HAL
open science

C-Strategy: A Dynamic Adaptive Strategy for the CLONALG Algorithm

Maria Cristina Riff, Elizabeth Montero, Bertrand Neveu

To cite this version:

Maria Cristina Riff, Elizabeth Montero, Bertrand Neveu. C-Strategy: A Dynamic Adaptive Strategy for the CLONALG Algorithm. Marina L. Gavrilova and C. J. Kenneth Tan. Transactions on Computational Science VIII, Springer-Verlag, pp.41-55, 2010, Lecture Notes in Computer Science, 978-3-642-16235-0. 10.1007/978-3-642-16236-7_3 . hal-00654389

HAL Id: hal-00654389

https://enpc.hal.science/hal-00654389

Submitted on 20 Sep 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

C-Strategy: A Dynamic Adaptive Strategy for the CLONALG Algorithm*

María Cristina Riff[1], Elizabeth Montero[1,2], and Bertrand Neveu[3]

[1] Universidad Técnica Federico Santa María,
Valparaíso, Chile
{Maria-Cristina.Riff,Elizabeth.Montero}@inf.utfsm.cl
[2] Université Nice Sophia Antipolis,
France
[3] Projet COPRIN, INRIA Sophia-Antipolis
France
Bertrand.Neveu@sophia.inria.fr

Abstract. The control of parameters during the execution of bio-inspired algorithms is an open research area. In this paper, we propose a new parameter control strategy for the immune algorithm CLONALG. Our approach is based on reinforcement learning ideas. We focus our attention on controlling the number of clones. Our approach provides an efficient and low cost adaptive technique for parameter control. We use instances of the Travelling Salesman Problem. The results obtained are very encouraging.

1 Introduction

When we design a bio-inspired algorithm to address a specific problem we need to define a representation, and a set of components to solve the problem. We also need to choose parameter values which we expect will give the algorithm the best results. This process of finding adequate parameter values is a time-consuming task and considerable effort has already gone into automating this process [10]. Researchers have used various methods to find good values for the parameters, as these can affect the performance of the algorithm in a significant way. The best known mechanism to do this is *tuning parameters* on the basis of experimentation, something similar to a generate and test procedure. Taking into account that an immune algorithm run is an intrinsically dynamic adaptive process, we can expect that a dynamic adaptation of the parameters during the search could help to improve the performance of the algorithm, as it has been shown for other metaheuristics [3], [5], [23], [29], [7], [16], [15]. The idea of adapting and self-adapting parameters during the run is not something new, but we need to manage a trade-off between the improvement of the search and the adaptation cost. In this case, the idea is to monitor the search to be able to trigger actions from an adaptive parameter control strategy, in order to improve the performance of the well-known immune algorithm CLONALG [4].

* Partially Supported by the Fondecyt Project 1080110.

M.L. Gavrilova et al. (Eds.): Trans. on Comput. Sci. VIII, LNCS 6260, pp. 41–55, 2010.
© Springer-Verlag Berlin Heidelberg 2010

We present a brief revision of the related work in the parameter control domain in the following section. The algorithm CLONALG is briefly described in section 3 with remarks on the aspects of parameter control. We propose a method in section 4 which includes a monitoring task. We have tested the algorithm using our strategy with instances of the Travelling Salesman Problem, that is reported in section 5. The conclusions in section 6 resume our experience from this study and the following trends of this research.

2 Related Work

We can classify the parameter selection process into two different methods:*tuning* and *parameter control*, [6]. Tuning, as mentioned above, implies in the worst case a generate and test procedure, in order to define which are the "best" parameter values for a bio-inspired algorithm. Usually, these parameter values are fixed for all the runs of the algorithm. As we mentioned before, Revac [19] has been proposed to tune of evolutionary algorithms in an efficient way. The idea is to use an estimation of distribution algorithms in an iterative process to converge upon a set of good values for the parameters. The ParamILS [12] technique is also available for tuning, it uses a local search strategy to search the parameter values space in an efficient way. The Racing methods [1] use statistical information about the performance of the tuned algorithm to carry out a race between different parameter configurations.

We can expect that the "best" parameter values depend on the step of the algorithm we are using. The main disadvantage of the tuning methods is the time they require to perform the calibration process. For that reason, the idea of designing strategies to allow the algorithm to change its parameter values during its execution appears as a promising way to improve the performance of a bio-inspired algorithm. In this context many ideas have been proposed in the evolutionary research community, including self-adaptability methods. In self-adaptive methods, each individual incorporates information which can influence the parameter values or the evolution of these processes can be used to define some criteria to produce changes, [22], [9], [16].

The parameter control methods can be grouped according to the parameter controlled. For example, we can find strategies which control the population size. The parameter-less algorithm [13] implements a population control performing simultaneous runs of a genetic algorithm with different population sizes, emulating a race among the multiple populations. As the search progresses, parameter-less eliminates small populations which present a worse average performance than larger populations. A recent research of Eiben et al. [7] proposes a strategy to change the population size of the genetic algorithm, taking into account the state of the search. The method is called PRoFIGA. PRoFIGA is able to change the population size according to the convergence and stagnation of the search process.

