A generic approach to parameter control

Giorgos Karafotias, S.K. Smit, A.E. Eiben

Vrije Universiteit Amsterdam

Abstract. On-line control of EA parameters is an approach to parameter setting that offers the advantage of values changing during the run. In this paper, we investigate parameter control from a generic and parameter-independent perspective. We propose a generic control mechanism that is targeted to repetitive applications, can be applied to any numeric parameter and is tailored to specific types of problems through an off-line calibration process. We present proof-of-concept experiments using this mechanism to control the mutation step size of an Evolutionary Strategy (ES). Results show that our method is viable and performs very well, compared to the tuning approach and traditional control methods.

1 Introduction

When defining an evolutionary algorithm (EA) one needs to configure various settings: choose components (such as variation and selection mechanisms) and set numeric values (e.g. the probability of mutation or the tournament size). These configurations largely influence performance making them an important aspect of algorithm design.

The field of evolutionary computing (EC) traditionally distinguishes two approaches for setting parameter values[4]:

- *Parameter tuning*, where parameter values are fixed in the initialization stage and do not change while the EA is running.
- *Parameter control*, where parameter values are given an initial value when starting the EA and undergo changes while the EA is running.

The capability of parameter control to use adequate parameter values in different stages of the search is an advantage, because the run of an EA is an intrinsically dynamic process. It is intuitively clear –and for some EA parameters theoretically proven– that different values may be optimal at different stages of the evolution. This implies that the use of static parameters is inherently inferior to changing parameter values on-the-fly.

The conceptual distinction between tuning and control can be lifted if we consider the control mechanism as an integral part of the EA. In this case the EA and its parameter control mechanism (that may be absent) are considered as one entity. Furthermore, this composed entity may or may not be tuned before being applied to a new problem. The resulting matrix of four options is shown in Figure 1.

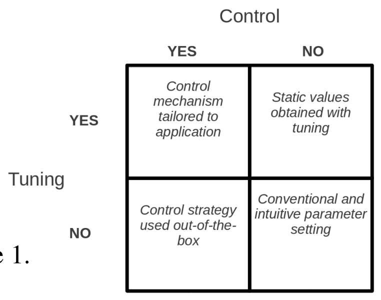
Control
YES
NO
Tuning
YES
NO
Control mechanism tailored to application
Static values obtained with tuning
Control strategy used out-of-the-box
Conventional and intuitive parameter setting

Fig. 1. Tuning vs control

The combinations in the top row have the advantage of enhanced performance at the cost of the tuning effort [11]. The options in the left column offer the benefits of time varying parameter values mentioned above with a tradeoff of increased complexity.

In this paper we introduce a generic control mechanism that is located in the top-left corner of the matrix, i.e. it combines on-line parameter adjustment (control) and off-line configuration (tuning). This means that the evolutionary algorithm incorporates a parameter control mechanism (seen as a black box for the moment) but this controller itself has certain parameters that can be configured for a problem through an off-line tuning process. Such a method has both the advantage of enhanced performance, and the possibility to be tuned to a specific problem (and is therefore most appropriate for repetitive problems). On the other hand, this mechanism also has the disadvantages of being complex, and the need for computational time dedicated to problem tailoring. Hence, the questions we want to address in this paper are:

- Is such an approach viable?
- What is the added value of this approach, when compared to traditional approaches such as static parameter-values, the 1/5th rule and self-adaptation?
- On what kind of feedback from the search process should such a parameter control mechanism base its decisions?

2 Related work

Parameter control is an increasingly popular topic in the field of evolutionary algorithms[5]. The outline of the most commonly used methods is quite similar: one of the parameter values is altered based on some specific evidence. Most often these methods are designed for specific parameters. The most popular parameter-specific control methods focuses on mutation probability [6], mutation step size [12] and operator selection [17] but methods also exist for the selection pressure [18], the population-size[16], the fitness function [9] and the encoding [13].

