Calibrating Strategies for Evolutionary Algorithms

Elizabeth Montero and María-Cristina Riff

Abstract— **The control of parameters during the execution of evolutionary algorithms is an open research area. In this paper, we propose new parameter control strategies for evolutionary approaches, based on reinforcement learning ideas. Our approach provides efficient and low cost adaptive techniques for parameter control. Moreover, it is a general method, thus it could be applied to any evolutionary approach having more than one operator. We contrast our results with tuning techniques and HAEA a random parameter control.**

I. INTRODUCTION

When we design an evolutionary algorithm to address a specific problem we need to define a representation, and a set of operators to solve the problem. We also need to choose parameters values which we expect to 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 [5], [16].

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 like a generate and test procedure. Taking into account that a run of an evolutionary algorithm is an intrinsically dynamic adaptive process, we could expect that a dynamic adaptation of the parameters during the search could help to improve the performance of the algorithm, [1], [2], [8], [9], [4].

The idea of adapting and self-adapting parameters during the run is not new, but we need to manage a trade-off between both the improvement of the search and the adaptation cost. The key idea here is to monitor the search to be able to trigger actions from parameter control strategies, in order to improve the performance of the evolutionary algorithms.

This paper is organized as follows: In the next section we briefly present the related work. In section 3 we introduce our approach. In section 4 we present the performance evaluation. Finally, section 5 presents the conclusions and future work.

II. RELATED WORK

We can classify the parameter selection process into two different methods: *tuning* and *parameter control*, [3]. 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 an evolutionary algorithm. Usually, these parameter values are fixed for all the runs of the algorithm. Recently, a new technique called REVAC [10] has been proposed to do tuning in an efficient way. The idea is to use the entropy notion in an iterative process to converge upon a set of good values for the parameters.

Supported by Fondecyt Project 1060377

Department of Computer Science, Universidad Técnica Federico Santa María, Valparaíso, Chile, Elizabeth.Montero@inf.utfsm.cl, María-Cristina.Riff@inf.utfsm.cl

We can expect that the "best" parameter values depend on the step of the algorithm we are using. For that reason, the idea to design strategies to allow the algorithm to change its parameter values during its execution appears a promising way to improve the performance of an evolutionary algorithm. In this context many ideas have been proposed in the research community, amongst them we find self-adaptability processes where the individuals include information that can influence the parameter values and the evolution of these processes could use some criteria to produce changes, [7], [1]. Another interesting method has been proposed in [6]. In this approach the adaptation of the parameter values of a genetic algorithm are externally managed by an agent. It receives the search information from the genetic algorithm, and does a reinforcement learning task giving new parameter values to the algorithm. In this approach the Genetic Algorithm is not itself adaptive.

Researchers from the metaheuristics community have also been motivated on this subject proposing methods for parameter control or tuning. For instance, Hutter et al. [18] recently proposed an approach for stochastic local search algorithms. Birattari et al., [19] have proposed an statistical approach for tuning metaheuristics named Racing. A recent research of Eiben et al. [4] proposes a strategy to change the population size of the genetic algorithm taking into account the state of the search. Gómez introduced the Hybrid Adaptive Evolutionary Algorithm HAEA in [11], a self-adaptive dynamic parameter control for genetic operators. In his approach the algorithm randomly modifies the operator probabilities. He has applied his method to solve continuous optimization problems. The probability of an operator could be increased in the case of positive behavior or decreased in case of a negative one. The reward and penalty values are randomly generated.

We introduce in this paper two strategies which are able to implement efficient parameter control using the idea of taking into account the evolution, and taking some action during the execution in order to accelerate the convergence process.

III. 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 do 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 the

1-4244-1340-0/07/$25.00 ©2007 IEEE

tests to find the best combination can not be done in the whole search space some methods have been proposed in order to define a reduced but significant set of tests. The most promising technique in this context is REVAC [10].
The principal problem with using tuning is that the combination of the parameter values found is on *average* the best set evaluated using the set of tests. In our experience the result of tuning can lead to wrong conclusions. For example, given a set of parameter values for the operators 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 be applied to guide the search of the evolutionary algorithm. We clearly verify this situation in the tests presented in section 4.

