Information Sciences 181 (2011) 552–566

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

On-the-fly calibrating strategies for evolutionary algorithms [☆]

Elizabeth Montero [a,b], María-Cristina Riff [b,*]

[a] Université de Nice Sophia-Antipolis, France
[b] Department of Computer Science, Universidad Técnica Federico Santa María, Av. España No. 1680, Valparaíso, Chile

ARTICLE INFO

Article history:
Received 12 December 2008
Received in revised form 15 September 2010
Accepted 19 September 2010

Keywords:
Evolutionary algorithms
Parameter control

ABSTRACT

The issue of controlling values of various parameters of an evolutionary algorithm is one of the most important and interesting areas of research in evolutionary computation. In this paper we propose two new parameter control strategies for evolutionary algorithms based on the ideas of reinforcement learning. These strategies provide efficient and low-cost adaptive techniques for parameter control and they preserve the original design of the evolutionary algorithm, as they can be included without changing either the structure of the algorithm nor its operators design.

1. Introduction

The performance of an evolutionary algorithm does not depend only on a good design and implementation, but it is also strongly influenced by the parameter values which determine its behavior. Usually, an evolutionary algorithm incorporates several parameters, like the population size, the operator's probabilities or the total number of generations. Determining the most appropriate parameter values for an arbitrary problem is a complex problem, since such parameters interact with each other in a highly nonlinear manner, and no mathematical models of such interaction currently exist. Throughout the years, two main types of methods have been proposed for setting up the parameter values of an evolutionary algorithm: off-line and on-line strategies [14,18,26,15,4,23]. An off-line method (called tuning) searches for the best set of parameter values which will guide all the subsequent runs of the algorithm. Once defined, these values remain fixed. If the final result is not satisfactory, then a new set of parameter values is defined, and the evolutionary algorithm iterates again. Evidently, this process can be very expensive (in terms of computational time), since many different trials may be required before reaching a set of satisfactory parameter values. Additionally, the set of values chosen by the user are not necessarily the best possible for the given problem, but only the best from the (arbitrary number of) trials performed by the user. On the other hand, on-line methods [6,8,32,35,10,24], focus on changing the parameter values during the search of the algorithm. Thus, the strategy must (1) decide when to change the parameter values and (2) determine the new values for the parameters. Needless to say that these two decisions can strongly affect the performance of the algorithm.

The motivation of this work is to address the on-line parameter determination problem. We propose two new methods which improve the behavior of evolutionary algorithms; these methods can be implemented without adding any significant overhead and without introducing any major changes to its original algorithm design. The decisions made during the search are based on the current information available from a monitoring process, allowing 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.

☆ Supported by Fondecyt Project 1080110.
* Corresponding author.
E-mail addresses: ureta.montero.elizabeth@etu.unice.fr (E. Montero), maria-cristina.riff@inf.utfsm.cl, mcriff@inf.utfsm.cl (M.-C. Riff).

0020-0255/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2010.09.016

The paper is organized as follows. We briefly review the related work in the following section. The ideas behind our methods come from three major sources. First, the parameter values are dynamically computed during the execution of the algorithm, using an on-the-fly monitoring technique. Secondly, the computation of the new values for the parameters is done accordingly to the past performance of each operator, whereas nearly all of the existing methods compute news values independently of the results obtained during the latest iterations. Finally, our methods do not require any change of the algorithm itself, but only the addition of a small control layer to encapsulate the algorithm. This makes our techniques completely independent of the algorithm and therefore very portable. Our approaches are introduced in the Section 3, where we explain how the search is monitored to be able to trigger actions to perform parameter control strategies, in order to improve the performance of the evolutionary algorithms. We have compared our approaches to a set of well-known tuning techniques. The results exhibited in Section 4 showed that using our techniques the algorithm was able to self-manage its initial lack of knowledge about the values of the parameters. This paper concludes with Section 5 where we present the conclusions and some ideas for future work.

2. Related work

The parameter selection process can be classified into two different methods: *tuning* and *parameter control* [9]. Tuning implies, in the worst case, a generate and test procedure, in order to define which are the "best" parameter values for an evolutionary algorithm. These parameter values, usually called configuration, are fixed for each run of the algorithm.

There are a couple of tuning methods that have been proposed in the last few years.

Czarn et al. [5] proposed a statistical framework to study the parameter values and parameter interaction based on the use of anova tests. They studied the results of tuning the mutation and crossover probability of a genetic algorithm in four well-known optimization functions.

Gibbs et al. [11] proposed a method based on genetic drift to tune the parameters of the genetic algorithm. In their method, they performed a set of experiments to generate information to analyse parameter interaction. This information is collected, based on the rate of genetic drift in terms of the change in population fitness variance. From this information, they perform mutual information analysis to determine any relationship between the parameters of the algorithm.

In [4], Birattari et al. proposed a parameter tuning technique based on the competition between a set of configurations, known as the Racing method. In each step, some configurations, which are considered statistically worse than others, are eliminated. The method searches until there is just one configuration, or there is no statistical difference in performance between the configurations on hand.

Recently, the Revac [26] technique has been proposed to perform tuning in an efficient way. Revac is defined as an estimation of distribution algorithm where each parameter is initially considered uniformly distributed. Each parameter distribution changes according to how the search process progresses. Moreover, it uses the entropy notion to estimate the importance of the parameter value in the algorithm.

There are more tuning strategies as Calibra proposed by Adenso-Diaz and Laguna [1]. Calibra is based on Taguchi's partial fractional experimental design coupled with a local search procedure. The method sets up a series of experiments based on Taguchi's method, and uses this information to perform a local search procedure. The goal of this search is to find a local optimum solution by reducing the range of each parameter value in each step. The process is done until no significant change is detected or the ranges of the parameters converge to a single point.

In [15], the authors propose the ParamILS technique based on an iterated local search procedure. In this case, the technique requires the definition of a finite search range of values for each tuned parameter. The ParamILS method searches for a good configuration of parameter values according to the performance measures based on the dominance of the configurations.

In [27], Pavón et al. studied a tuning technique based on a Bayesian case-based reasoning system. The strategy is composed of a phase of construction for the case base. This phase requires the user to possess good knowledge about the domain. It requires the user define the number of performance measures and the selection of a finite set of parameter values for each parameter tuned. A second phase is included to tune the parameters of new instance for the same problem. The incremental learning performs a search among the structures stored in the set of model solutions to predict the new parameter values.

We can expect that the "best" parameter values depend on the step of the algorithm we are running. For that reason, the idea of parameter control is to design strategies to allow the algorithm to change its parameter values during its execution.

The parameters related to operators are a very well studied issue in the parameter control area. In [22], the Compass method is proposed. It implements an adaptive parameter technique to calibrate the operator's rates in a genetic algorithm. The Compass method varies the parameters rates according to the reward each operator achieves. The reward amount is based on the population diversity variation, mean fitness variation and the execution time. The operator selection is performed proportionally to the reward of each operator.

Aine et al. [2] implemented a meta-level framework for an adaptive parameter control based on the notion of *performance profiling* of algorithms. Two performance metrics are used to evaluate an intermediate population: quality of the objective space and diversity of the encoding space. Crossover and mutation rates are controlled in the genetic algorithm for a traveling salesman problem. Crossover, mutation and inversion rates are controlled in the GASP algorithm for the standard cell placement problem.