Some generic control methods for numeric parameters also exist. In [19] an adaptive mechanism is proposed that works in alternating epochs, first evaluating parameter values in a limited set and then applying them probabilistically. In the end of every such pair of epochs the set of possible parameter values is updated according to some heuristic rule. In Lee and Takagi [8] an adaptive control mechanism based on fuzzy logic is proposed. Instantiation of the rule set of the controller is achieved through an off-line calibration process using a GA. Lee and Takagi concluded that such an approach was very beneficial, and led to a much better performance than using the fixed parameter values. However, the fixed values used in this study were the ones commonly used at that time, based on the early work of DeJong, rather than found using parameter tuning.

A two-layer approach to control is presented in [3] and [10]: the lower layer adaptively controls EA parameters driven by an upper level that enforces a user-defined schedule of diversity or exploration-exploitation balance (though these are not parameters per se). The algorithm in [10] includes a built-in learning phase that calibrates the controller to the EA and problem at hand by associating parameter values to diversity and mean fitness using random samples. In [3], the lower control level is an adaptive operator selection method that scores operators according to the diversity-fitness balance they achieve as compared to a balance dictated by the upper level user defined schedule. However, neither of the two make a comparison against static parameter-values found using parameter tuning.

Extensive literature reviews on parameter control can be found in [5] and [2].

3 Parameter Control Roadmap

In this section we present a simple framework for parameter control mechanisms. The purpose of this framework is not to provide any theoretical grounding or proofs but to serve as a roadmap that helps in designing and positioning one's mechanism.

We define a parameter control mechanism as a combination of three components:

1. A choice of parameters (i.e. *what* is to be controlled).
2. A set of observables that will be the input to the control mechanism (i.e. what *evidence* is used).
3. An algorithm/technique that will map observables to parameter values (i.e. *how* the control is performed).

These components are briefly described in the following paragraphs. However, they are based on the definition of the state of an evolutionary algorithm, which is therefore introduced first.

EA State We define the state S_{EA} of an evolutionary algorithm as:

$$S_{EA} = \{G, \bar{p}, \mathcal{F}\} \tag{1}$$

where G is the set of all the genomes in the population, $\bar{p}$ is the vector of current parameter values, and $\mathcal{F}$ is the fitness function.

A triple S_{EA} uniquely specifies the state of the search process for a given evolutionary algorithm (the design and specific components of the EA need not be included in the state since they are the same during the whole run) in the sense that S_{EA} fully defines the search results so far and is the only observable factor that influences the search process from this point on (though not fully defining it, given the stochastic nature of EA operators). Time is not part of S_{EA} as it is irrelevant to the state itself; it introduces an artificial uniqueness and a property that is unrelated to the evolution. Of course, state transitions are not deterministic.

3.1 Parameters

The starting point when designing a control mechanism is the parameter to be controlled (as well as choices such as when and how often the parameter is updated). The importance of various parameters and the effect or merit of controlling each of them are subjects that will not be treated here (we refer to [2]). Instead, here we will only distinguish between *numeric* (e.g. population size, crossover probability) and *symbolic* (e.g. recombination operator) parameters.

3.2 Observables

The observables are the values that serve as inputs to the controller's algorithm. Each observable must originate from the current state S_{EA} of the EA since, as defined above, it is the only observable factor defining how the search will proceed from this point on.

However, the raw data in the state itself are unwieldy: if we were to control based on state S_{EA} directly, that would imply that the control algorithm should be able to map every possible S_{EA} to proper parameter values. Consequently, preprocessing is necessary to derive some useful abstraction, similar to the practise of dataset preprocessing in the field of data mining. We define such an observable derivation process as the following pipeline:

$$Source \rightarrow (Digest) \rightarrow (Derivative) \rightarrow (History)$$

Parentheses denote that steps can be bypassed.

i. *Source*: As stated above, the source of all observables is the current state of the EA, i.e. the set of all genomes, the current parameter values and the fitness function.
ii. *Digest*: A function $D(S_{EA}) = v$ that maps an EA state to a value, e.g. best fitness or population diversity.
iii. *Derivative*: Instead of using directly a value v we might be more interested in its speed or acceleration (e.g. to make the observable independent to the absolute values of v or to determine the effect of the previous update as the change observed in the most recent cycle).
iv. *History*: The last step in defining an observable is maintaining a history of size W of the value received from the previous step. This step includes a decision on the sliding window size W and the definition of a function $F_H(v_1, v_2, ..., v_W)$[1] that, given the last W values, provides a final value or vector (e.g. the minimum value, the maximum increase between two consecutive steps, the whole history as is etc.).