It is also possible to find strategies which combine methods to control several parameters. In [24], Srinivasa et al. propose the SAMGA method. The SAMGA method incorporates an adaptive parameter control technique to vary the population size, mutation rate and crossover rate of each population of a parallel

genetic algorithm. The changes are performed according to the relative performance of one population against the others. Moreover, it dynamically controls the migration rate based on the stagnation of the global search procedure. Recently, Montiel et al. [17] introduced the HEM algorithm which implements an evolutionary algorithm. The HEM algorithm incorporates an adaptive intelligent system which allows the parameter values of the individuals to be controlled in a self-adaptive way. The intelligent system is able to learn about the evolutionary algorithm performance based on expert knowledge and feedback from the search process.

The parameter setting problem, usually suited for evolutionary algorithms, also has been extended to other research areas. In [14] Mezura and Palomeque proposed a parameter control strategy for the constraint-handling mechanism used by a Differential Evolution Algorithm. In their strategy they self-adapted three parameters and one parameter is deterministically controlled. In [28], the authors proposed a two-fold strategy to deterministically control the exploration/exploitation balance during the search and to adaptively coordinate the crossover and mutation operation. This strategy is implemented in an evolutionary multi-objective algorithm with binary representation.

One of the first approaches to parameter control in immune systems was proposed by [8]. In this work Garret proposed a self-adaptive control strategy to control the amount of clones generated from each cell. The amount of clones is established according to predictions of the performance of the algorithm based on the performance that the algorithm showed in the previous iterations. In [11], Hu et al. propose an adaptive control strategy to control the exploration/exploitation rate in the search. In their approach, they divide the repertoire set in three subsets based on their quality. The cells in the best set are mutated according to a micro-mutation process based on the normalized performance of these cells. This approach implements an elitist crossover operator that operates in the two better set of cells. The strategy proposed is able to control the amount of exploitation/exploration according to the variation of two parameters associated to the micro-mutation and crossover operators here defined. The parameter associated to the micro-mutation operator is controlled in an adaptive way and the parameter associated to the crossover operator is controlled in a deterministic way.

The advantage of changing the parameter values during the execution becomes more relevant when we are tackling NP-complete problems. In this paper we introduce a strategy which is able to implement efficient parameter control by using the idea of taking into account the evolution, while also taking action during the execution in order to accelerate the convergence process.

3 Description of CLONALG

Artificial Immune Systems (AIS) have been defined as adaptive systems inspired by the immune system and applied to problem solving. In this paper we are interested in CLONALG that is an artificial immune algorithm based on Clonal Selection. CLONALG has successfully been applied to solve various complex

Standard CLONALG
Begin
Cells = Initialise population of A antibodies (1)
Calculate Fitness (Cells) (2)
i=1;
Repeat
S_p = Selected **n** best antibodies from Cells (3)
P_c = Generate C clones of each antibody from S_p (4)
P_c = Mutation(P_c) (5)
Cells = Cells + P_c (6)
Cells = Selected **n** best antibodies from Cells (7)
Cells = Replace worst (Cells, Generate $(A - \mathbf{n})$ New antibodies); (8)
$i = i + 1$;
until i=Max-number-of-iterations
End

Fig. 1. Standard version of CLONALG

problems. It follows the basic theory of an immune system response to pathogens. Roughly speaking, the components of the algorithm are cells named antibodies and an antigen that is an invader attacking the immune system. The immune system reproduces those cells that recognise antigens. Cells that match well are cloned (create copies of themselves). The better the match, the more clones. The clones undergo small mutations which may improve their recognition. A selection process retains the best cells as memory cells. The CLONALG pseudocode is shown in figure 1.

The algorithm starts generating a random population of Cells and computing their fitness related to the problem at hand. The iterative process begins by constructing a sub-population S_p of size **n**, composed by the **n** best antibodies belonging the population of Cells. A new population of clones P_c is constructed by generating C clones of each element on S_p. This population of clones follows a mutation process, according to a mutation rate, computed using an exponential distribution with parameter ρ, in order to improve the evaluation value of the clones. The set Cells is updated including the **n** best antibodies from P_c and incorporating new $(A - n)$ randomly generated antibodies to construct the population of size A.

The standard parameters of CLONALG are: ρ, C, n and A. Our study is focused on dynamically controlling parameter C. The value of C is related to the exploitation of the search space, because at each iteration the algorithm tries to improve these antibodies applying a mutation procedure.

4 Parameter Control Strategy

The key idea in our approach is to design a low computational cost strategy to control the population size of clones in CLONALG. Our aim is to propose and to evaluate a strategy that allows parameter adaptation in an immune based

approach according to the problem at hand. We propose an adaptive reinforcement control evaluated in the experimental comparison section.