Self-adaptive control techniques allow each individual to use its own parameter values. These strategies, include information in the individuals that influence the parameter values and, define a set of criteria to produce changes. In [30], the authors propose a method that allows each individual to select the crossover operator according to the information inherited from its parents. Gómez introduced the Hybrid Adaptive Evolutionary Algorithm (HAEA) in [12], a self-adaptive dynamic parameter control for genetic operators. The probability of an operator could be increased in the case of positive behavior or decreased in case of a negative one.

Recently, Mallipeddi et al. [20] proposed a strategy to improve the use of a set of mutation operators in an ensemble of evolutionary programming strategies. The approach uses a population of vectors for each mutation operator included. In each generation, each population generates offspring using its respective mutation operator. During the selection stage, each new population is formed considering the parents of the respective population and the offspring generated by all of the populations of the algorithm. In their implementation, they consider an ensemble of mutation operators, using the adaptive evolutionary programming (AEP) [21].The AEP algorithm is an adaptive version of a classical evolutionary programming algorithm. AEP performs variations of the values of the step sizes (η_i) of vectors according to the median of the best values of the parameter found in previous iterations.

In [29], Polandian adapts the number of offspring generated for a crossover operator in genetic algorithms. In his study, Polandian calculates the amount of children generated according to the degree of difference between parents.

One of the first strategies to control the population size in genetic algorithms is the GAVaPS method [3]. This method incorporates the life time parameter to each individual in the population. This parameter is calculated based on the fitness of the individual according to the fitness of the population. In each generation, the life time of each individual is decreased by one. At the end of each iteration, the individuals with a life time of zero are eliminated from the population.

The parameter less-genetic algorithm proposed by Lobo and Goldberg [19] uses the scheme's theorem to set the crossover probability and selection rate. Moreover, it implements a population control method. This method is based on the idea that a genetic algorithm which works with a large population has less probability of stagnation than a genetic algorithm with a small population. In considering this idea, the parameter-less algorithm performs simultaneous runs of a genetic algorithm with different population sizes, emulating a race among the multiple populations. As the search progresses, parameter-less algorithm eliminates small populations which present worse average performances than larger populations.

A recent research of Eiben et al. [10] proposes a strategy to change the population size of the genetic algorithm, taking into account the state of the search. The PRoFIGA method is able to vary the population size according to the convergence and stagnation of the search process.

In [17], Laredo et al. evaluate a simple variable population size strategy. This strategy is a deterministic parameter control strategy that defines a speed and a severity parameter. In this case, the initial population size is determined according to the minimum amount of building blocks that a selectorecombinative genetic algorithm needs to converge to good solutions.

Another interesting method has been proposed in [28]. In this approach, the adaptation of the parameter values of a genetic algorithm is externally managed by an agent. It receives the search information from the genetic algorithm, and performs a reinforcement learning task, after which it gives news parameter values to the algorithm. In this approach, the genetic algorithm is not itself adaptive.

In [33], Srinivasa et al. propose the SAMGA method. The SAMGA method incorporates an adaptive parameter control technique to change three parameters of a parallel genetic algorithm. The method controls the population size, mutation rate and crossover rate of each population 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. [25] 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 on the search process.

In [13], Hashemi and Meybodi studied a set of control strategies to adapt three parameters of the particle swarm optimization heuristic. In their study, they propose two control level strategies, an individual level strategy (self-adaptive) and a swarm level strategy (adaptive). Moreover, they propose two kinds of variation strategies, the adventurous and the conservative approach. In the adventurous approach, the values of parameters are selected randomly from a finite set. In the conservative approach, the parameter value changes according to a fixed amount.

3. Parameter control strategies

The key idea in our approach is to design low computational cost strategies for parameter control of genetic operators. When we choose to carry out tuning, to find the best parameter value combinations, we need to solve a combinatorial problem. Furthermore, the performance of the algorithm strongly depends on these values. Because tests to find the best combination can not be done in the whole search space. Some methods like Racing algorithms [4,37], ParamILS [15] and Revac [26] have been proposed. These methods are designed in order to define a reduced but significant set of tests.

The main problem with using tuning, is that the combination of the parameter values found, is on *average*, the best evaluated using the set of instances. In our experience, the result of tuning can lead to wrong conclusions. For example, given a combination of parameter values for the operator's rates, we can conclude that one operator is, on average, not useful.

However, when we tackle another instance of the problem, this operator could be the best to apply in guiding the search of the evolutionary algorithm. We clearly verify this situation in the results section.

The idea of our strategies, is to identify which operators are able to generate better offsprings in the current step of the search. For that purpose, we need the following definitions:

Definition 1. Given a set of parents, a fitness function F to be maximized and a genetic operator O_k selected to be applied using the probability from P_h to generate a child C_j. We define a success measure for the operator O_k in its ath application, $S_a(O_k)$ as:

$$S_a(O_k) = F(C_j) - F(P) \tag{1}$$

where $F(C_j)$ and $F(P)$ are the respective fitness of the child and the average fitness of its parents.

When the $S_a(O_k)$ measure is a positive value, it means that the operator has generated a better child than its parents. In this case, we speak about a reward for the operator. If $S_a(O_k)$ value is negative, it means that the operator has generated a child with a worse fitness value than the average fitness of its parents. In this situation, all of our proposed strategies will penalize this operator. Our strategies only differ in the way that they use this information to compute the penalty or reward for the operator. Our aim is to propose and to evaluate two kinds of strategies that allow parameter adaptation in an evolutionary approach according to the problem at hand. We propose two types of reinforcement control: a self-adaptive and an adaptive control. Both of them will be evaluated in the experiments results section.

3.1. Techniques for self-adaptive control

The idea is to make a self-adaptive parameter control strategy where the representation of an individual includes the parameter values of the genetic operators. Thus, each chromosome has its own operators probability values. Roughly speaking, an operator receives a reward when its application generates an individual better than its parents. Analogously, it receives a penalty when the offspring has a worse fitness value than its parents. In this paper we propose two kinds of self-adaptive controls. Both of them use the idea of rewards/penalties. We called light-self-adaptive control (LSA_c) the strategy where rewards and penalties are randomly computed. In the second version, SA_c, both the rewards and the penalties strongly depend on the improvement/degradation of the evaluation function value.

We need to distinguish between a positive and a negative behavior. The following two definitions explicitly state that:

Definition 2. Given a set of operators O_k, $k = o_1,\ldots,o_M$ with a success measure $S_a(O_k) \geqslant 0$, $M \leqslant p$, we define $Max - i_l$ as the maximum improvement made by the operators during the last l generations as:

$$Max - i_l = \underset{a=1,\ldots,A_k,k=o_1,\ldots,o_M}{\text{Arg max}} (S_a(O_k)) \tag{2}$$

where A_k is the number of applications of the operator O_k in the last l generations and p is the number of operators.

This function analyzes all operators that have obtained better individuals than their parents, identifying the biggest improvement of the child's fitness function in the last l iterations.

Analogously, for the degradation of the fitness function as follows:

Definition 3. Given a set of operators O_k, $k = o_1,\ldots,o_M$ with a success measure $S_a(O_k) < 0$, $M \leqslant p$, we define $Max - d_l$ as the maximum degradation done by the operators during the last l generations as:

$$Max - d_l = \underset{a=1,\ldots,A_k,k=o_1,\ldots,o_M}{\text{Arg max}} (|S_a(O_k)|) \tag{3}$$

where A_k is the number of applications of the operator O_k in the last l generations, and p is the number of operators.

