses those of its solutions which are to close to each other. If the distance is less than a defined threshold r_{excl} it is assumed that both solutions belong to the basin of attraction of the same optimum and one of them can be safely excluded. The threshold r_{excl} has to be tuned respectively to the severity of changes. The excluded solutions are replaced by randomly generated new ones. Except for introduction of the two modifications the last difference between the original and the tuned version can be found in the step of *Aging*. In the tuned version the step is omitted which means that there is no limit for the lifetime of the solutions.

Finally it is necessary to note that we assumed that the optimization system "knows" when its environment has changed. However the algorithm does not start from scratch but reevaluates its population of solutions and continues the search process.

2.2 Mutation Operator Based on Potential Functions

In our approach a Gray coded binary representation of solutions is in use. The binary mutation operator works on the idea of flipping randomly selected bits. In our research we focused of studying efficiency of different strategies of calculation of the number of bits to be mutated. Therefore ten versions of the operator are proposed that differ in the way of calculation of that number. The number depends on $f'(\mathbf{x}_i)$ which is the fitness of the solution $\mathbf{x}_i$ normalized in [0,1]:

$$f'(\mathbf{x}_i) = \frac{f(\mathbf{x}_i) - f_{min}}{(f_{max} - f_{min})},$$
$$f_{max} = \max_{\mathbf{x}_j, \forall j \in \{1, \ldots |P|\}} f(\mathbf{x}_j) \text{ and } f_{min} = \min_{\mathbf{x}_j, \forall j \in \{1, \ldots |P|\}} f(\mathbf{x}_j).$$

The numbers of mutated bits are defined with the so called *potential functions*. They can be divided into two classes: proportional and inverse proportional. In the former case the number of mutated bits grows as $f'(\mathbf{x}_i)$ grows. In the latter case the number is higher for solutions with lower values of $f'(\mathbf{x}_i)$. Besides the two classes are divided into two types: with positive or negative acceleration. The formulas of the potential functions for all classes and groups are presented in Table 1. Graphs of the functions are presented in Figure 2. It is important to note that the final value of the potential function used for calculation of the number of mutated bits is always increased by a small constant value ϵ to avoid situation where the potential is set to zero and no mutation is performed at all. In the case of the least value of potential just one randomly selected bit is mutated. When the potential reaches to maximum value in the graph it means that the maximum available number of randomly selected bits is mutated.

The number of mutated bits varied from 1 to $l/2$ where l is the number of bits granted to represent all the coordinates of a solution. In the step of mutation the algorithm selects randomly one of the potential functions and applies it to modify the solution. Each of the

Table 1. Formulas of the potential functions

	formula	symbol	ρ
proportional and positive acc.			
1.	$\alpha_i = \frac{\exp(f'(\mathbf{x}_i)^\rho) - \exp(0)}{\exp(1) - \exp(0)}$	$\uparrow \alpha_+ \exp$	2
2.	$\alpha_i = f'(\mathbf{x}_i)^\rho$	$\uparrow \alpha_+$pow	2
proportional and negative acc.			
3.	$\alpha_i = \frac{\ln(f'(\mathbf{x}_i)^{1/\rho} + 1) - \ln(1)}{\ln(2) - \ln(1)}$	$\uparrow \alpha_- \ln$	2
4.	$\alpha_i = \sqrt[\rho]{f'(\mathbf{x}_i)}$	$\uparrow \alpha_-$sqrt	2
inverse prop. and positive acc.			
5.	$\alpha_i = 1/\rho * \exp(-f'(\mathbf{x}_i))$	$\downarrow \alpha_+$org2	25
6.	$\alpha_i = (-f'(\mathbf{x}_i) + 1)^\rho$	$\downarrow \alpha_+$pow	2
7.	$\alpha_i = \exp(-\rho * f'(\mathbf{x}_i))$	$\downarrow \alpha_+$org1	5
8.	$\alpha_i = \frac{\exp(-f'(\mathbf{x}_i)^{1/\rho}) - \exp(-1)}{\exp(0) - \exp(-1)}$	$\downarrow \alpha_+ \exp$	2
inverse prop. and negative acc.			
9.	$\alpha_i = \frac{\ln(-f'(\mathbf{x}_i)^\rho + 2) - \ln(1)}{\ln(2) - \ln(1)}$	$\downarrow \alpha_- \ln$	2
10.	$\alpha_i = \sqrt[\rho]{-f'(\mathbf{x}_i) + 1}$	$\downarrow \alpha_-$sqrt	2

potential functions has a counter which is incremented every time the mutated solution is better than the original one. Probability of selection of the operator is proportional to the value of the counter. This way the potential functions that give more improvements in the solutions have more chances to be used in the future.