4.1 Adaptive Control

CLONALG works with a set of antibodies as we mentioned before: A population of C clones. The value of this parameter can be controlled in order to guide the trade-off between the intensification and the diversification process of the algorithm. Our idea for this control comes from reinforcement learning. We clearly distinguish between a positive behaviour and a negative one either to reward (increase) or to penalize (reduce) the number of antibodies that belong to the set of clones.

Before we introduce our strategy we require the following definition:

Definition 1. *Given a fitness function F to be maximized, a population P_c of C clones and a population S_p of* **n** *selected antibodies. We define a success measure for the parameter C in the $i-th$ iteration, $PS_i(C)$ as:*

$$PS_i(C) = F(BP_c) - F(BS_p) \tag{1}$$

where $F(BP_c)$ and $F(BS_p)$ are the respective fitness of the best antibody of the populations P_c and S_p. The idea is to evaluate if there is an improvement of the best pre-solution found by the algorithm. In this case we reward this situation by increasing the number of C clones, that means more exploitation or intensification of the search f the algorithm. When the algorithm decides to reduce the number of clones it produces that the new population of cells will be completed adding more new cells. From a conceptual point of view it can be interpreted as more exploration or diversification of the search of the algorithm.

Figure 2 shows the structure of the algorithm with the adaptive control. It is important to remark that in lines 3 and 7 the fitness are data available in the original CLONALG code. We explicitly show their evaluations here because we use them in lines 9 and 10. The algorithm computes the $PS_i(C)$ value in order to increase (decrease) C (number of clones) value. Obviously, the algorithm manages extreme situations, as C must be greater than 1.

Remark 1. The previous definition and the procedure can easily be adapted when the function to be optimized must be minimized, as we use in the experimental comparison section with the travelling salesman problem

The main motivation of the work reported in this paper is to address the on-line parameter determination problem. We propose a new method which improves the performance of CLONALG and also allows a self-adaptation of the parameter C without adding a significant overhead and without introducing any major changes to its original algorithmic design. The decisions made during the search are based on information available from a monitoring process. Such information allows the algorithm to trigger changes when deemed necessary, based on the potential advantages that such changes could bring. It is also important to keep a proper balance between the level of improvement achieved and the computational cost required to reach it.

CLONALG-with adaptive control
Begin
Cells = Initialise population of A antibodies (1)
$i = 1$;
Repeat
S_p = Selected **n** best antibodies from Cells (2)
$F(BS_p)$= Fitness of the best antibody in S_p (3)
P_c = Generate C clones of each antibody from S_p (4)
P_c = Mutation(P_c) (5)
P_c = Selected **n** best antibodies from P_c (6)
$F(BP_c)$ = Fitness of the best antibody in P_c (7)
New-Cells = P_c + Generate $(A - \mathbf{n})$ New antibodies (8)
$PS_i(C) = F(BP_c) - F(BS_p)$ (9)
If $(PS_i(C) > 0)$ **then** $C = C + 1$
else If $C > 2$ **then** $C = C - 1$ (10)
$i = i + 1$ (11)
Cells = New-Cells (12)
until i=Max-number-of-iterations
End

Fig. 2. Clonal Selection Algorithm with Adaptive Control

5 Experimental Comparison

In this section, we experimentally evaluate the CLONALG algorithm using our strategy.

5.1 Experimental Set-Up

The hardware platform for the experiments was a PC CPU Intel Core i7-920 4GB RAM, Mandriva 2009.

5.2 Travelling Salesman Problem

Travelling Salesman Problem (TSP) is a classical optimization problem, defined as the task of finding the shortest path for visiting N cities and returning to the original point. TSP is a classical combinatorial optimization problem [26], [31], [21]. Several approaches have been proposed to face the TSP using artificial immune systems. The more well-known approach was proposed by de Castro and von Zuben [2] that we briefly present in the following section.

TSP-CLONALG: The original version of CLONALG to solve TSP [2] called TSP-CLONALG, has some particularities that differs of its standard version to be noticed. The number of clones of each antibody is calculated according to equation 2. We can observe that the amount of clones is determined by the

parameter β and the ranking of the antibody k. The total amount of clones at each iteration is calculated according to equation 3.

$$C_k = round\left(\frac{\beta \cdot A}{k}\right) \tag{2}$$

$$C = \sum_{k=1}^{n} C_k \tag{3}$$

In order to improve the performance of the standard version of the CLONALG to solve TSP the authors have added two new parameters: The number of randomly generated cells to be included (d) after a fixed number of iterations ($iter$). In the original version of CLONALG the random cells were incorporated at each iteration and the number of random cells was fixed at $(A-n)$. Thus, this version of the algorithm uses six parameters: ρ, β, A, n, d, $iter$. Using β and A it computes the total amount of parameters C.