The above observable derivation is meant to be a conceptual framework and not an implementation methodology. For example, the current success ratio (in the context of Rechenberg's 1/5 rule) can in theory be derived from a state S_{EA} by applying the selection and variation operators to G and calculating the fitnesses of the results though obviously that would be a senseless implementation.

3.3 Algorithm

Any technique that maps a vector of observable values to a vector of parameter values can be used as an algorithm for the control mechanism, e.g. a rule set, an ANN or a function are all valid candidates. The choice of the proper algorithm seems to bear some resemblance to choosing an appropriate machine learning technique given a specific task or dataset. Whether EA observables display specific characteristics that make certain biases and representations more suitable is a question that needs to be investigated. In any case, it is obvious that given the type of parameter controlled (i.e. numeric or nominal) different techniques are applicable.

[1] Notice that indices have no relation to time but merely indicate a sequence of W elements

Here we distinguish between two main categories of control techniques, regardless the algorithm and representation used, based on a fundamental characteristic of the controller: whether it is static or it adapts itself to the evolutionary process.

i. *Static*: A static controller remains fixed during the run, i.e. given the same observables input it will always produce the same parameter values output. In other words, the values produced only depend on the current observables input:

$$\boldsymbol{p} = c(\boldsymbol{o}) \;\; and \;\; \boldsymbol{o_1} = \boldsymbol{o_2} \Rightarrow c(\boldsymbol{o_1}) = c(\boldsymbol{o_2})$$

where $\boldsymbol{o} \in O$, $\boldsymbol{p} \in P$ are the vectors of observables and parameter values respectively and $c : O \mapsto P$ is the mapping of the controller.

ii. *Dynamic*: A dynamic controller changes during the run, i.e. the same observables input can produce different parameter values output at different times. This implies that the controller is stateful and that the values produced depend on both the current observables input and the controller's current state:

$$\boldsymbol{p} = c_p(\boldsymbol{o}, S_C) \;\; and \;\; S_C^{t+1} = c_S(\boldsymbol{o_t}, S_C^t)$$

where $\boldsymbol{o} \in O$, $\boldsymbol{p} \in P$ are the vectors of observables and parameter values respectively, $S_C \in \mathcal{S}$ is the state of the controller and $c_p : O \times \mathcal{S} \mapsto P$, $c_S : O \times \mathcal{S} \mapsto \mathcal{S}$ are the mappings of the controller.

According to this classification, a time-scheduled mechanism is a trivial case of a dynamic controller; it maintains a changing state (a simple counter) but is "blind" to the evolutionary process since it does not use any observables. It should be noted that we do not consider control mechanisms necessarily as separate and distinct components, e.g. we classify self-adaptation in ES as a dynamic controller since it implicitly maintains a state influenced by the evolutionary process.

4 Experimental Setup

The experiments presented here are designed as a proof of concept for the viability of a generic, EA-independent and parameter-independent control mechanism that is instantiated through an off-line tuning process and targeted to repetitive applications. That means that the present configuration belongs in the upper left square of Fig 1, i.e. it combines on-line control of the EA parameters and off-line tuning of the controller to a specific kind of problem.

The parameter we chose for our initial control experiments is the mutation step size σ in evolution strategies. The specific parameter may seem a trivial choice given the number and efficiency of existing control techniques but its simplicity and the existence of related theory and practical experience make σ a parameter suitable for analysis.

4.1 Evolutionary algorithm

The EA used, is a $(10 + \lambda)$ ES with Gaussian mutation. It has no recombination and uses uniform random parent selection.

4.2 Parameter

As stated above, the controlled parameter is the mutation step size σ. In this experiment, σ will be updated in every generation from the start of the run.