Definition 4. Given the ath application of an operator O_k selected using the probability of parent P_h to generate a child C_j, we define its O_k probability as:

$$P_{C_j}(O_k) = \begin{cases} (1+\delta) * P_h(O_k) & \text{for reward } LSA_c \\ (1-\delta) * P_h(O_k) & \text{for penalty } LSA_c \\ \left(1 + \frac{S_a(O_k)}{Max_l}\right) * P_h(O_k) & \text{for } SA_c \end{cases} \tag{4}$$

where, δ is a random number between (0,1), and

$$Max_l = \begin{cases} Max - i_l & \text{if } S_a(O_k) \geqslant 0 \\ Max - d_l & \text{otherwise} \end{cases} \tag{5}$$

Thus, each child generated in the ath application of operator O_k has its probability value computed by Eq. (4) included in its representation. Both LSA_c and SA_c have the same conclusion about the operator O_k, in terms of being rewarded or penalized. The difference is that LSA_c does not regard the amount of the improvement/degradation obtained by the operator application. Instead, SA_c computes a relative importance among operators based on the fitness variations. In other words, SA_c computes the new probability of the operators, giving a reinforcement to the genetic operator. This reinforcement is proportional to the quality of the child that it has generated. Obviously SA_c is more expensive in computational time than LSA_c. The key idea of SA_c is to make a fine discrimination among operators that show a similar behavior by updating the operator's probabilities according to the success measure. LSA_c just notices if the operator has produced an improvement or a deterioration of the fitness evaluation. These control strategies are for an individual based approach. In the following section, we introduce a mechanism, supported by the same definitions, to specify a population based control technique.

3.2. Adaptive control: A_c

Based on the same ideas as the self-adaptive control strategies about operator penalties and rewards, we introduce an adaptive control strategy. The key idea is to do a population based control. Thus, the operator probability values are the same for all of the individuals in the current population. Each operator probability value is updated at the beginning of each new generation considering the information of the operator performance in the last l previous generations.

More formally, and in order to define a discrimination criteria between a positive behavior and a negative one, we introduce the next two definitions:

Definition 5. Given a set of operators O_k, $k = o_1, \ldots, o_M$ and the average success measure $\overline{S_a(O_k)} \geqslant 0$, $M \leqslant p$, we define $Max - ia_l$ as the average maximum improvement done by the operators during the last l generations as:

$$Max - ia_l = \underset{a=1,\ldots,A_k,k=o_1,\ldots,o_M}{\text{Arg max}} (\overline{S_a(O_k)}) \tag{6}$$

where A_k is the number of applications of the operator O_k in the last l generations, and p is the number of operators.

In this case, all individuals in the same generation have the same parameter values. This function uses the success of each operator measured by the average of the improvement made at each of its application in the last l generations. Only the operators, that on average, have generated better offspring than their parents, are considered.

Analogously, we define the degradation as follows:

Definition 6. Given a set of operators O_k, $k = o_1, \ldots, o_M$ and the average success measure $\overline{S_a(O_k)} < 0$, $M \leqslant p$, we define $Max - da_l$ as the average maximum degradation done by the operators during the last l generations as:

$$Max - da_l = \underset{a=1,\ldots,A_k,k=o_1,\ldots,o_M}{\text{Arg max}} (\overline{|S_a(O_k)|}) \tag{7}$$

where A_k is the number of applications of the operator O_k in the last l generations and p is the number of operators.

Finally, the algorithm computes the reinforcement for the operator probabilities taking into account its average improvement/degradation in a proportional factor as follows:

Definition 7. Given the current probability value of the operator O_k, $Pr_t(O_k)$, and the average of the success measure $\overline{S_a(O_k)}$ in the current generation, we define the O_k probability value in the next generation as follows:

$$Pr_{t+1}(O_k) = Pr_t(O_k) + \rho * \frac{\overline{S_a(O_k)}}{Max_{al}} \tag{8}$$

where

$$\underset{a_l}{Max} = \begin{cases} Max - ia_l & \text{if } \overline{S_a(O_k)} \geqslant 0 \\ Max - da_l & \text{otherwise} \end{cases} \tag{9}$$

where ρ is included for providing smooth parameter adjustments, as it is required for any effective dynamic parameter control strategy. Its value is fixed at 0.5.

Remark 1. Thierens [34] analyzes the adaptive allocation rules for adaptive parameter controls. He presents a discussion about the techniques called probability matching. Those techniques allow the algorithm to modify the parameter values using a reward computed as a proportion of the quality of new solutions. In his description of the probability matching approaches, all operators are competing to obtain a reward, and all of them are considered in the same way. Conversely, our approach distinguishes between the set of good operators and the bad ones. It considers two "competitions", where good operators compete to obtain a higher reward and bad operators compete to have a lower penalty. Thierens also proposed the adaptive pursuit algorithm, where the algorithm works with both rewards and penalties. Our approach can be understood as a hybrid strategy between probability matching and adaptive pursuit.

3.3. Observations

The strategies (A_c, LSA_c and SA_c) have some common features:

1. They give a positive or negative reinforcement to the operator probability values, according to the quality of the offspring generated by their applications.
2. The probability values need to be updated in a smooth way in order to allow the algorithm to continue exploring and exploiting in the promising search space.
3. They establish a minimum value for each operator probability in order to keep a minimum level of performance information about each operator in each generation.

Obviously, the overhead for the algorithm is more important for the self-adaptive strategies than for the adaptive strategy. For the self-adaptive strategies the analysis and modifications of the algorithm are done by each individual, where as for A_c, the observation and updates are done by the population.

4. Experimental results

In this section, we experimentally evaluate and compare our parameter control strategies.

4.1. Experimental setup

To test the control strategies proposed, we set the initial probability for each operator in $\frac{1}{p}$, where p is the number of genetic operators. Thus, the algorithm does not need to be previously tuned in respect to its operator probabilities. In each case, we performed 25 runs of each algorithm with different initial random seeds. The hardware platform for the experiments was a PC Intel Core i7 920, with 4 GB of RAM, running Mandriva 2009.1 operating system. The algorithms have been implemented in C++. We use the g++ optimizer. The code for these tests is available on a website.[1]

4.2. Standard genetic algorithm and functions

For the tests, we use the same algorithm and conditions established in two well known research publications, both related to parameter control strategies: the statistical approach of Czarn et al. [5] and the estimation of distribution algorithm, Revac, by Nannen and Eiben [26]:

- The algorithm corresponds to a standard genetic algorithm. It uses Gray based binary representation, elitism, single point crossover and bit-flip mutation as described in [5].
- The population has a fixed size of 50 individuals.

4.3. Functions tested

We consider two sets of functions described in Table 1:

- Test Set I contains the functions Sphere, Rosembrock and Step from the De Jong test set [16] ($f_1,\ldots,f_3$). It also considers the Schaffer's f_6 (f_4 in our paper) [7]. The stop criterion for f_1, f_2 and f_3 is when the convergence limit, i.e. distance between the best current solution and the optimal one, is lower than 10^{-10}, and for f_4 it is fixed at 10^{-6}.
- Test Set II contains the Rastrigin, Schwefel, Griewank and Ackley functions [25] ($f_5,\ldots,f_8$). All these functions are multimodal functions. In this case, D corresponds to the number of dimensions. In our tests we have set D to 5 dimensions and the convergence limit to 10^{-1}.

In the following sections, we present the evaluation of our strategies. We compare both the number of iterations needed to converge and the evaluations used by the algorithms. For the comparisons, we consider both tuned versions of the algorithm, and the algorithms using the three parameter control techniques proposed.