In this paper we compare the performance between TSP-CLONALG using 6 parameters and the standard CLONALG version using our strategy. In our study we have detected that the parameters ρ and β are not relevant for the algorithm when it uses our control strategy. Thus, for our approach we just considered 2 parameters (A, n).

In both versions of CLONALG the representation is a permutation of the cities and the affinity measure is the length of the tour associated to each instantiation or antibody. We have considered as stopping condition when the algorithm is not able to get a better solution during 40 iterations.

Before continuing our description of tests we will describe the tuning method used in our experiments.

5.3 Tuning Method: Revac

In our experiments we used the tuning strategy called *Relevance Estimation and Value Calibration* method, Revac. The Revac method was proposed by Eiben & Nannen in [19]. Revac is defined as an estimation of distribution algorithm [20]. It works with a set of parameter configurations, a population. For each parameter, Revac starts the search process with a uniform distribution of values within a given range. As the process advances Revac performs transformation operations with the aim of reducing each parameter distribution to a range of values that performs the best.

Revac method works with a population of 100 parameter configurations and at each iteration generates a new calibration using a crossover and a mutation operator. Revac performs a total amount of 1000 runs of the algorithm to be tuned, that means in average 10^9 number of evaluations.

5.4 Evaluation of the Adaptive Strategy

We report the results using the strategy considering Clones Variations (C).

Scenario 1. In our experiments we have analyzed the results achieved by de Castro and von Zuben in [2]. In their work they study the results obtained by CLONALG method in an instance of 30 cities from [18] displayed in figure 3. For this scenario we have used the information described in [2] and we have interpreted their *swap* operator as the well-known *2-opt* as the move for mutation. We have used two versions of tuned CLONALG. The first configuration uses the parameter values proposed by the authors of the method, we called this configuration *hand-tuned* configuration. The second one has automatically been obtained by Revac. Table 1 shows the original parameter configuration and this obtained by Revac. In the original version for TSP, the algorithm differs from the classical CLONALG. For TSP two new parameters are included as we mentioned in section 5.2. It considers a set of $d = 60$ random cells to be included after every 20 iterations. Our adaptive technique works on the standard CLONALG, thus these two parameters are not required for the algorithm using control.

Table 1. Parameter configurations for the 30 cities instance

Instance	ρ	β	A	**n**	d
Hand-tuned	2.5	2.0	300	150	60
Revac	1.2	2.8	190	178	97

Table 2 shows the performance obtained by the tuned parameter configurations and the adaptive strategy proposed for solving the 30 city instance. The three versions of the algorithm found the optimal value of 48873.

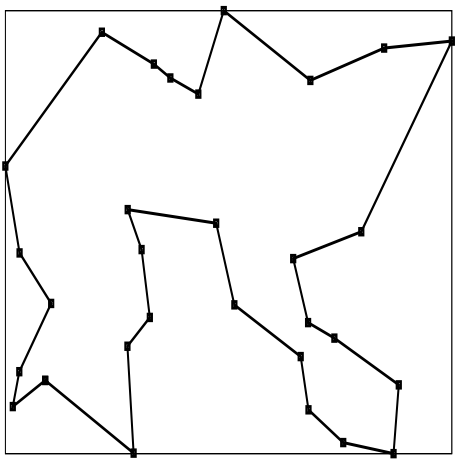


Fig. 3. Coordinates of the 30 cities instance

From the results we observe that the C-strategy outperforms all other versions. However it appears using more evaluations than the tuned versions. The

algorithm using C-strategy has found the optimal tour 25 times, of the 50 total runs, the original hand-tuned version found 6 times and the version tuned using Revac only found the optimal solution 8 times.

We do not know how many times the algorithm has been run and the corresponding number of evaluations done to obtain the hand-tuned parameters configuration. Revac performed around of 10^8 number of evaluations in order to obtain the parameter values. These number of evaluations are not included in the Table 2.

Table 2. Performance evaluation in the 30 cities instance. AS: Average Solution, $\%\Delta_{Av}$: Relative distance to optimum of the average of 10 runs, E_{Av}: Average evaluations, E_{best}: Evaluations of the best run, OF: Times the optimum was found.

Algorithm	AS	$\%\Delta_{Av}$	E_{Av}	E_{best}	OF
Hand-tuned	50795	3.9	48693	51256	6
Tuned by Revac∗	50976	4.3	45649	47203	8
C-strategy	50109	2.5	76413	78990	25

∗ Tuned by Revac for 30 cities instance. Revac required around of 10^8 number of evaluations for tuning this instance. These evaluations are not included in the number of evaluations reported in this table.