Our aim is to propose and to evaluate two kinds of strategies that allow parameter adaptation in an evolutionary approach according to the problem in hand. We propose two types of reinforcement control: A Self-Adaptive (SA_c) and An Adaptive (A_c) one. Both of them will be evaluated in the experiments results section.

A. Self-Adaptive Control: SA_c

The idea is to make 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 probabilities values. Roughly speaking, an operator receives a reward when its application generates a better individual than its parent. Analogously, it receives a penalty when the offspring has a worse fitness value than its parent. In our approach, both the rewards and the penalties strongly depend on the evaluation function value.

Definition 3.1: 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 an offspring C_j. We define a success measure for the operator O_k in its $a-th$ application, $S_a(O_k)$ as:

$$S_a(O_k) = F(C_j) - F(P_h) \quad (1)$$

where $F(C_j)$ and $F(P_h)$ are the respective fitness of the child and of the parent h.

Remark 3.1: The value of $S_a(O_k)$ is positive when the child generated by the operator O_k in its a_{th} application has a better evaluation than its parent P_h.

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

Definition 3.2: Given a set of operators O_k, $k = o_1, .., o_M$ with a success measure $S_a(O_k) \geq 0$, $M \leq p$, we define $Max - i_l$ as the maximum improvement done by the operators during the last l generations as:

$$Max - i_l = Argmax_{a=1,..,A_k,k=o_1,..,o_M}(S_a(O_k)) \quad (2)$$

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 fitness function in the last l iterations.

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

Definition 3.3: Given a set of operators O_k, $k = o_1, .., o_m$ with a success measure $S_a(O_k) < 0$, $m \leq p$, we define $Max - d_l$ as the maximum degradation done by the operators during the last l generations as:

$$Max - d_l = Argmax_{a=1,..,A_k,k=o_1,..,o_m}(|S_a(O_k)|) \quad (3)$$

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.

Using these functions the algorithm computes the new probability of the operators, given a reinforcement to the genetic operator, proportionaly to the quality of the child that it has generated. In the same way, an operator will be penalized if the childs it has generated are worse than their parents.

More formally,

Definition 3.4: Given the application a 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:

$$P_{C_j}(O_k) = P_h(O_k) + \frac{S_a(O_k)}{Max_l} \quad (4)$$

where,

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

Thus, each child generated in the application a_{th} by applying the operator O_k has its probability value computed by the equation 4 included in its representation.

B. Adaptive Control: A_c

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

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

Definition 3.5: Given a set of operators O_k, $k = o_1, .., o_M$ and the average success measure $\overline{S_a(O_k)} \geq 0$, $M \leq p$,

we define $Max - ia_l$ as the average maximum improvement done by the operators during the last l generations as:

$$Max - ia_l = Argmax_{a=1,..,A_k,k=o_1,..,o_M}(\overline{S_a(O_k)}) \quad (6)$$

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 at 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 3.6: Given a set of operators O_k, $k = o_1, .., o_m$ and the average success measure $\overline{S_a(O_k)} < 0$, $m \leq p$, we define $Max - da_l$ as the average maximum degradation done by the operators during the last l generations as:

$$Max - da_l = Argmax_{a=1,..,A_k,k=o_1,..,o_m}(\overline{|S_a(O_k)|}) \quad (7)$$

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

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

Definition 3.7: 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:

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

where

$$Max_{al} = \begin{cases} Max - ia_l & \text{if } \overline{S_a(O_k)} \geq 0 \\ Max - da_l & \text{otherwise} \end{cases} \quad (9)$$

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 3.2: Thierens [17] analyses the adaptive allocation rules for adaptive parameter control. He presents a discussion about the techniques called as Probability Matching, those techniques that 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 for obtaining a reward, and all of them are considered in the same way. By contrary, our approach distinguishes between the set of good operators and the worse ones. It considers two "competitions", good operators compete for obtaining a higher reward and the worse operators compete for having 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.