4.4. Results – Test Set I

For the first set of tests using functions $f_1 - f_4$ we consider the tuned version of the algorithm obtained with two different methods: the first version consists of determining the parameter values using the Revac algorithm from Nannen and Eiben [26] (TC_1) and the second version consists of using the Czarn et al. [5] statistically estimated parameter values (TC_2), which

[1] http://www.inf.utfsm.cl/emontero/adaptive-ea.

Table 1
Two sets of functions for testing.

	Test Set I	
f_1	$\sum_{i=1}^{3} x_i^2$	$-5.12 \leqslant x_i \leqslant 5.12$
f_2	$100 \cdot (x_2 - x_1^2)^2 + (1 - x_1)^2$	$-2.048 \leqslant x_i \leqslant 2.048$
f_3	$30 + \sum_{i=1}^{5} \lfloor x_i \rfloor$	$-5.12 \leqslant x_i \leqslant 5.12$
f_4	$0.5 + \frac{(sin(\sqrt{x_1^2+x_2^2}))^2-0.5}{(1+0.0001\cdot(x_1^2+x_2^2)^2)}$	$-100 \leqslant x_i \leqslant 100$
	Test Set II	
f_5	$A \cdot D + \sum_{i=1}^{D} x_i^2 - A \cdot cos(w \cdot x_i)$	$-10 \leqslant x_i \leqslant 10$
f_6	$418.9829 \cdot D + \sum_{i=1}^{D} - 1 \cdot x_i \cdot sin(\sqrt{\lvert x_i \rvert})$	$-500 \leqslant x_i \leqslant 500$
f_7	$\sum_{i=1}^{D} \frac{x_i^2}{4000} - \Pi_{i=1}^{D} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$-100 \leqslant x_i \leqslant 100$
f_8	$-20 \cdot exp(-0.2 \cdot \sqrt{\frac{1}{D} \cdot \sum_{i=1}^{D} x_i^2}) - exp\left(\frac{1}{D}\sum_{i=1}^{D} cos(2 \cdot \pi \cdot x_i)\right) + 20 + e$	$-32 \leqslant x_i \leqslant 32$

Table 2
Crossover and mutation probability values tuned.

F	Revac (TC_1)		Czarn et al. (TC_2)	
	p_c	p_m	p_c	p_m
f_1	0.90	0.01	1.00	0.07
f_2	0.82	0.02	0.00	0.21
f_3	0.98	0.03	1.00	0.05
f_4	0.60	0.06	0.00	0.15

Table 3
Average comparison – standard genetic algorithm with tuning.

F	Revac (TC_1)			Czarn et al. (TC_2)		
	It.	Ev.	Time (s)	It.	Ev.	Time (s)
f_1	11586	577050	10.28	1087	50818	1.04
f_2	8076	367245	5.64	49507	2425834	35.08
f_3	183	9159	0.28	581	29001	0.76
f_4	11728	561925	8.28	24351	1192236	16.52

Table 4
Average comparison – standard genetic algorithm with self-adaptive and adaptive controls.

F	A_c			SA_c			LSA_c		
	It.	Ev.	Time (s)	It.	Ev.	Time (s)	It.	Ev.	Time (s)
f_1	1494	71746	1.2	1443	75085	1.44	13684	732213	13.28
f_2	8366	369697	5.88	12566	578979	9.2	14457	726594	10.16
f_3	327	16296	0.44	399	21717	0.56	368	21172	0.52
f_4	5316	234691	3.68	9649	456502	6.76	15242	786041	10.48

have proposed different parameter values depending on the function to be solved. These parameter values are shown in Table 2.

We remark that the parameter values are really different. For instance, for the functions f_2 and f_4, TC_2 considers that crossover must not be applied, however for TC_1 crossover has a higher probability than mutation.

We report the results obtained using both tuning methods in Table 3. For each function, the table shows the average number of iterations, the average number of evaluations and the time spent for each function using the two different tuning methods (TC_1 and TC_2). We note that:

1. The algorithm behavior strongly depends on the parameter values fixed by the two tuning methods and the function to be solved.
2. The results obtained by TC_1 are quite different to the results from TC_2.

The previous two points further increase the motivation to use an adaptive parameter control strategy, allowing a self-calibration of the algorithm, avoiding a specific tuning for a given problem. We report our results in Table 4.

Table 5
Crossover and mutation probabilities.

F	Revac (TC_1)		ParamILS (TC_3)	
	p_c	p_m	p_c	p_m
f_5	0.41	0.01	0.05	0.05
f_6	0.19	0.01	0.15	0.05
f_7	0.58	0.01	0.15	0.05
f_8	0.02	0.02	0.15	0.05

Table 6
Average comparison – standard genetic algorithm with tuning.

F	Revac (TC_1)			ParamILS (TC_3)		
	It.	Ev.	Time (s)	It.	Ev.	Time (s)
f_5	1438	62546	2.44	8204	403103	14.48
f_6	838	34732	1.48	2563	125960	5.32
f_7	1666	75563	2.88	19332	949716	34.24
f_8	363	17355	0.6	845	41493	1.56

Table 7
Average comparison – standard genetic algorithm with self-adaptive and adaptive controls.

F	A_c			SA_c			LSA_c		
	It.	Ev.	Time (s)	It.	Ev.	Time (s)	It.	Ev.	Time (s)
f_5	7734	381368	14.52	11567	661851	24.36	43710	2474303	94.88
f_6	2690	132824	4.6	5514	328898	11.28	33372	1915083	69.64
f_7	19455	958953	35.2	20832	1097991	39.6	27014	1524281	53
f_8	936	46349	1.72	1877	108314	3.96	8120	462933	17.68

According to [36], there are some special characteristics which make f_4 a more difficult problem to be solved and therefore, a more interesting one for testing genetic algorithms. It is a nonseparable and nonlinear function, and moreover multiple passes of line search cannot yield competitive solutions. This is not the case for function f_1 which is separable and can always be solved by line search. For f_4, our approaches outperform all the other strategies considered in this work.

4.5. Results – Test Set II

For Test Set II we use the ParamILS method described in [15] and the Revac method from [26] to tune the parameters values (crossover and mutation probabilities). For the tests we have used the ParamILS code available in the ParamILS website.[2]

The values obtained are shown in Table 5.

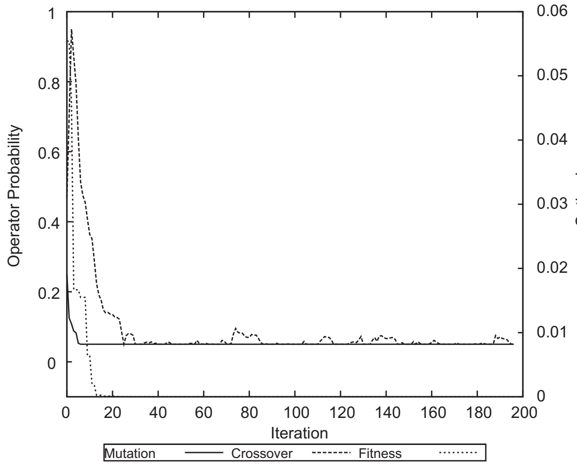
Table 6 shows an important difference of performance between the two tuned versions of the algorithm. The version tuned by Revac shows a much better performance than the one tuned by ParamILS (in terms of the number of iterations, number of evaluations and execution time). However the results in Table 6 for Revac and ParamILS do not include the huge amount of time required to find the parameter values. We report our results in Table 7.