4.3 Observables

The observables that act as input to the controller, are based on the current parameter values, diversity and fitness. The first input, the current σ, is input directly without going through the digest, derivative and history stages of the pipeline. The second input is the diversity, using the Population Diversity Index (PDI) [15] as the digest function and bypassing derivatives and history. Finally, we use two different fitness-based observables: (i) f_N uses a digest of the best fitness normalized in $[0, 1]$ and no history, (ii) Δf uses a best fitness digest (f_B) and a history with length W and the history function

$$F_H(f_B^1, ..., f_B^W) = \frac{f_B^W - f_B^{W/2}}{f_B^W - f_B^1}$$

We choose to use this formula to measure change instead of a derivative to make the controller robust to shifting and stretching of the fitness landscape.

We compare the two fitness-based observables on their own as well as paired with the diversity observable. The Δf observable is combined with the current σ observable following the intuition that if changes in fitness are observed then changes in the parameter value should be output, thus the old value must be available. This yields four sets of observables: $\{f_N\}$, $\{f_N, PDI\}$, $\{\Delta f, \sigma\}$ and $\{\Delta f, PDI, \sigma\}$.

4.4 Control Method

As a control method we chose a neural network (NN) as a generic method for mapping real valued inputs to real valued outputs. We use a simple feed-forward network without a hidden layer. The structure of the nodes is fixed and the weights remain static during an EA run and are set by the off-line tuning process. All inputs are, by definition, in the range $[0, 1]$. The weights are tuned $w \in [-1, 1]$. We tested three different activation functions (Table 1) with the output $o \in [0, 1]$. We also tested limiting the range of σ by multiplying the output with 0.1 (testing traditional practise of keeping $\sigma < 0.1$).

All six activation approaches were combined with all four observable sets. All these combinations were tested against six standard test problems[1], namely: Sphere, Corridor, Rosenbrock, Ackley, Rastrigin and Fletcher & Powell . This setup was repeated for two generation gaps: $\lambda = 70$ as a standard in numeric optimization practise and $\lambda = 1$ motivated by robotics applications [7]. Therefore, the total number of control-problem instances is 288, see Table 1.

In all cases, the search for good controllers (NN weights) was performed using BONESA [14], which is a state-of-the-art parameter tuning method for tuning real valued parameters. After each tuning session, the EA was ran 100 times using the best found controller instance, to validate its performance. These are then compared to outcomes of 100 runs with:

- a static σ, that is also tuned using BONESA, using the same computational budget as for finding the controller instance
- the Rechenberg's 1/5 rule applied to a global σ using the success ratio of all mutations in the population
- a self-adaptation approach using a single σ
- the theoretical optimum derived by Rechenberg [12] (applicable only to the Sphere and Corridor functions)

Table 1. Experimental Setup

λ	$1, 70$
Observables	$\{f_N\}$, $\{f_N, PDI\}$, $\{\Delta f, \sigma\}$, $\{\Delta f$, PDI, $\sigma\}$
Activations	$a_1(x) = x$ $a_2(x) = 1/(1 + e^{-6x})$ $a_3(x) = \tanh 3x$
Problems	Sphere, Corridor, Ackley, Rosenbrock, Rastrigin, Fletcher& Powell
EA	$(10 + \lambda)$-ES with: Gaussian mutation, no recombination and uniform random parent selection, limit of 10000 evaluations
Instantiation	BONESA with a budget of 3000 tests to calibrate weights $w_i \in [-1, 1]$

5 Results

The performance results of the calibrated control mechanisms and the benchmarks are presented in Table 2.

First, we consider the performance of control over the test problems by examining the results column-wise. We can observe that the calibrated control mechanisms are able to perform well (or comparably to the benchmarks) on all problems. Comparing our control to the tuned static approach, we find that on all problems except F&P, there are multiple control settings that are significantly better. For the F&P function there are control settings with no significant difference in average. Compared to self-adaptation, our control mechanism is overwhelmingly better on the Corridor, Ackley and Rastrigin problems while, for the other test functions, there are several settings that result in a tie. Control is always able to outperform the theoretical optimum (notice that this optimum was derived for a $(1 + 1)$ES).