Table 3. Revac tuning results

Instance	ρ	β	A	**n**	d
burma14	1.8	0.1	87	87	36
ulysses22	1.4	0.2	84	84	65
eil51	1.0	0.1	120	120	4
berlin52	1.8	0.3	112	70	37
st70	1.1	0.2	109	20	36
eil76	1.7	0.2	142	92	53
pr76	0.8	0.1	93	63	28
gr96	2	0.4	133	94	45
kroA100	1.7	0.7	104	95	45
eil101	2.5	0.7	147	103	52
ch130	1.4	0.8	149	73	52
ch150	2.4	3.2	219	164	59
gr202	3.1	0.5	152	150	50
tsp225	2.4	2.0	215	136	43
a280	0.9	3.8	222	203	70
pcb442	0.7	4.1	206	177	59
gr666	1.1	2.6	194	174	97
pr1002	0.5	1.4	172	110	66

Scenario 2. We have tested our technique using symmetric travelling salesman problem instances from TSPLib[1]. The TSPLib is a set of more than 100 instances

[1] `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`

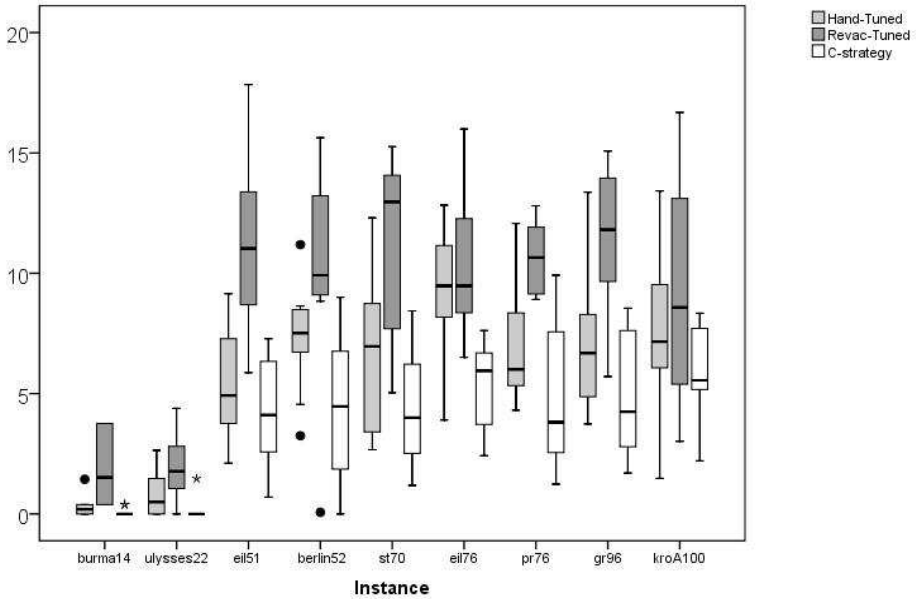
of the TSP collected by Professor Gerhard Reinelt in the mid 1990âĂŹs and that has been extensively used in the metaheuristic research area in the last years [25], [27], [30].

We have also calibrated the algorithm using Revac to solve other instances of the TSP. For this, we have also estimated the value of parameter d, which corresponds to the number of random cells to be inserted each 20 iterations. The results obtained are shown in Table 3. We can observe that, for each instance, Revac has identified different parameter values. This means that the behaviour of the algorithm strongly depends on both the problem instance and the values of its parameters. The amount of selected antibodies, $\mathbf{n}$, ranges from around 20% to 95% of the population size and the amount of random antibodies, d, incorporated to the population each 20 generations ranges from 3% to 50% of the population size. The size of the population of cells A and ρ are also dependent on the instance at hand. The algorithm using our adaptive strategy does not require to be re-tuned each time it must solve another new instance of the problem. Thus, the time invested on tuning is strongly reduced.

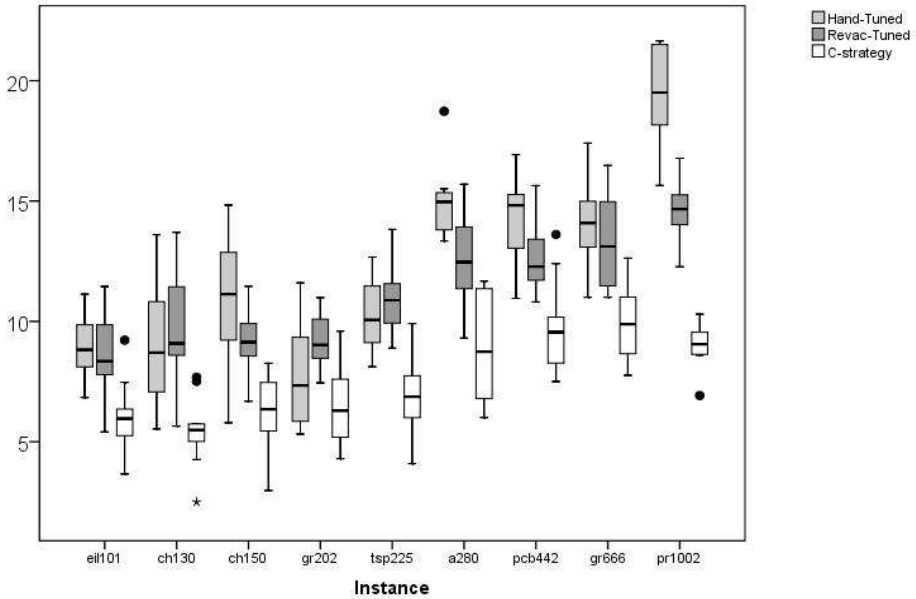
Graphs in figure 4 show the relative distance to optimun for each instance comparing the performance of the C-strategy against to the hand-tuned and Revac-tuned performance.