C. Observation

Both strategies (A_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 childs 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.

.

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

IV. Experimental Results

The goal of the following experiments is to evaluate the performance of our dynamic parameter control strategies.

A. Standard Genetic Algorithm and Functions

For testing we used the same algorithm and conditions established in two well-known research publications, both related to parameter control strategies: [12] and [10]. There are:

- Functions tested are f_1, f_2, f_3, [13] and f_6, [14]. They are shown in table I.
- The algorithm is the standard genetic one that uses binary representation, elitism, one-point crossover and mutation as usual.
- The initial probability for each operator is fixed in $\frac{1}{p}$, where p is the number of genetic operators. Thus, the algorithm does not need to be previously tunned respect to its operator probabilities.
- The algorithm runs 10 times for each function. The population size is fixed at 50. The stopping criteria for f_1, f_2 and f_3 is when the distance between the current best solution and the optimal one is lower than 10^{-10}, and for f_6 is when the distance is lower than 10^{-6} from zero, that is the optimal value.

$f_1 = \sum_{i=1}^{3} {x_i}^2$,	$-5.12 \leq x_i \leq 5.12$
$f_2 = 100 \cdot (x_2 - {x_i}^2)^2 + (1 - x_1)^2$,	$-2.048 \leq x_i \leq 2.048$
$f_3 = 30 + \sum_{i=1}^{5} \lfloor x_i \rfloor$,	$-5.12 \leq x_i \leq 5.12$
$f_6 = 0.5 + \frac{(sin(\sqrt{x_1^2 + x_2^2}))^2 - 0.5}{(1 + 0.0001 \cdot (x_1^2 + x_2^2)^2)}$,	$-100 \leq x_i \leq 100$

TABLE I
Functions Tested

B. Hardware

The hardware platform for the experiments is an Intel Dual Core 3.4 Ghz PC with 512 MBytes of RAM, running Mandriva Linux 2007 operating system. The algorithms have been implemented in C++. We use the g++ optimizer.

C. Evaluation of the Adaptive and Self-Adaptive Strategies

In this section we present the evaluation of our strategies. We compare both the success rate and time spent of the algorithms using various techniques for parameter control:

- The algorithm tunned with two methods: Nannen and Eiben parameter values determined using REVAC algorithm (TC_1) and Czarn et al. statistically estimated parameter values (TC_2), who have proposed different parameter values depending on the function to be solved. These parameter values are shown in table II. We remark that the parameter values are quite different. For instance, for the functions f_2 and f_6, TC_2 considers that crossover must not be applied, however for TC_1 crossover has a higher probability than mutation.

Function	Nannen & Eiben (TC_1)		Czarn et al. (TC_2)	
	p_c	p_m	p_c	p_m
f_1	0.90	0.012	1.00	0.0677
f_2	0.82	0.0146	0.00	0.2115
f_3	0.98	0.0338	1.00	0.0511
f_6	0.60	0.0604	0.00	0.1501

TABLE II
CROSSOVER AND MUTATION PROBABILITY VALUES TUNNED

- For Adaptive Control: The Hybrid Adaptive Evolutionary Algorithm HAEA described in section 2 and our new techniques SA_c, A_c.

We report the results obtained in table III for tuning. For each function, the table shows both the final iteration and the average time spent by each approach in seconds. We note that:

1) The algorithm behaviour strongly depends on the parameter values fixed at the beginning and the function to be solved.
2) The average time required by TC_1 for tuning the algorithm for one function was around 8 hours CPU Time. This time is invested in 1000 iterations required by REVAC using 10 different seeds for each problem.
3) The results obtained by TC_1 are quite similar to the ones from the best statistical-obtained values

The previous three 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. In the following table we report the results obtained using HAEA, SA_c and A_c.