4.6. Convergence

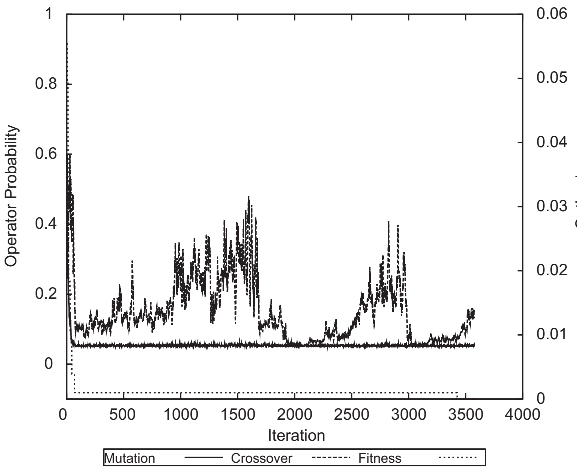
We present the convergence graphs for the three proposed techniques in functions f_4 and f_7. The graphs show the operator's probability by generations. The algorithm does different calibrations according to the functions.

Our control strategies start their search process with equal initial values of operator probabilities (in this case 0.5). The parameter control strategies invest time to vary the initial values during the search process. For both functions, the probability of mutation found by the tuning algorithms was lower than 0.1. Our techniques just require a few number of iterations to adapt the initial mutation probability value by using higher rewards in the first iterations. For instance, in Fig. 1(a) the mutation probability decreases its value in the first 20 iterations. The A_c strategy shows a more stable behavior as we can observe in Figs. 1(a) and 2(a). The self-adaptive strategies show a dynamic variation of probabilities, as it can be seen in Figs. 1(b) and 2(b). The dynamic behavior is more important for LSA_c strategy than for SA_c strategy. This situation is due

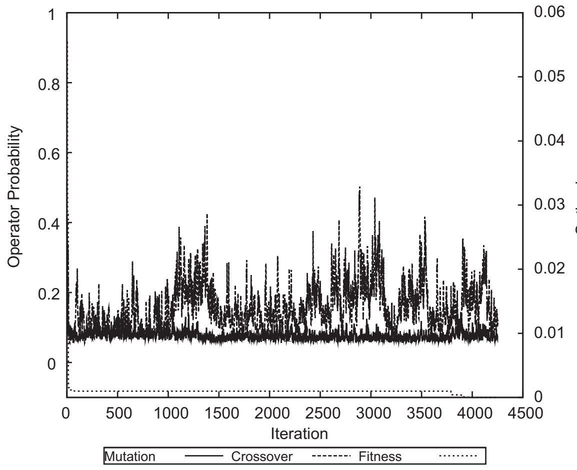
[2] http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/.



(a) A_c



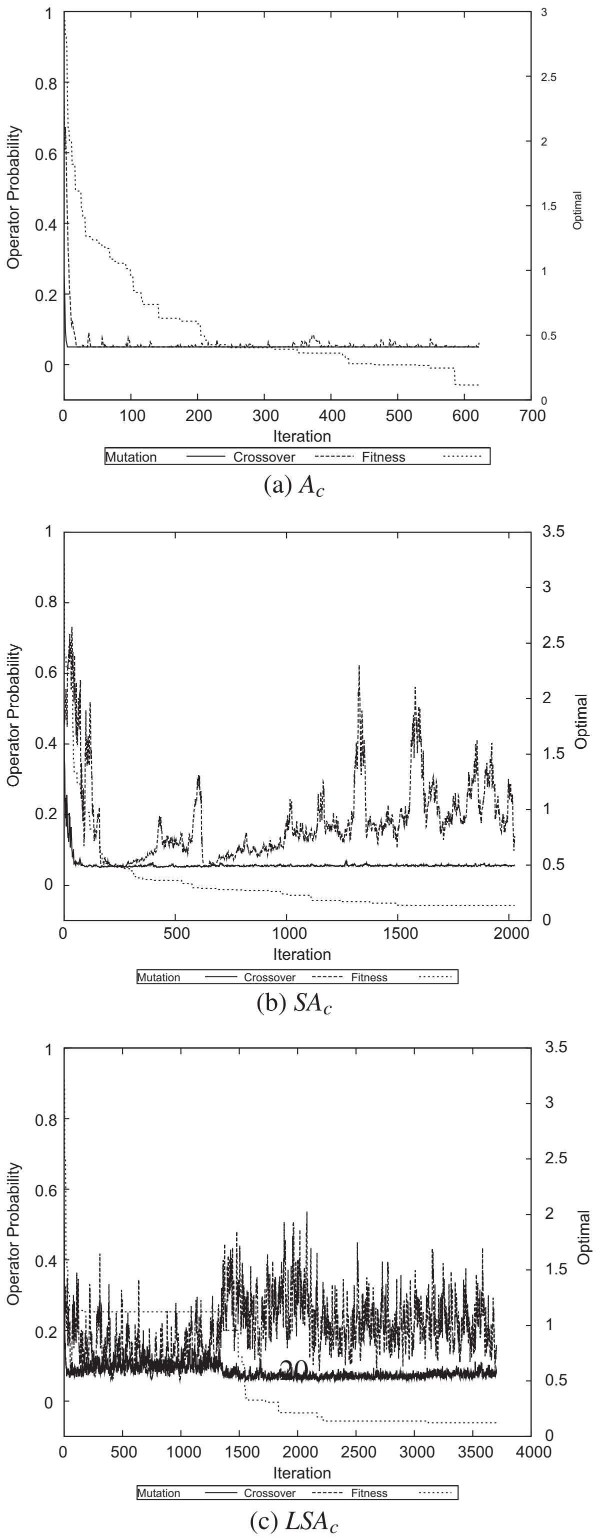
(b) SA_c



(c) LSA_c

Fig. 1. Convergence of control strategies in function f_4: Schaffer.

to the changes of the probabilities of the SA_c strategy which are guided by knowledge, unlike the LSA_c strategy where they are a random amount.



(a) A_c

(b) SA_c

(c) LSA_c

Fig. 2. Convergence of control strategies in function f_7: Griewank.

