TUNING EVOLUTIONARY ALGORITHMS
AND TUNING EVOLUTIONARY ALGORITHM CONTROLLERS

Giorgos Karafotias
VU University Amsterdam
g.karafotias@vu.nl

Selmar Kagiso Smit
VU University Amsterdam
selmar@gmail.com

Agoston Endre Eiben
VU University Amsterdam
gusz@cs.vu.nl

Abstract In the last few years various powerful parameter tuning methods have been presented for evolutionary algorithms. These can nicely handle the tuning problem, delivering good parameter values that remain constant during an EA run. In this paper we conduct a feasibility study into using one of these tuning methods to solve the control problem. The idea is to specify a parameterized mechanism to control EA parameters on-the-fly and optimize the parameters of the control mechanism with our tuning algorithm. We perform proof-of-concept experiments with 3 parameters of an evolution strategy on 5 different objective functions and find that on all problems the EA using the tuned controller is better or at least as good as the EA whose parameters were tuned directly.

Keywords: Evolutionary algorithms, parameter tuning, parameter control

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 [5, 8]. 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 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. 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. The combinations in the top row have the advantage of enhanced performance at the cost of the tuning effort [17]. The options in the left column offer the benefits of time varying parameter values mentioned above with a trade-off of increased complexity.

Here, we focus on the top left cell of the matrix, i.e. control that is tailored to a specific problem. This approach combines on-line parameter adjustment (control) and off-line configuration (tuning). The EA incorporates a parameter control mechanism, but this controller itself has certain parameters that can be configured for a problem through an off-line tuning process. This tunable control mechanism is proposed as an alternative to tuning static parameter values: instead of spending time to search for good EA parameter values, the same effort can be spent to make a good calibration of a mechanism that performs on-line control of these parameters. Such a method has the advantage of enhanced performance thanks to varying parameter values, and the possibility to be tuned to a specific problem. The objective of this paper is to introduce such a generic mechanism and to answer the following questions:
Is such an approach viable?

What is the added value of this approach, when compared to static parameter values?

On what kind of feedback from the search process should such a parameter control mechanism base its decisions?

To answer these questions we introduce a generic framework that helps understand the decisions behind the design of a control mechanism and conduct experiments with three parameters of an evolution strategy.

2. Related work

Parameter control is an increasingly popular topic in the field of evolutionary algorithms [14]. 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 focus on mutation probability [9], mutation step size [21] and operator selection [26] but methods also exist for the selection pressure [27], the population-size [25], the fitness function [15] and the encoding [22].

The mutation step size of evolution strategies was one of the first parameters that was considered for control. In the field of Evolution Strategies, controlling the mutation step size was one of the key ingredients to its success. Analysis of the simple corridor and sphere problems in large dimensions in the early 70’s led to Rechenberg’s 1/5th success rule that changes the mutation step size, based on the feedback about its success. Not much later, self-adaptation of sigma was introduced.

Self-adaptive control is not based on such deterministic rules to change parameter values. Instead, it encodes the $\sigma$ into the chromosomes and co-evolves them with the problem parameters. In general, this is much less interpretable since the immediate effect on the parameter-values is not very clear. Furthermore, because in each run these values are ‘re-learned’, the information gained about the appropriate parameter values for different situations, cannot be reused in another run or another EA.

The control of the population size of evolutionary algorithms has been previously examined in several works. One approach taken to the population size parameter was to eliminate it all together by introducing the notion of individuals’ lifetimes as was done in GAVaPS [1]. Another approach was to approximate a good population size on-line: the parameter-less GA [11] automatically performs an on-line search for a good population size by creating and running populations of progres-
sively larger size. Direct control of the population size parameter has also been studied deterministically [12], adaptively [6] and self-adaptively [4].

Some generic control methods for numeric parameters also exist. In [28] 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 [13] 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 generic numeric control is presented in [3] and [16]: 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 [16] 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. There are a few studies that compare control strategies with tuned parameter values [19, 18, 10, 20], however these typically arrive to a conclusion opposite to that of [13, 3, 16].

Extensive literature reviews on parameter control can be found in [7] and [2].

3. Parameter Control Road-map

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 road-map 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 A 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:

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

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

$$\text{Source} \rightarrow (\text{Digest}) \rightarrow (\text{Derivative}) \rightarrow (\text{History})$$

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 $F_H(v_1, v_2, ..., v_W)$ 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 Mapping Technique

Any technique that translates a vector of observable values to a vector of parameter values can be used as an mapping for the control mechanism, e.g., a rule set, an ANN or a function are all valid candidates. The choice of the proper technique 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.

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:

$$\vec{p} = c(\vec{o}) \text{ and } \vec{o}_1 = \vec{o}_2 \Rightarrow c(\vec{o}_1) = c(\vec{o}_2)$$

where $\vec{o} \in O$, $\vec{p} \in P$ 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:

$$\vec{p} = c_p(\vec{o}, S_C) \text{ and } S_{C}^{t+1} = c_S(\vec{o}_t, S_C^t)$$

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

The motivation of this paper is to provide a proof of the concept that a tunable and generic controller can have an advantage over tuning static parameter values. For this purpose we need to compare the performance of the two approaches. To facilitate this comparison, though,
we need concrete representatives of the two methods. For the case of tuned static parameters this is trivial, however, we require a tunable generic controller as a representative of tunable control. We describe such a controller in this section; notice that the described controller is merely one possible implementation used for the experiments and we do not suggest it as an ‘optimal’ controller design.