Second, we consider control settings separately (by examining the results row-wise) to determine what observables and activation functions are better, and if there is a combination that consistently performs well. From this perspective, using a hypertangent activation is always the best option: performance is always significantly better than static for most problems, while it is significantly better than self-adaptation in three out of six problems and at least as good in four. The best choice of observables is either $\{f_N\}$ or $\{\Delta f, \sigma\}$.

Though choosing between f_N or Δf is mostly a matter of feasibility (calculating normalized fitness is not possible if the bounds of the fitness values are not known beforehand), including a diversity observable or not, is a more fundamental question. Contrary to our initial expectations, adding diversity to the input does not always offer an advantage even for multimodal problems (including the irregular and asymmetric Fletcher&Powell function). Keeping all other factors fixed, using diversity as an input produces a significant improvement in 23.6% of all cases and only in 8.3% of the cases that are using the hypertangent activation. This advantage of using diversity combined with the hypertangent activation is only observed for the Rosenbrock problem that is asymmetric but still unimodal. Our assumption is that the ineffectiveness of observing diversity is due to the survivor selection mechanism. Diversity could be useful as feedback if σ control aimed at maintaining a diverse population. However, $(\mu + \lambda)$ selection negates such an effort since any new "explorative" individuals will have inferior fitness and will be immedeately discarded. Thus, including diversity in the observables when a strong selection pressure is used is not useful (or even deleterious if we consider the increase in the weights space).

Table 2. Performances results. For each problem, we mark the control mechanisms that are significantly better than static (underlined), significantly better than self-adaptation (bold) and not significantly different than self-adaptation (italic). All problems are to be minimized.