Function	TC_1		TC_2	
	Final iteration	time [s]	Final iteration	time [s]
f_1	645.2	0.9	1003.7	1.3
f_2	16326.9	15.5	1224.9	1.3
f_3	176.6	0.3	239.6	0.4
f_6	20295	20.3	16117.8	16.6

TABLE III
SUCCESS RATE COMPARISON - STANDARD GENETIC ALGORITHM WITH TUNING

Function	HAEA		SA_c		A_c	
	Final iteration	time [s]	Final iteration	time [s]	Final iteration	time [s]
f_1	3434.9	9	1276.3	5.1	5257.1	12.8
f_2	1822.7	5.5	2014.1	6.2	1904.6	3.1
f_3	96.7	0.4	135.7	0.6	98.9	0.6
f_6	12564.5	28.9	2457.4	8.1	2870.5	4.6

TABLE IV
SUCCESS RATE COMPARISON - STANDARD GENETIC ALGORITHM WITH SELF-ADAPTIVE AND ADAPTIVE CONTROLS

The three adaptive techniques include an overhead for the algorithm. One iteration done with SA_c is more expensive in time than A_c and quite similar to HAEA. Both methods SA_c and HAEA do a self-adaptation procedure and require to do an extra individual evaluations and comparison tests. In addition, the A_c procedure does population evaluations and the adaptation involves a set of individuals and not a particular one. HAEA does a random reinforcement to an individual, where-as in our approach, this value strongly depends on the quality of the solution found by applying a specific genetic operator. The standard deviation of the number of generations required to reach the goal for both techniques are shown in table V.

Function	SA_c	A_c
f_1	491.7	2096.5
f_2	898.6	1016.8
f_3	71.6	19.9
f_6	2461.0	2052.2

TABLE V
STANDARD DEVIATION OF THE NUMBER OF GENERATIONS TO REACH THE GOAL

According to [15], there are some special characteristics which make f_6 more difficult problem to be solved and therefore, a more interesting one for testing genetic algorithms. It is a non separable and non linear function, and moreover multiple passes of line search cannot yield competitive solutions. This is not the case for the function f_1 which is separable and can always be solved by line search. For the function f_6, our approaches outperform all the other strategies considered in this work. The time used by A_c is four times lower than the time spent by TC_2.

Remark 4.1: It is important to signal that when our approach is included on an evolutionary algorithm that suffers of stucking on a local optima our reinforcement mechanism will tend to decrease all the parameter probabilities. Thus, this information can be used as a stop criteria of the search.

D. Convergence for SA_c

We present the convergence graphs for SA_c in figures 1, 2, 3, 4. The graphs shown the operators probability by generations. The algorithm does different calibrations according to the functions.