3 The Benchmark and Applied Measure

Moving Peaks Benchmark generator [3] was selected as a test-bed for experiments. Its description, parameters settings and a source code are available at the web page [2]. The parameters of MPB are set exactly the same as specified in scenario 2 of this benchmark. The fitness landscape consists of 10 moving peaks and it is defined for the 5-dimensional search space with boundaries for each of dimensions set to $[0; 100]$. Moving peaks vary their height randomly within the interval $[30; 70]$, width within $[1; 12]$ and position by a distance of 1.

To evaluate the results, an offline error [3] measure was used. The measure represents the average deviation of the best solution from the optimum evaluated since the last change of the fitness landscape. Every experiment was repeated 50 times.

We also need to mention that in our experiments the offline error is not evaluated

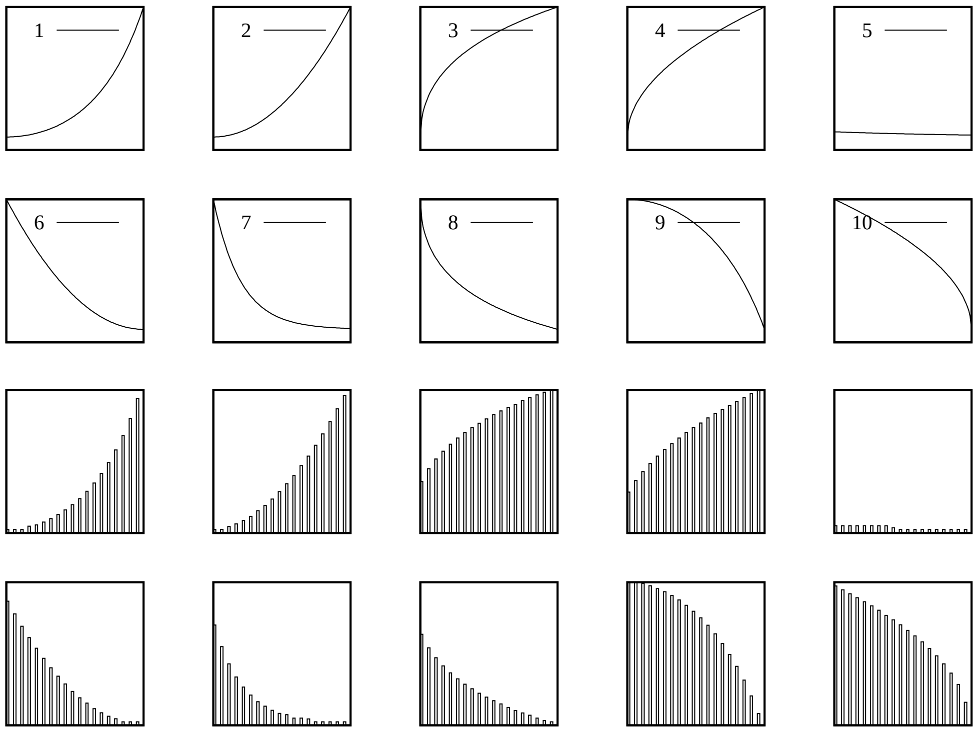

Figure 2. Graphs of the potential functions (the first and the second row) and respective average numbers of mutated bits for different values of the normalized fitness of solution varying from 0 to 1 (the third and the fourth row)

from the beginning of the experiment but the evaluation starts form the 30th change in the fitness landscape. It is important to stress because as it was shown in [7] the influence of the evaluations of the offline error in the initial phase of the experiment significantly increases its final value.

4 Results of Experiments

For the testing environment a set of tests was performed where the influence of the mutation operator on the offline error was observed. All the tests were performed for 16-bit representation of each of the coordinates of a solution and for 10 solutions in the population.

For the experiments with original version of the algorithm the maximum life-time υ was tested for two values: 3 and 4. Number of clones c was set to 2 or 4. The mutation operator worked in two modes. In the 1st mode one randomly selected coordinate of the five always remains untouched. In the 2nd mode all the five coordinates are modified. It gave four versions of the algorithm: two solutions management strategies for the 1st mode: $[\upsilon = 3, c = 2]$ and $[\upsilon = 4, c = 4]$ and the same two strategies for the 2nd mode.

The tuned version of the algorithm has the number of clones $c = 2$. Besides this version was also tested with the two modes of mutation mentioned above. For both algorithms maximum number of fitness function calls $FFE_{\max}$ was set to 300000.

Two types of information were gathered during experiments. The first one is a current value of offline error. The second one is represented by the final values of counters of successful executions (i.e. the executions where the mutated solution is better than the original one) for each of the potential functions used in the mutation operator. Mean of the current value of offline error is presented in Figure 3. Graphs with box-and-whisker diagrams of numbers of successful mutations for each of the ten tested potential functions including all five quartiles are presented in Figure 4.