	$\lambda = 1$						$\lambda = 70$					
	Sphere	Corridor	Rsnbk	Ackley	Rsgn	FletchP	Sphere	Corridor	Rsnbk	Ackley	Rsgn	FletchP
$\{f_N$,PDI$\}$												
lin	0.01504	**3.174**	*7.906*	**1.639**	**35.35**	*9355*	0.01525	**3.175**	*7.898*	**1.004**	**36.36**	*9141*
lin.1	0.0326	*9.124*	**5.001**	*14.25*	*61.08*	*1.008e+04*	0.04045	*9.133*	11.62	*16.18*	*57.73*	1.251e+04
sig	0.0963	**1.694**	*6.626*	**2.081**	**34.73**	*9880*	0.1009	**1.696**	*7.219*	**2.104**	**35.28**	1.09e+04
sig.1	0.0126	*9.124*	*5.969*	**5.634**	*61.41*	*8969*	0.04751	*8.938*	*6.666*	**7.454**	*59.9*	*1.002e+04*
tanh	0.03605	**1.682**	**4.709**	**0.07069**	**35.79**	*8618*	0.04408	**1.684**	*7.129*	**0.3**	**35.09**	*1.013e+04*
tanh.1	0.07144	*9.124*	**5.56**	**4.755**	*57.09*	*8538*	0.07844	*8.938*	*6.698*	**7.758**	*57.16*	1.054e+04
$\{f_N\}$												
lin	9.704e-07	**1.514**	57.22	**9.114e-07**	**46.62**	*8481*	9.003e-06	**1.638**	181.8	**8.978e-07**	**49.22**	1.589e+04
lin.1	0.01152	*9.22*	371.5	**7.964**	*60.97*	2.704e+04	1.438	*9.135*	869.5	**11.7**	*59.86*	7.54e+04
sig	0.538	**1.593**	11.18	**4.908**	**36.21**	*7669*	0.472	**1.701**	11.45	**13.52**	**36.25**	*6675*
sig.1	0.05019	*9.124*	**5.27**	**9.725**	*60.87*	*1.016e+04*	0.05714	*8.938*	*7.228*	**7.091**	*58.87*	1.087e+04
tanh	*9.223e-07*	**1.512**	18.75	**9.156e-07**	**20.93**	*7714*	*8.889e-07*	**1.602**	74.1	**8.789e-07**	**23.64**	*1.026e+04*
tanh.1	9.903e-07	*9.13*	150.9	**2.573**	*57.89*	1.572e+04	0.09544	*8.864*	469.4	**4.896**	*59.55*	3.421e+04
$\{\Delta f$, PDI, $\sigma\}$												
lin	0.003837	**6.519**	*6.283*	**2.608**	**35.2**	*7992*	0.02674	**6.519**	*8.136*	**3.467**	**46.99**	*8631*
lin.1	0.02484	*9.124*	**5.382**	18.48	*61.07*	*9401*	0.04189	*9.217*	19.49	18.43	*58.27*	1.276e+04
sig	0.1092	**1.787**	*7.007*	**7.269**	**35.46**	*9850*	0.1147	**1.792**	*7.022*	**5.597**	**36.75**	*8892*
sig.1	0.01506	*9.124*	*6.031*	**8.065**	*61.02*	*1.104e+04*	0.03794	*9.218*	*6.706*	**10.7**	*58.4*	*9969*
tanh	0.007184	**2.147**	*6.834*	**0.4822**	**34.35**	*6159*	0.003973	**2.15**	*7.499*	**0.8447**	**38.72**	*8217*
tanh.1	0.009213	*9.124*	**5.645**	**9.314**	*58.3*	*8622*	0.06944	*8.938*	*6.86*	*14.35*	*60.08*	*1.008e+04*
$\{\Delta f, \sigma\}$												
lin	0.0027	**6.24**	**5.379**	**1.4**	**49.95**	*8757*	0.002984	**6.24**	*8.11*	**7.87**	**36**	*7489*
lin.1	0.01787	*9.129*	**5.369**	18.51	*60.48*	*1.081e+04*	0.04558	*9.22*	12.19	18.45	*58.82*	1.182e+04
sig	0.06929	**1.79**	*7.485*	**6.75**	**32.44**	*9914*	0.06655	**1.792**	*7.895*	**8.002**	**37.46**	*8917*
sig.1	0.01255	*8.938*	*5.966*	**6.929**	*61.2*	*9806*	0.04192	*9.217*	*6.651*	**11.44**	*59.57*	*1.032e+04*
tanh	0.002631	**1.963**	**5.715**	**0.762**	**32.2**	*6467*	0.0008254	**1.964**	*6.282*	**0.9197**	**36.67**	*6759*
tanh.1	0.002881	*8.938*	**5.726**	**8.473**	*57.99*	*8137*	0.05975	*8.938*	*6.8*	**11.41**	*59.08*	*1.006e+04*
static	0.01721	**4.659**	6.721	**9.383**	**39.31**	*1.073e+04*	0.06903	**3.378**	7.254	**9.234**	**37.07**	*9313*
15rule	6.414	*9.221*	1212	18.44	85.84	1.85e+05	0.5263	*9.124*	34.34	18.28	*56.29*	1.828e+04
sssa	9.256e-07	8.938	7.203	15.04	57.65	8740	8.887e-07	8.938	7.458	15.65	55.34	6781
rechopt	0.002939	**4.659**	21.49	**7.887**	**39.2**	8712	1.027	**4.659**	21.73	**7.863**	**39.2**	*8631*

6 Conclusions and Future Work

Based on our results, the questions stated in the introduction can be easily answered. The main conclusion that can be drawn is that the generic approach taken in this paper is viable and fruitful. In contrast to previous work, we were able to find a problem-tailored control mechanism that outperforms the tuned (but static) parameter values for each of the problems. More specific, the combination of $\{f_N\}$, or $\{\Delta f, \sigma\}$, as observables and a hypertangent activation function of the neural network, outperforms the tuned parameter values in most problems. Even more promising is the fact that this combination performed equally well or better than the traditional methods for controlling a single σ, such as the 1/5th rule and self-adaptation. However, in contrast to these methods, our generic approach has the possibility to be extended to other parameters, possibly leading to an increased performance. Although this comes with the added cost of specific problem-tailoring, the benefits, especially in case of repetitive problems, can be significant.