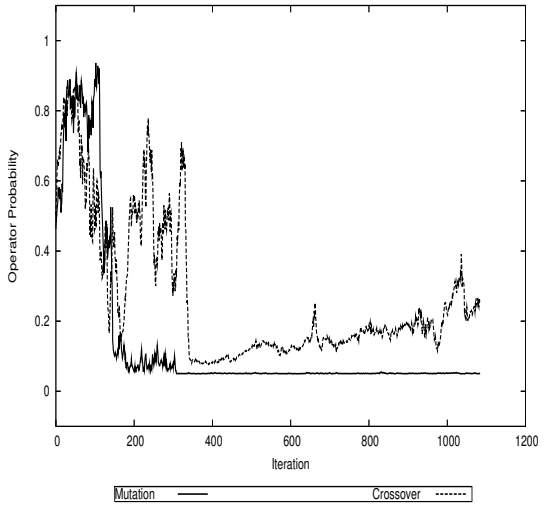

Fig. 1. f_1: Operators Probability, Fitness and Iterations

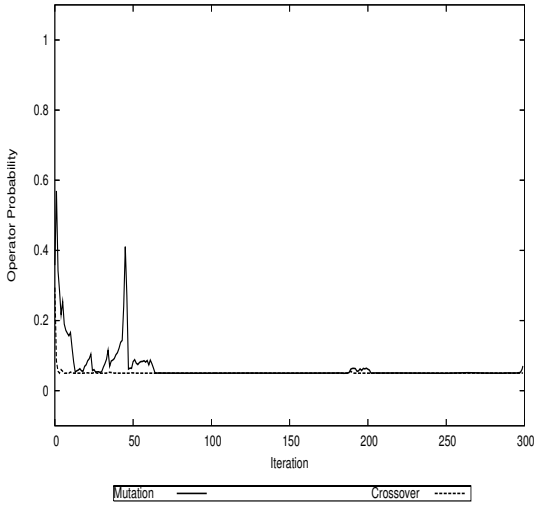

Fig. 2. f_2: Operators Probability, Fitness and Iterations

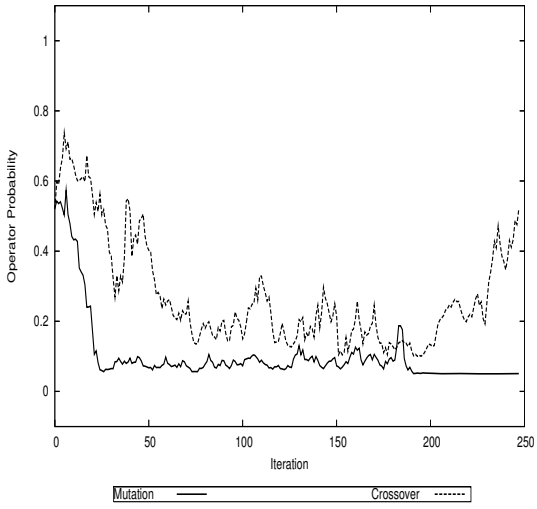

Fig. 3. f_3: Operators Probability, Fitness and Iterations

V. Conclusions

We propose in this paper 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 have good results in terms of time required to solve a given problem. The main advantage of using adaptive strategies to improve the behaviour of the standard genetic algorithm comes from non existing pre-execution time. REVAC can obtain very good results to find tuned parameters for a given problem, but the parameters configuration step requires a huge amount of time compared to the one needed for the tuned algorithm to find a solution. As an adapting strategy does not rely on a given problem, the obtained results cannot be as good as the best hand-obtained tuning values. However, hand tuning is also a time consuming task, as it required several executions of the same problem to find the best suitable values for the parameters. From the results obtained we note that SA_c and A_c globally require less execution time to find a solution than HAEA, when SA_c and HAEA have a quite similar overhead.

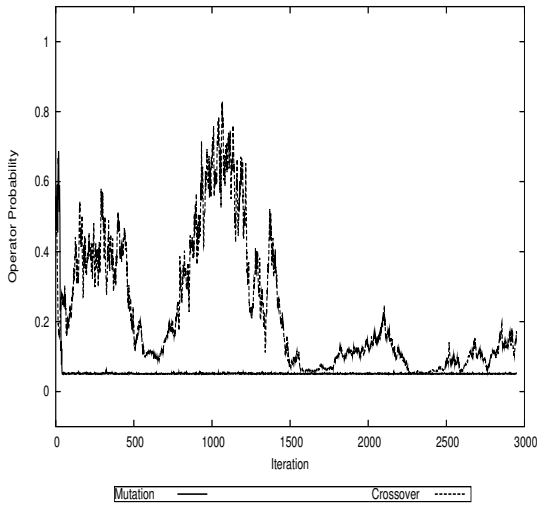

Fig. 4. f_6: Operators Probability, Fitness and Iterations

VI. Future Work

A promising research area is the collaboration between various parameter control strategies. We will test our strategies with more complex functions.