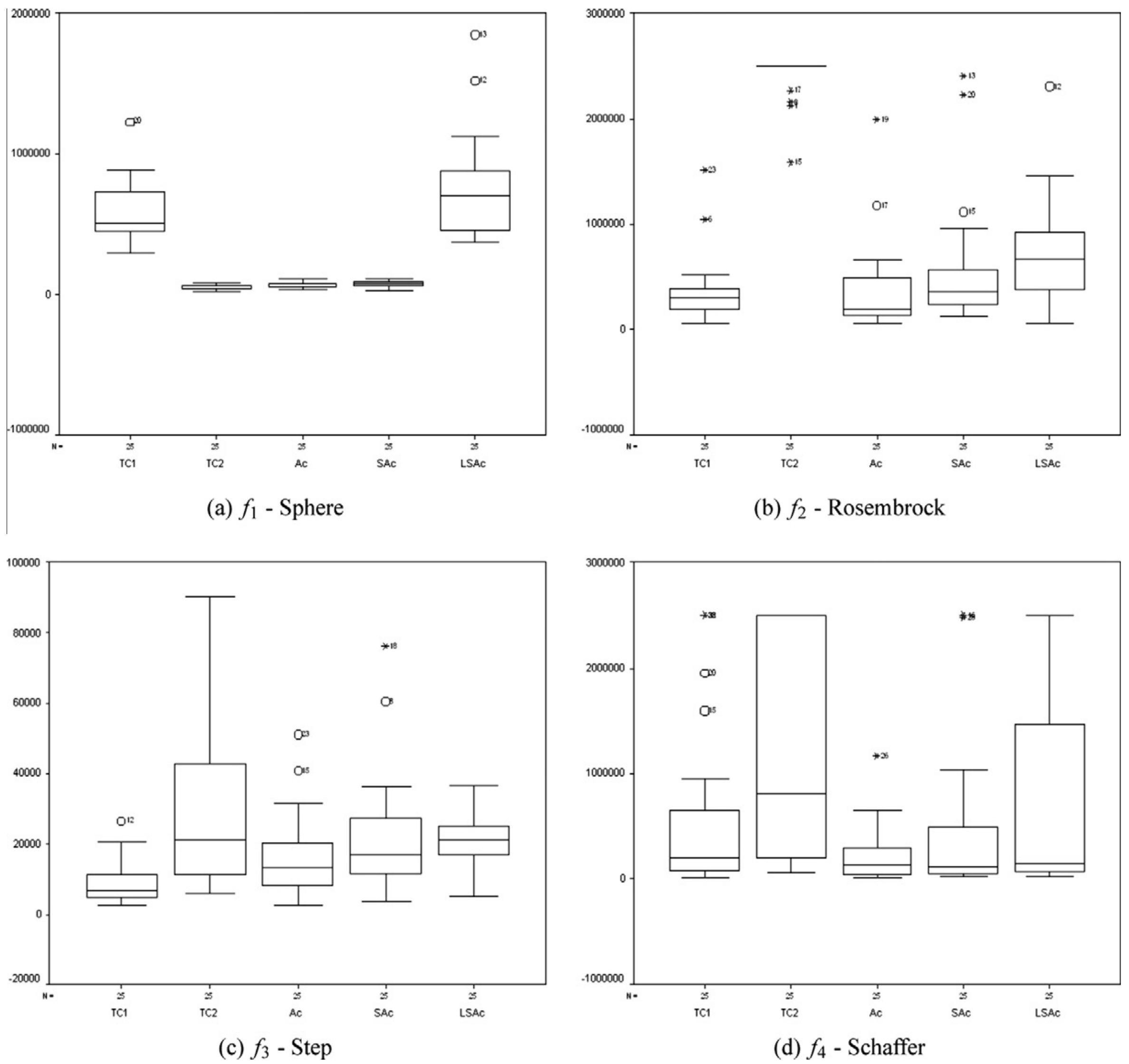


Fig. 3. Evaluations for Revac, Czarn et al., A_c, SA_c and LSA_c.

4.7. Statistical analysis

In order to appreciate the statistical significance of our techniques, in the following section we present the box-plots generated for each of the functions tested in respect to the number of evaluations. All these graphs correspond to 25 executions with different initial random seeds.

4.7.1. Box-plots

The TC_1 tuned algorithm has shown a better performance than TC_2 in relation to the number of evaluations. In all cases, we can observe a competitive performance of the control strategies compared to the performance shown by the tuned algorithms. Among the three control strategies, the A_c adaptive method shows a more stable behavior. For f_5 and f_6, LSA_c show a worse performance. It is explained by the fact that the algorithm for these functions requires a small mutation probability value. LSA_c is not able to accomplish it because its random updates of probability values periodically increase this probability value (see Figs. 3 and 4).

4.7.2. Statistical tests

To better analyze these results, we have done a statistical comparison of metaheuristic methods that have been used previously by [31]. In this case, we have performed the test to compare the performance of the adaptive control strategies

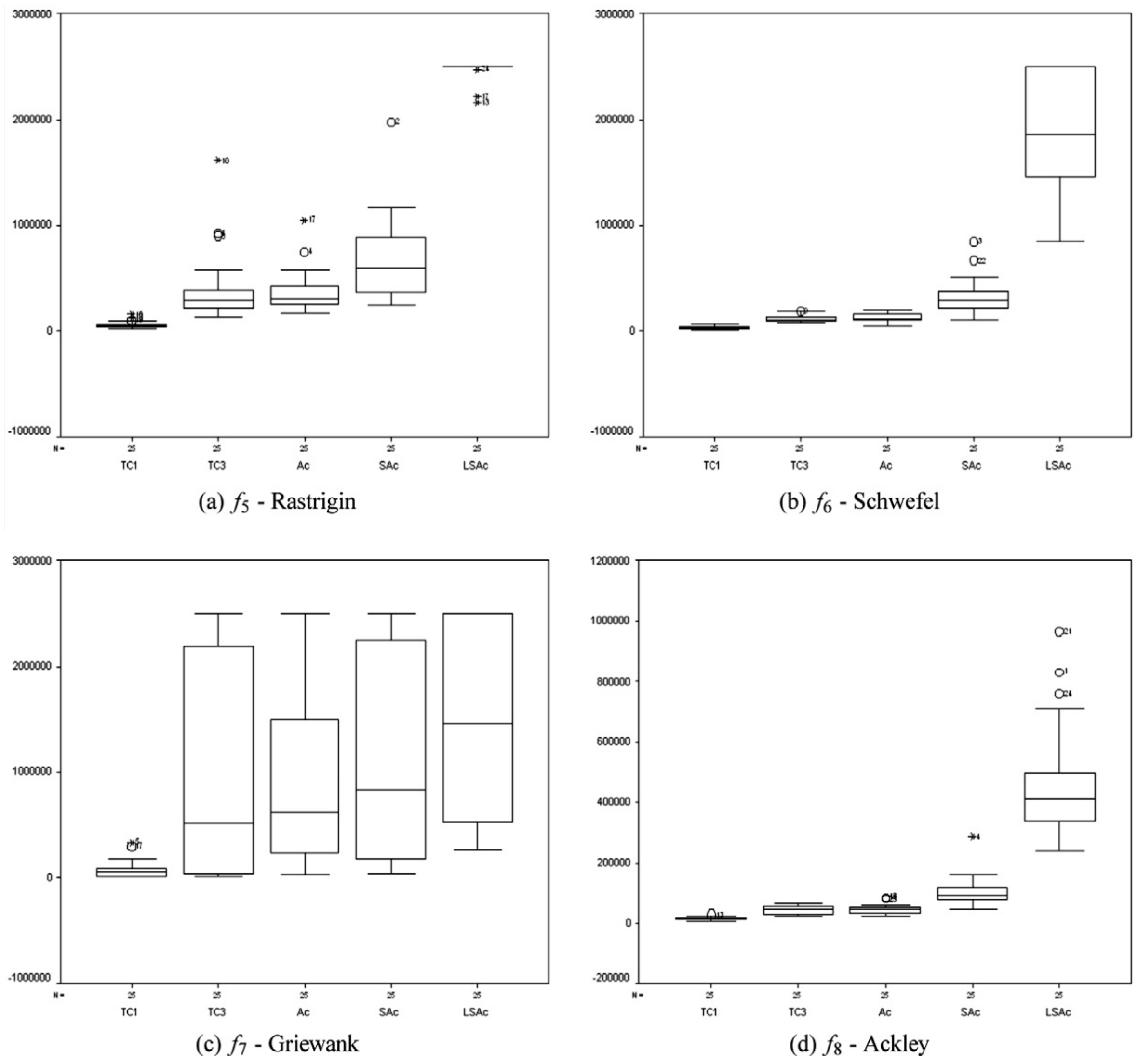


Fig. 4. Evaluations for Revac, ParamILS, A_c, SA_c and LSA_c.