Table 4. Performance of Hand-tuned configuration. AS: Average Solution, $\%\Delta_{Av}$: Relative distance to optimum of the average of 10 runs, E_{Av}: Average evaluations, BS: Best solution of 10 runs, $\%\Delta_{Best}$: Relative distance to optimum of the best solution found, E_{best}: Evaluations of the best run.

Instance	AS	$\%\Delta_{Av}$	E_{Av}	BS	$\%\Delta_{Best}$	E_{best}
burma14	3333	0.3	16747	3323	0.0	9824
ulysses22	7073	0.8	36631	7013	0.0	30925
eil51	450	5.5	126175	435	2.1	113980
berlin52	8093	7.3	135670	7787	3.2	135263
st70	720	6.6	210339	693	2.7	213953
eil76	589	9.4	207454	559	3.9	239400
pr76	115678	7.0	244200	112825	4.3	291478
gr96	59110	7.1	343631	57276	3.7	346040
kroA100	22874	7.5	374095	21596	1.5	407959
eil101	685	8.9	315624	672	6.8	324708
ch130	6661	9.0	510299	6448	5.5	519051
ch150	7248	11.0	577297	6906	5.8	603625
gr202	43353	8.0	884525	42296	5.3	820052
tsp225	4323	10.4	984360	4234	8.1	1069647
a280	2965	15.0	1338809	2923	13.3	1356322
pcb442	58021	14.3	2328181	56343	11.0	2385774
gr666	338838	15.1	4014526	332335	12.9	4210456
pr1002	308884	19.2	6568289	299594	15.7	6749630



(a) Instances: burma14 - kroA100



(b) Instances: eil101 - pr1002

Fig. 4. Performance: Hand-Tuned, Revac-Tuned and C-strategy

Table 5. Performance of Revac-tuned configurations. AS: Average Solution, $\%\Delta_{Av}$: Relative distance to optimum of the average of 10 runs, E_{Av}: Average evaluations, BS: Best solution of 10 runs, $\%\Delta_{Best}$: Relative distance to optimum of the best solution found, E_{best}: Evaluations of the best run, Δ^+: Improvement of relative distance to optimum compared to the hand-tuned version.

Instance	AS	$\%\Delta_{Av}$	E_{Av}†	BS	$\%\Delta_{Best}$	E_{best}†	Δ^+
burma14	3388	2.0	518	3336	0.4	707	-1.7
ulysses22	7150	2.0	1561	7013	0.0	1823	-1.1
eil51	473	11.1	4052	451	5.9	4256	-5.5
berlin52	8305	10.1	9799	7547	0.1	9565	-2.8
st70	750	11.1	13008	709	5.0	17287	-4.5
eil76	594	10.4	12049	573	6.5	13426	-1.0
pr76	119710	10.7	7854	117792	8.9	6036	-3.7
gr96	61503	11.4	33120	58362	5.7	39705	-4.3
kroA100	23285	9.4	56371	21924	3.0	50938	-1.9
eil101	683	8.6	122179	663	5.4	144320	0.2
ch130	6708	9.8	124983	6455	5.6	131313	-0.8
ch150	7124	9.1	724714	6964	6.7	749362	1.9
gr202	43833	9.1	274549	43150	7.4	306989	-1.2
tsp225	4343	10.9	849222	4264	8.9	819274	-0.5
a280	2905	12.6	2227123	2819	9.3	2303958	2.3
pcb442	57221	12.7	3372096	56269	10.8	3344210	1.6
gr666	333980	13.5	3779875	327663	11.3	3979137	1.7
pr1002	297017	14.7	4857562	290840	12.3	4965511	4.6

† The amount of evaluations specified does not include the around of 10^9 evaluations performed by the tuning process.

Our strategy helps the algorithm to improve the quality of the solutions found by the tuned versions. The greater is the number of cities the more noteworthy are the results obtained by the C-strategy.

6 Conclusions

We propose in this paper a strategy for an adaptive parameter control for CLONALG.

For intensification, our strategy works with the number of clones to be improved. When the improvement procedure has been applied successfully the algorithm increases the number of clones which will follow the improvement process in the next iteration.