References

[1] L. Davis. *Adapting Operator Probabilities in Genetic Algorithms* In *Proceedings of 3rd. International Conf. on Genetic Algorithms and their Applications*, pp. 61-69, 1989

[2] K.Deb and S. Agrawal. *Understanding Interactions among Genetic Algorithms Parameters* In *Foundations of Genetic Algorithms*, Vol. 5, pp. 265-286, 1998

[3] A.E. Eiben, R. Hinterding and Z. Michalewicz. *Parameter Control in Evolutionary Algorithms IEEE Transactions on evolutionary computation*, Vol. 3(2):124-141, 1999.

[4] A. E. Eiben, E. Marchiori and V.A. Valko. *Evolutionary Algorithms with on-the-fly Population Size Adjustment PPSN VIII*, *Lecture Notes in Computer Science*, Vol.3242, pp. 41-50, 2004.

[5] R. Hinterding, Z. Michalewicz and A.E. Eiben. *Parameter Control in Evolutionary Computation* In *IEEE Transactions on Evolutionary Computation*, Vol. 3, No. 2, pp. 124-141, 1999.

[6] J. Pettinger and R.Everson. *Controlling Genetic Algorithms with Reinforcement Learning Department of Computer Science, University of Exeter*, Technical Report EX4 4QF, UK, 2003.

[7] M-C Riff and X. Bonnaire. *Inheriting Parents Operators: A New Dynamic Strategy to improve Evolutionary Algorithms Foundations of Intelligent Systems, ISMIS'2002*, *Lectures Notes on Computer Science* Vol. 2366, pp. 333-341, 2002.

[8] J. Smith and T.C. Fogarty. *Operator and parameter adaptation in genetic algorithms* In *Soft Computing*, Vol. 1(2), pp. 81-87, 1997.

[9] A. Tuson A. and P. Ross. *Adapting Operator Settings in Genetic Algorithms* In *Evolutionary Computation*, Vol.(6):2, pp. 161-184, 1998.

[10] V. Nannen and A.E. Eiben Relevance Estimation and Value Calibration of Evolutionary Algorithm Parameters In *Proceedings of the Joint International Conference for Artificial Intelligence (IJCAI)*, pp. 975-980, 2007.

[11] J. Gómez. *Self Adaptation of Operator Rates in Evolutionary Algorithms*, In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1162-1173, 2004.

[12] A. Czarn, C. MacNish, K. Vijayan, and B. Turlach. *Statistical exploratory analysis of genetic algorithms: the influence of gray codes upon the difficulty of a problem* In *Proc. 17th Australian Joint Conference on Artificial Intelligence (AI 2004)*, *Lecture Notes in Artificial Intelligence*, Vol. 3339, pp. 1246-1252, 2004.

[13] K. A. De Jong. *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, Ann Arbor, 1975.

[14] L. Davis. *Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York,NY*, 1991.

[15] D. Whitley, K. Mathias, S. Rana and J. Dzubera. *Building Better Tests Functions Proceedings of the 6th International Conference on Genetic Algorithms*, pp. 239-247, 1995.

[16] F. Lobo C., F. Lima, and Z. Michalewicz *Parameter Setting in Evolutionary Algorithms (Studies in Computational Intelligence) Springer*, Vol. 54, 2007.

[17] D. Thierens *Adaptive Strategies for Operator Allocation Studies in Computational Intelligence 54,* pp. 77-90, 2007.

[18] F. Hutter, Y. Hamadi, H. Hoos and K. Leyton-Brown *Performance Prediction and Automated Tuning of Randomized and Parametric Algorithms* 12th International Conference on Principles and Practice of Constraint Programming, France, *Lecture Notes in Computer Science*, Vol. 4204, pp. 213-228, 2006.

[19] M. Birattari, T. Stutzle, L. Paquete, and K. Varrentrapp (2002). *A Racing Algorithm for Configuring Metaheuristics Proceedings of the Genetic and Evolutionary Computation Conference GECCO*, pp. 11-18, 2002.