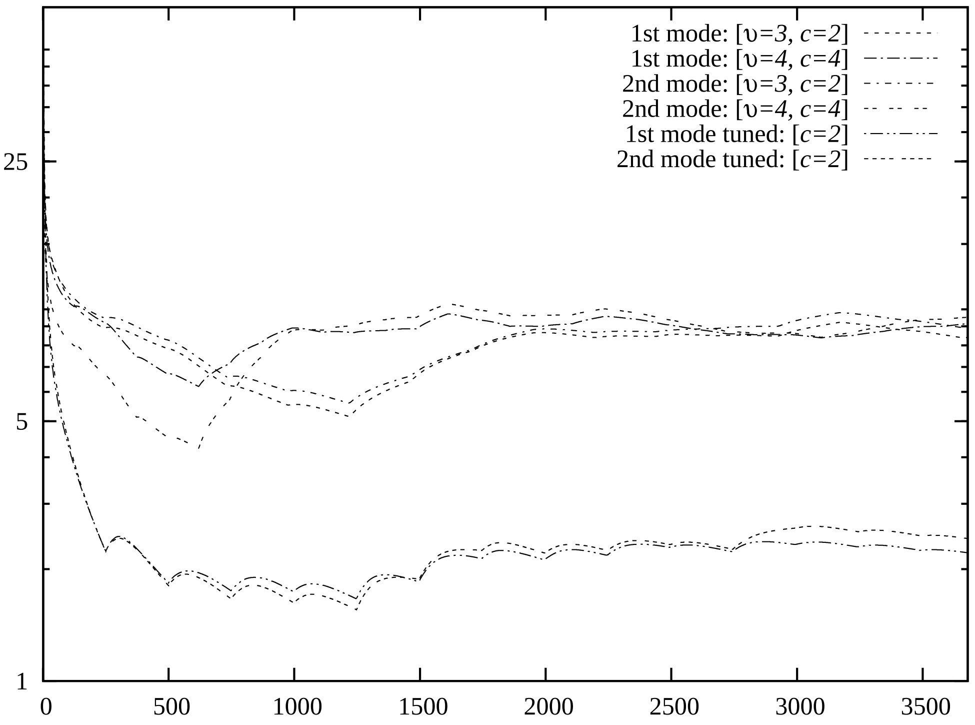

Figure 3. Current value of offline error for different values of algorithm parameters settings

5 Conclusions

There are three main conclusions arising from the presented results. The first one is about the mutation modes: 1st and 2nd. The second one is about the strategies of management of the solutions. The last one is about the potential functions in the mutation.

When we compare the values of offline error for the modes 1st and 2nd in Figure 3 it is not so easy to indicate the leader. In the first stage of the search process the 2nd mode outperforms the 1st one. However after some time when the search process is getting stabilized the 1st mode is the one which give slightly lower value of offline error. So eventually the 1st mode could be the winner however it must be stressed that it depends