In our design we have taken into account the overhead produced by including our strategy. It is a low cost strategy. The main advantage of using adaptive strategies to improve the behavior of the CLONALG comes from non-existing pre-execution time. The tuning techniques can obtain very good results to find tuned parameters for a given problem, but the parameter configuration step

Table 6. Performance of C-strategy. AS: Average Solution, $\%\Delta_{Av}$: Relative distance to optimum of the average of 10 runs, E_{Av}: Average evaluations, BS: Best solution of 10 runs, $\%\Delta_{Best}$: Relative distance to optimum of the best solution found, E_{best}: Evaluations of the best run, Δ^+: Improvement of relative distance to optimum compared to the hand-tuned version.

Instance	AS	$\%\Delta_{Av}$	E_{Av}	BS	$\%\Delta_{Best}$	E_{best}	Δ^+
burma14	3326	0.1	13326	3323	0.0	13069	0.2
ulysses22	7023	0.1	37915	7013	0.0	39049	0.7
eil51	444	4.2	280361	429	0.7	212817	1.3
berlin52	7870	4.4	325300	7542	0.0	276394	2.9
st70	705	4.5	624727	683	1.2	673642	2.1
eil76	567	5.3	715914	551	2.4	649833	4.1
pr76	113273	4.7	742192	109504	1.2	838858	2.2
gr96	57878	4.8	1484345	56145	1.7	1653666	2.2
kroA100	22545	5.9	1643365	21753	2.2	1629237	1.5
eil101	667	6.0	1394766	652	3.7	1300641	2.9
ch130	6444	5.5	2803527	6263	2.5	3069959	3.5
ch150	6931	6.2	3969191	6722	3.0	4137293	4.9
gr202	42771	6.5	7422549	41886	4.3	41886	1.4
tsp225	4186	6.9	9877811	4076	4.1	9599825	3.5
a280	2809	8.9	17376294	2734	6.0	17240650	6.1
pcb442	55693	9.7	45624518	54588	7.5	49324914	4.6
gr666	323210	9.8	117452296	317200	7.8	120874559	5.3
pr1002	282506	9.1	286994425	276963	6.9	280607399	10.2

requires a huge amount of time compared to the time needed for the tuned algorithm to find a solution.

The original TSP-CLONALG algorithm uses 6 parameters. Accorditong to the results obtained using Revac, the value of these parameters are different for different instances of the TSP. Thanks to our dynamic control strategy the number of parameters to solve TSP is reduced to only 2. Moreover, these 2 parameters require only one tuning step whatever is the TSP instance to be solved from these used in our tests.

The algorithm using our adaptive strategy does not require to be re-tuned for solving new instances of the problem. Thus, the time invested for tuning is strongly reduced. The strategy that is proposed helps CLONALG to improve its performance when it is solving Travelling Salesman Problem instances. The C-strategy is able to improve the performance in all instances tested with a relative average improvement of 3.3.

As future work we would like to evaluate our strategy in CLONALG using other hard combinatorial problems. We are also considering to include this strategy in other kinds of artificial immune algorithms based on Immune Network Models. We are working to include this strategy in CD-NAIS, an immune algorithm that solves hard instances of Constraint Satisfaction Problems. A promising research area is the collaboration between various parameter control strategies.

References

1. Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In: Proceedings of the Genetic and Evolutionary Computation Conference, New York-United States, July 2002, pp. 11–18. Morgan Kaufmann, San Francisco (2002)
2. De Castro, L.N., Von Zuben, F.: The clonal selection algorithm with engineering applications. In: Proceedings of Workshop on Artificial Immune Systems and their Apllications, GECCO, pp. 36–37. Morgan Kaufmann, San Francisco (2000)
3. Davis, L.: Adapting operator probabilities in genetic algorithms. In: Proceedings of the third international conference on Genetic algorithms, pp. 61–69. Morgan Kaufmann, San Francisco (1989)
4. de Castro, L.N., Timmis, J.: Artificial Immune Systems: A New Computational Intelligence Approach. Springer, Heidelberg (2002)
5. Deb, K., Agrawal, S.: Understanding interactions among genetic algorithm parameters. In: Foundations of Genetic Algorithms, vol. 5, pp. 265–286. Morgan Kaufmann, San Francisco (1999)
6. Eiben, A.E., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. IEEE Transactions on Evolutionary Computation 3, 124–141 (1999)
7. Eiben, A.E., Marchiori, E., Valkó, V.A.: Evolutionary algorithms with on-the-fly population size adjustment. In: Yao, X., Burke, E.K., Lozano, J.A., Smith, J., Merelo-Guervós, J.J., Bullinaria, J.A., Rowe, J.E., Tiňo, P., Kabán, A., Schwefel, H.-P. (eds.) PPSN 2004. LNCS, vol. 3242, pp. 41–50. Springer, Heidelberg (2004)
8. Garrett, S.M.: Parameter-free, adaptive clonal selection. In: IEEE Congress on Evolutionary Computation, vol. 1, pp. 1052–1058 (2004)
9. Gómez, J.: Self adaptation of operador rates in evolutionary algorithms. In: Deb, K., et al. (eds.) GECCO 2004. LNCS, vol. 3102, pp. 1162–1173. Springer, Heidelberg (2004)
10. Hinterding, R., Michalewicz, Z., Eiben, A.E.: Adaptation in evolutionary computation: A survey. In: IEEE International Conference on Evolutionary Computation, pp. 65–69 (1997)
11. Hu, J., Guo, C., Li, T., Yin, J.: Adaptive clonal selection with elitism-guided crossover for function optimization. In: International Conference on Innovative Computing, Information and Control, pp. 206–209 (2006)
12. Hutter, F., Hoos, H., Stützle, T.: Automatic algorithm configuration based on local search. In: Proceedings of the Twenty-Second Conference on Artifical Intelligence, pp. 1152–1157 (2007)
13. Lobo, F.G., Goldberg, D.E.: The parameter-less genetic algorithm in practice. Information Sciences 167(1-4), 217–232 (2004)
14. Mezura-Montes, E., Palomeque-Ortiz, A.G.: Parameter control in differential evolution for constrained optimization. In: IEEE International Conference on E-Commerce Technology, pp. 1375–1382 (2009)
15. Montero, E., Riff, M.C., Basterrica, D.: Improving MMAS using parameter control. In: IEEE Congress on Evolutionary Computation, Hong-Kong, June 2008, pp. 4007–4011 (2008)
16. Montero, E., Riff, M.C.: Self-calibrating strategies for evolutionary approaches that solve constrained combinatorial problems. In: An, A., Matwin, S., Raś, Z.W., Ślęzak, D. (eds.) Foundations of Intelligent Systems. LNCS (LNAI), vol. 4994, pp. 262–267. Springer, Heidelberg (2008)