Table 8
Wilcoxon signed ranks test – Schaffer function.

			N	Mean rank	Sum of ranks
$A_c - TC_1$	Negative ranks	$A_c < TC_1$	48	26.5	1272.00
	Positive ranks	$A_c > TC_1$	2	1.50	3.00
$SA_c - TC_1$	Negative ranks	$SA_c < TC_1$	46	27.2	1243.00
	Positive ranks	$SA_c > TC_1$	4	8.00	32.00
$LSA_c - TC_1$	Negative ranks	$LSA_c < TC_1$	19	25.79	490.00
	Positive ranks	$LSA_c > TC_1$	31	25.32	785.00
$SA_c - A_c$	Negative ranks	$SA_c < A_c$	7	10.29	72.00
	Positive ranks	$SA_c > A_c$	43	27.98	1203.00
$LSA_c - A_c$	Negative ranks	$LSA_c < A_c$	0	0.00	0.00
	Positive ranks	$LSA_c > A_c$	50	25.50	1275.00
$LSA_c - SA_c$	Negative ranks	$LSA_c < SA_c$	1	9.00	9.00
	Positive ranks	$LSA_c > SA_c$	49	25.84	1266.00

proposed to the best tuned algorithm for solving the Schaffer's function (f^4) and for solving the Griewank function (f^7). We performed 50 independent population runs for all algorithms. Using the information of the average number of evaluations

Table 9
Test statistics – Wilcoxon signed ranks test – Schaffer function – (a) based on positive ranks and (b) based on negative ranks.

	$A_c - TC_1$	$SA_c - TC_1$	$LSA_c - TC_1$	$SA_c - A_c$	$LSA_c - A_c$	$LSA_c - SA_c$
Z	−6.125(a)	−5.845(a)	−1.424(b)	−5.459(b)	−6.154(b)	−6.067(b)
Asymp. Sig. (2 − t)	0.00	0.00	0.154	0.00	0.00	0.00

Table 10
Wilcoxon signed ranks test: Griewank function – (a) it considers ties.

			N	Mean rank	Sum of ranks
$A_c - TC_1$	Negative ranks	$A_c < TC_1$	8	8.63	69.00
	Positive ranks	$A_c > TC_1$	42	28.71	1206.00
$SA_c - TC_1$	Negative ranks	$SA_c < TC_1$	3	8.33	25.00
	Positive ranks	$SA_c > TC_1$	47	26.60	1250.00
$LSA_c - TC_1$	Negative ranks	$LSA_c < TC_1$	0	0.00	0.00
	Positive ranks	$LSA_c > TC_1$	50	25.50	1275.00
$SA_c - A_c$	Negative ranks	$SA_c < A_c$	28	24.50	686.00
	Positive ranks	$SA_c > A_c$	21(a)	25.67	539.00
$LSA_c - A_c$	Negative ranks	$LSA_c < A_c$	12	23.33	280.00
	Positive ranks	$LSA_c > A_c$	35(a)	24.23	848.00
$LSA_c - SA_c$	Negative ranks	$LSA_c < SA_c$	17	19.24	327.00
	Positive ranks	$LSA_c > SA_c$	33	28.73	948.00

Table 11
Test statistics – Wilcoxon signed ranks test: Griewank function – (a) based on positive ranks and (b) based on negative ranks.

	$A_c - TC_1$	$SA_c - TC_1$	$LSA_c - TC_1$	$SA_c - A_c$	$LSA_c - A_c$	$LSA_c - SA_c$
Z	−5.488(a)	−5.913(a)	−6.154(a)	−0.731(b)	−3.005(a)	−2.997(a)
Asymp. sig. (2 − t)	0.00	0.00	0.00	0.465	0.003	0.003

done by the algorithms, we performed the pair-wise Wilcoxon test. The results are shown in Tables 8 and 9 for function f^4 and Tables 10 and 11 for f^7. All the computations have been performed by the statistical software package SPSS.

The pair-wise of the Wilcoxon test with Bonferroni corrected levels of observed significance of Table 9 for f_4 shows that the number of evaluations done by TC_1 are significantly higher than A_c and SA_c evaluations. However, it tends to be lower than LSA_c evaluations. The number of evaluations among our techniques are significantly lower for the A_c strategy.

We have done the same analysis for function f_7 from the Test Set II. Table 11 indicates that only between SA_c and A_c, the hypothesis for both algorithms to have the same average number of evaluations is accepted. There are significant differences among the others algorithms. From Table 10 we observe that compared to our three adaptive methods, TC_1 has always a much better mean rank. However, the mean rank computation does not include the millions of evaluations usually required by TC_1 to find the tuned parameter values. We have done the sensitivity analysis, and considered an additional 1500 evaluations for each run of the algorithm in Revac is sufficient to invert the results and to give a decisive advantage to our adaptive techniques. Therefore we can conclude that our techniques are significantly less consuming than TC_1.

4.8. General conclusions of the results

The three adaptive techniques include an overhead for the algorithm. One iteration made with SA_c is more time consuming than A_c and quite similar to LSA_c. Both methods SA_c and LSA_c perform a self-adaptation procedure and require doing extra individual evaluations and comparison tests. In addition, the A_c procedure does population evaluations and its adaptation process involves a set of individuals instead of a particular one. LSA_c does a random reinforcement to each individual, whereas for SA_c, this value strongly depends on the quality of the solution found by applying a specific genetic operator. It is important to point out that when our approach is included on an evolutionary algorithm which suffers from stacking on a local optima, our reinforcement mechanism will tend to decrease all the parameter probabilities. Thus, this information can be used as a criteria to stop the search.

5. Conclusions and future work

In this paper we have proposed two strategies for dynamic parameter control, an adaptive and a self-adaptive one. Both strategies help the standard genetic algorithm to do on the fly self-calibration. The results show that both adaptive and self-adaptive control strategies provide good results in terms of time required to solve a given problem.

One of the requirements for these strategies was to preserve the original design of the evolutionary algorithm. In this sense, our strategies can be included without changing the design of any operator. The main advantage of using adaptive strategies to improve the behavior of the standard genetic algorithm comes from nonexisting pre-execution time.

The tuning techniques can provide very good results to find tuned parameters for a given problem, but the parameter configuration step requires a huge amount of time compared to the time needed for the tuned algorithm to find a solution. However, as an adaptive strategy is not based on any given problem, the results cannot be as good as those obtained by the evolutionary algorithm that uses the best parameter values found by extensive hand-tuning for a particular instance. From the results obtained, we note that the adaptive technique proposed A_c globally requires less execution time to find a solution than the self-adaptive strategies. Both self-adaptive techniques, SA_c and LSA_c, show a quite similar overhead. We plan to work on the extension of the parameter control strategies to other population-based algorithms in which the parameter setting problem is essential in the definition of a good search algorithm. Another promising research area is the collaboration between different parameter control strategies.

Acknowledgments

Valuable comments from Z. Michalewicz and C. Coello as well as from the three anonymous referees and journal editor are gratefully acknowledged.