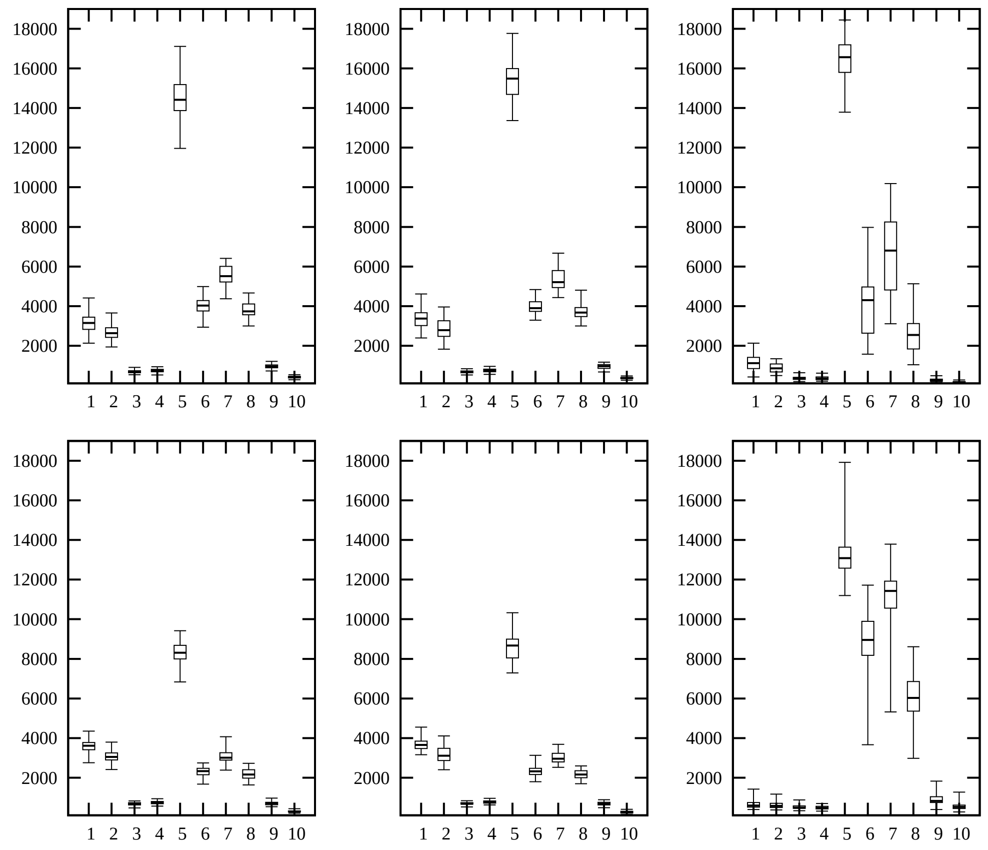

Figure 4. Box-and-whisker diagrams of numbers of successful mutations for each of the ten tested potential functions: 1st row – 1st mode, 2nd row – 2nd mode

of the time we have for searching. If the optimization has to be done for a short time and for a small number of changes the 2nd mode should rather be selected.

When we compare the two strategies of the solutions management in the original version of CSA in Figure 3 the more aggressive search is performed with $[\upsilon = 4, c = 4]$. However this strategy cannot manage with the changes regularly appearing in the environment and quickly loses its advantage. Strategy $[\upsilon = 3, c = 2]$ needs more time to reach as good level of offline error as the competitor can but also in this case loss of quality of the returned results can be observed in the further part of the search process. Eventually in the last part of the search process results returned by both strategies are quite similar to each other. The tuned version of CSA outperforms the original version with an ease which is not very surprising. What is more interesting, even this algorithm after the first successes slightly looses quite good level of offline error. This loss of quality can be interpreted as problems with flexibility of reaction to changes. It is caused most probably by lack of appropriate diversity in the set of solutions in further stages of the process of search. The diversity is present in the beginning of the search but disappears after some time and cannot be quickly obtained with the given mutation operator. But anyway in spite of this disadvantage the results of the tuned CSA are comparable to the results presented in the recent publications (see e.g. [1]).

Figure 4 allows to compare efficiency of potential functions tested in the mutation operator. Definitely the function No 5 is the most successful. It mutates very gently all the solutions in the population. This high number of successes confirms well known opinion about restrained application of bit flipping in the mutation. The numbers of successes gained by the remaining functions are more interesting. In case of the original version of the algorithm one can observe more successes for the functions with the positive acceleration than with the negative one. In case of application of the 1st mode of mutation the functions with the inverse proportional and positive acceleration seem to have a bit more successes than those with just the proportional and positive acceleration. This advantage is much more evident in case of the tuned version of CSA. Here the functions No 5 and 7 have the largest number of successes which means that the best strategy for mutation is to do it moderately except for the worst solutions in the population where intensive mutation could be desirable from time to time.

Bibliography

[1] T. Blackwell and J. Branke. Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation*, 10(4):459 – 472, Aug. 2006.

[2] J. Branke. The moving peaks benchmark. URL: http://www.aifb.uni-karlsruhe.de/~jbr/MovPeaks/movpeaks/.

[3] J. Branke. Memory enhanced evolutionary algorithm for changing optimization problems. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proc. of the Congress on Evolutionary Computation*, volume 3, pages 1875–1882. IEEE Press, Piscataway, NJ, 1999.

[4] Pavone M. Cutello V., Nicosia G. A hybrid immune algorithm with information gain for the graph colouring problem. In *Genetic and Evolutionary Computation – GECCO-2003*, volume 2723 of *LNCS*, pages 171–182. Springer-Verlag, 2003.

[5] K. Trojanowski and S. T. Wierzchoń. Studying properties of multipopulation heuristic approach to non-stationary optimisation tasks. In M. A. Kłopotek, S. T. Wierzchoń, and K. Trojanowski, editors, *IIS 2003:Intelligent Information Processing and Web Mining*, Advances in Soft Computing, pages 23–32. Springer Verlag, 2003.

[6] Krzysztof Trojanowski. Clonal selection principle based approach to non-stationary optimization tasks. In *Evolutionary Computation and Global Optimization 2006*, number 156 in Prace Naukowe, Elektronika, z.156, pages 375–384. Warsaw Univ. of Technology Publishing House, 2006.

[7] Krzysztof Trojanowski. B-cell algorithm as a parallel approach to optimization of moving peaks benchmark tasks. In *Sixth International Conference on Computer Information Systems and Industrial Management Applications (CISIM 2007)*, pages 143–148. IEEE Computer Society Conference Publishing Services, 2007.