Inevitably, this conclusion depends on the experimenters design decisions. In our case, the most important factors (beyond the underlying algorithm itself) are the following:

- The observables chosen as inputs to the control strategy.
- The parameters to be controlled
- The technique that maps a vector of observable values to a vector of parameter values

Changing either of these can, in principle, lead to a different conclusion. With respect to the observables chosen, we can conclude that these indeed highly influence the results. Remarkably, adding diversity as an input appears to have hardly any added value for controlling σ, possibly due to the selection mechanism used. The normalized fitness and $\{\Delta f, \sigma\}$ appear to be the best choices for input. Note that calculating the normalized fitness is not always possible, and is most probably the less robust choice.

Regarding the future, we expect more studies along these ideas, exploring the other possible implementations of the most important factors. Most enticing is the possibility of applying it to the other parameters of the evolutionary algorithm. This has the prospect of delivering high quality control methods that can adapt the underlying algorithm to the different stages of the search process.

References

1. T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, Oxford, UK, 1996.
2. K. De Jong. Parameter setting in EAs: a 30 year perspective. In F. Lobo, C. Lima, and Z. Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, pages 1–18. Springer, 2007.
3. G. di Tollo, F. Lardeux, J. Maturana, and F. Saubion. From adaptive to more dynamic control in evolutionary algorithms. In *Proceedings of the 11th European conference on Evolutionary computation in combinatorial optimization*, EvoCOP'11, pages 130–141. Springer-Verlag, 2011.

4. A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter Control in Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.
5. A. E. Eiben, Z. Michalewicz, M. Schoenauer, and J. E. Smith. Parameter control in evolutionary algorithms. In *Parameter Setting in Evolutionary Algorithms*, pages 19–46. 2007.
6. T. C. Fogarty. Varying the probability of mutation in the genetic algorithm. In *Proceedings of the third international conference on Genetic algorithms*, pages 104–109, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.
7. G. Karafotias, E. Haasdijk, and A. E. Eiben. An algorithm for distributed on-line, on-board evolutionary robotics. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 171–178. ACM, 2011.
8. M. A. Lee and H. Takagi. Dynamic control of genetic algorithms using fuzzy logic techniques. In *PROCEEDINGS OF THE FIFTH INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS*, pages 76–83. Morgan Kaufmann, 1993.
9. M. Majig and M. Fukushima. Adaptive fitness function for evolutionary algorithm and its applications. *Informatics Research for Development of Knowledge Society Infrastructure, International Conference on*, pages 119–124, 2008.
10. J. Maturana and F. Saubion. On the design of adaptive control strategies for evolutionary algorithms. In *Artificial Evolution*, volume 4926 of *Lecture Notes in Computer Science*, pages 303–315. Springer, 2008.
11. V. Nannen, S. Smit, and A. Eiben. Costs and benefits of tuning parameters of evolutionary algorithms. In G. R. et al, editor, *Proceedings of the 10th international conference on Parallel Problem Solving from Nature (PPSN X)*, volume 5199 of *Lecture Notes in Computer Science*, pages 528–538. Springer, 2008.
12. I. Rechenberg. *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution*. Fromman-Hozlboog Verlag, Stuttgart, 1973.
13. N. N. Schraudolph and R. K. Belew. Dynamic parameter encoding for genetic algorithms. *Machine Learning*, 9:9–21, 1992.
14. S. Smit and A. Eiben. Multi-problem parameter tuning using bonesa. In J. Hao, P. Legrand, P. Collet, N. Monmarché, E. Lutton, and M. Schoenauer, editors, *Artificial Evolution*, pages 222–233, 2011.
15. S. K. Smit, Z. Szláavik, and Á. E. Eiben. Population diversity index: a new measure for population diversity. In *GECCO (Companion)*, pages 269–270, 2011.
16. R. Smith and E. Smuda. Adaptively resizing populations: Algorithm, analysis and first results. *Complex Systems*, 9(1):47–72, 1995.
17. W. M. Spears. Adapting crossover in evolutionary algorithms. In *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 367–384. MIT Press, 1995.
18. P. Vajda, A. E. Eiben, and W. Hordijk. Parameter Control Methods for Selection Operators in Genetic Algorithms. pages 620–630.
19. Y.-Y. Wong, K.-H. Lee, K.-S. Leung, and C.-W. Ho. A novel approach in parameter adaptation and diversity maintenance for genetic algorithms. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 7:506–515, 2003.