References

[1] B. Adenso-Diaz, M. Laguna, Fine-tuning of algorithms using fractional experimental designs and local search, Operation Research 54 (1) (2006) 99–114.
[2] S. Aine, R. Kumar, P. Chakrabarti, Adaptive parameter control of evolutionary algorithms to improve quality-time trade-off, Applied Soft Computing 9 (2) (2009) 527–540.
[3] J. Arabas, Z. Michalewicz, J. Mulawka, GAVaPS – a genetic algorithm with varying population size, in: Proceedings of the International Conference on Evolutionary Computation, 1994, pp.73–78.
[4] M. Birattari, T. Stützle, L. Paquete, K. Varrentrapp, A racing algorithm for configuring metaheuristics, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2002, pp. 11–18.
[5] A. Czarn, C. MacNish, K. Vijayan, B. Turlach, R. Gupta, Statistical exploratory analysis of genetic algorithms, IEEE Transactions on Evolutionary Computation 8 (4) (2004) 405–421.
[6] L. Davis, Adapting operator probabilities in genetic algorithms, in: Proceedings of the 3rd International Conference on Genetic Algorithms, Morgan Kaufmann, 1989, pp. 61–69.
[7] L. Davis, Handbook of genetic algorithms, Van Nostrand Reinhold Company, 1991.
[8] K. Deb, S. Agrawal, Understanding interactions among genetic algorithm parameters, Foundations of Genetic Algorithms, vol. V, 1999, pp. 265–286.
[9] A. Eiben, R. Hinterding, Z. Michalewicz, Parameter control in evolutionary algorithms, IEEE Transactions on Evolutionary Computation 3 (1999) 124–141.
[10] A. Eiben, E. Marchiori, V. Valkó, Evolutionary algorithms with on-the-fly population size adjustment, in: Proceedings of the Parallel Problem Solving from Nature, Springer, 2004, pp. 41–50.
[11] M. Gibbs, G.C. Dandy, H. Maier, A genetic algorithm calibration method based on convergence due to genetic drift, Information Sciences 178 (14) (2008) 2857–2869.
[12] J. Gómez, Self adaptation of operador rates in evolutionary algorithms, in: Proceedings of the Genetic and Evolutionary Computation Conference, Lecture Notes in Computer Science, vol. 3102, Springer, Berlin, 2004, pp. 1162–1173.
[13] A.B. Hashemi, M.R. Meybodi, A note on the learning automata based algorithms for adaptive parameter selection in PSO, Applied Soft Computing 11 (1) (2011) 689–705.
[14] R. Hinterding, Z. Michalewicz, A. Eiben, Adaptation in evolutionary computation: a survey, in: Proceedings of the IEEE International Conference on Evolutionary Computation, 1997, pp. 65–69.
[15] F. Hutter, H.H. Hoos, T. Stützle, Automatic algorithm configuration based on local search, in: Proceedings of the 22nd Conference on Artifical Intelligence, 2007, pp. 1152–1157.
[16] K.-D. Jong, An Analysis of the Behavior of a Class of Genetic Adaptive Systems, Ph.D. Thesis, University of Michigan, 1975.
[17] J. Laredo, C. Fernandes, J. Merelo, C. Gagné, Improving genetic algorithms performance via deterministic population shrinkage, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2009, pp. 819–826.
[18] F. Lobo, C. Lima, Z. Michalewicz (Eds.), Parameter Setting in Evolutionary Algorithms Studies in Computational Intelligence, vol. 54, Springer, 2007.
[19] F.G. Lobo, D.E. Goldberg, The parameter-less genetic algorithm in practice, Information Sciences 167 (1–4) (2004) 217–232.
[20] R. Mallipeddi, S. Mallipeddi, P. Suganthan, Ensemble strategies with adaptive evolutionary programming, Information Sciences 180 (9) (2010) 1571–1581.
[21] R. Mallipeddi, P. Suganthan, Evaluation of novel adaptive evolutionary programming on four constraint handling techniques, in: Proceedings of the IEEE World Congress on Computational Intelligence, 2008, pp. 4045–4052.
[22] J. Maturana, F. Saubion, A compass to guide genetic algorithms, in: Proceedings of the 10th International Conference on Parallel Problem Solving from Nature, Springer-Verlag, 2008, pp. 256–265.
[23] E. Montero, M. Riff, Self-calibrating strategies for evolutionary approaches that solve constrained combinatorial problems, Proceedings of the 17th International Symposium on Foundations of Intelligent Systems, vol. 4994, Springer, 2008, pp. 262–267.
[24] E. Montero, M. Riff, D. Basterrica, Improving MMAS using parameter control, in: Proceedings of the IEEE Congress on Evolutionary Computation, 2008, pp. 4007–4011.
[25] O. Montiel, O. Castillo, P. Melin, A.R. Díaz, R. Sepúlveda, Human evolutionary model: a new approach to optimization, Information Sciences 177 (10) (2007) 2075–2098.
[26] V. Nannen, A. Eiben, Relevance estimation and value calibration of evolutionary algorithm parameters, in: Proceedings of the Joint International Conference for Artificial Intelligence (IJCAI), 2006, pp. 975–980.
[27] R. Pavón, F. Díaz, R. Laza, V. Luzón, Automatic parameter tuning with a Bayesian case-based reasoning system. A case of study, Expert Systems with Applications 36 (2, Part 2) (2009) 3407–3420.
[28] J. Pettinger, R. Everson, Controlling Genetic Algorithms with Reinforcement Learning, Tech. Rep., Department of Computer Science, School of Engineering and Computer Science, University of Exeter, 2003.
[29] L. Poladian, Improving the success of recombination by varying broodsize and sibling rivalry, in: Proceedings of the 11th IEEE Congress on Evolutionary Computation, 2009, pp. 2439–2445.

[30] M.C. Riff, X. Bonnaire, Inheriting parents operators: a new dynamic strategy for improving evolutionary algorithms, in: Proceedings of the 13th International Symposium on Foundations of Intelligent Systems, 2002, pp. 333–341.
[31] O. Rossi-Doria, M. Sampels, M. Birattari, M. Chiarandini, M. Dorigo, L. Gambardella, J. Knowles, M. Manfrin, M. Mastrolilli, B. Paechter, L. Paquete, T. Stützle, A comparison of the performance of different metaheuristics on the timetabling problem, in: Proceedings of the 4th International Conference on Practice and Theory of Automated Timetabling, 2002, pp. 329–351.
[32] J. Smith, T. Fogarty, Operator and parameter adaptation in genetic algorithms, Soft Computing – A Fusion of Foundations, Methodologies and Applications 1 (2) (1997) 81–87.
[33] K. Srinivasa, K. Venugopal, L. Patnaik, A self-adaptive migration model genetic algorithm for data mining applications, Information Sciences 177 (20) (2007) 4295–4313.
[34] D. Thierens, Adaptive strategies for operator allocation, in: F. Lobo, C. Lima, Z. Michalewicz (Eds.), Parameter Setting in Evolutionary Algorithms, Springer, 2007, pp. 77–90.
[35] A. Tuson, P. Ross, Adapting operator settings in genetic algorithms, Evolutionary Computation 6 (2) (1998) 161–184.
[36] D. Whitley, K. Mathias, S. Rana, J. Dzubera, Building better test functions, in: Proceedings of the 6th International Conference on Genetic Algorithms, Morgan Kaufmann, 1995, pp. 239–246.
[37] B. Yuan, M. Gallagher, Combining Meta-EAs and racing for difficult EA parameter tuning tasks, in: F.G. Lobo, C.F. Lima, Z. Michalewicz (Eds.), Parameter Setting in Evolutionary Algorithms, Springer, 2007, pp. 121–142.