17. Montiel, O., Castillo, O., Melin, P., Díaz, A.R., Sepúlveda, R.: Human evolutionary model: A new approach to optimization. Information Sciences 177(10), 2075–2098 (2007)
18. Moscato, P., Fontanari, J.F.: Stochastic versus deterministic update in simulated annealing. Physics Letters A 146(4), 204–208 (1990)
19. Nannen, V., Eiben, A.E.: Relevance estimation and value calibration of evolutionary algorithm parameters. In: Joint International Conference for Artificial Intelligence (IJCAI), pp. 975–980 (2006)
20. Pelikan, M., Goldberg, D.E., Lobo, F.G.: A Survey of Optimization by Building and Using Probabilistic Models. Computational Optimization and Applications 21(1), 5–20 (2002)
21. Richter, D., Goldengorinand, B., Jäger, G., Molitor, P.: Improving the efficiency of helsgauns lin-kernighan heuristic for the symmetric tsp. In: Proceedings of the Fourth Workshop on Combinatorial and Algorithmic Aspects of Networking, pp. 99–111 (2007)
22. Riff, M.C., Bonnaire, X.: Inheriting parents operators: a new dynamic strategy to improve evolutionary algorithms. In: Hacid, M.-S., Raś, Z.W., Zighed, D.A., Kodratoff, Y. (eds.) ISMIS 2002. LNCS (LNAI), vol. 2366, pp. 333–341. Springer, Heidelberg (2002)
23. Smith, J.E., Fogarty, T.C.: Operator and parameter adaptation in genetic algorithms. Soft Computing - A Fusion of Foundations, Methodologies and Applications 1(2), 81–87 (1997)
24. Srinivasa, K.G., Venugopal, K.R., Patnaik, L.M.: A self-adaptive migration model genetic algorithm for data mining applications. Information Sciences 177(20), 4295–4313 (2007)
25. Stützle, T., Hoos, H.: Max-min ant system and local search for the traveling salesman problem. In: IEEE International Conference on Evolutionary Computation, pp. 309–314 (1997)
26. Stützle, T., Grün, A., Linke, S., Rüttger, M.: A comparison of nature inspired heuristics on the traveling salesman problem. In: Proceedings of the Parallel Problem Solving from Nature (PPSN VI), pp. 661–670. Springer, Heidelberg (2000)
27. Sun, W.-D., Xu, X.-S., Dai, H.-W., Tang, Z., Tamura, H.: An immune optimization algorithm for tsp problem. In: SICE 2004 Annual Conference, vol. 1, pp. 710–715 (2004)
28. Tan, K.C., Chiam, S.C., Mamun, A.A., Goh, C.K.: Balancing exploration and exploitation with adaptive variation for evolutionary multi-objective optimization. European Journal of Operational Research 197(2), 701–713 (2009)
29. Tuson, A., Ross, P.: Adapting operator settings in genetic algorithms. Evolutionary Computation 6(2), 161–184 (1998)
30. Yang, J., Wu, C., Pueh Lee, H., Liang, Y.: Solving traveling salesman problems using generalized chromosome genetic algorithm. Progress in Natural Science 18(7), 887–892 (2008)
31. Zhang, W., Looks, M.: A novel local search algorithm for the traveling salesman problem that exploits backbones. In: Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI 2005), pp. 343–350 (2005)