The controller is required to be generic enough to be able to handle any numeric parameter, capable of problem-specific calibration and compatible with any evolutionary algorithm. Below we present a design for an EA-independent, parameter-independent and tunable controller based on the framework presented in Section 3.

4.1 Parameters

Any numeric parameter can be controlled and the value of a controlled parameter is updated in each generation. To control a parameter, a range $[\text{min}, \text{max}]$ of values must be specified which will be the output range of the controller.

4.2 Observables

The observables that we currently use as input to the controller, are based on the current parameter values, diversity and fitness. However, in principle any observable that provides useful information about the current state of the algorithm can be used.

**Fitness based.** We use two observables based on fitness. Both use the best fitness digest of the current population: $f_B = \max \mathcal{F}(g), g \in G$. The simpler fitness-based observable $f_N$ is just the normalized value of the best fitness found in the current population: $f_N = \text{norm}(f_B), f_N \in [0, 1]$. This observable is only applicable when the bounds of the fitness function are known. The second fitness-based observable $\Delta f$ provides information about the change of the best fitness observed in the last generations. Instead of using the derivative step to describe this change we use a history of 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 history setting to measure change instead of the derivative step so to make the controller robust to shifting and stretching of the fitness landscape.

To summarize the two fitness-based observables are:

$$f_N = \text{norm}(f_B), f_N \in [0, 1]$$
\[ \Delta f = \frac{f_W^2 - f_W^W/2}{f_W^W - f_1^W}, \Delta f \in [0, 1], W = 100 \]

where \( f_B = \max F(g), g \in G \). In order to be able to use \( \Delta f \) from the beginning, the value of \( W \) grows from 2 to 100 with each generation. Notice that these observables are not used together but are two alternatives.

**Diversity based.** Diversity is observed using the Population Diversity Index (PDI) [24] as the digest function and bypassing derivatives and history. The Population Diversity Index is a measure that uses an entropy like approach for measuring the genotypic diversity in a real-valued population. It can identify clusters, and always returns a value between 0 and 1 indicating a fully converged (0), a uniformly distributed (1) population, or anything in between.

**Current parameter values.** The current values of the controlled parameters \( \vec{p}_c \) are also observed and input to the controller when the \( \Delta f \) observable is used. The reason is that if changes in fitness are observed then changes in the parameter value should be output, thus the old value must be available. Each value in \( \vec{p}_c \) corresponds to the current value of a controlled parameter and is normalized to \([0, 1]\) (this is possible because the range of all controlled parameters is known as described in 4.1).

Given two choices for observing fitness and the choice to include the diversity observable or not yields four sets of observables: \( \{f_N\} \), \( \{f_N, PDI\} \), \( \{\Delta f, \vec{p}\} \) and \( \{\Delta f, PDI, \vec{p}\} \).

### 4.3 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. Thus, the calibration of the controller consists of finding appropriate weights for a given problem or application type.

All inputs are, as defined above, in the range \([0, 1]\). The weights are tuned \( w \in [-1, 1] \). The activation of the neurons is a sigmoid function, horizontally compressed so as to reach very close to its asymptotes given a domain of \([0, 1]\).

When multiple parameters are controlled simultaneously a separate NN controls each parameter. If the current parameter values \( \vec{p}_c \) are used as input, then each NN uses only the value of the corresponding parameter as input. This may seem oversimplifying considering parameter in-
teraction but it also simplifies the controller and significantly decreases the number of weights that need to be calibrated.

5. Experimental Setup

We conducted experiments to evaluate the tunable control approach (represented by the generic controller described in the previous section) and how it compares with the tuned static approach. Using the same EA, solving the same problems and spending the same effort for off-line tuning/calibration, we compared the performance achieved when keeping parameter values static to the performance achieved when using the controller.

The evolutionary algorithm used, the relevant parameters, the method used for tuning and the overall experimental setup are described in the following subsections.

5.1 Evolutionary algorithm and parameters

The EA used, is a simple \((\mu + \lambda)\) ES with Gaussian mutation. It has no recombination and uses uniform random parent selection. This EA has three parameters: the population size \(\mu\), the generation gap \(g = \frac{\lambda}{\mu}\) and the mutation step size \(\sigma\). We run five distinct experiment sets controlling each parameter alone, controlling \(\mu\) and \(g\) together (considering their obvious interaction) and controlling all parameters together. In the experiments where one of the parameters is not controlled/tuned then it is always set to a value chosen based on conventional EC knowledge and intuition (see Table 1). These values also serve as initial values when a parameter is controlled.

\[
\begin{array}{c|l}
\mu & 10 \\
g & 7 \\
\sigma & (\text{Sphere}) 0.1, (\text{Corridor}) 0.1, (\text{Ackley}) 0.6, (\text{Rosenbrock}) 0.04, (\text{Fletcher&Powell}) 0.07, (\text{Rastrigin}) 0.1 \\
\end{array}
\]

5.2 Off-line tuning

Both controller calibrations and static values are found through an off-line tuning process using Bonesa [23]. Bonesa is an iterative model-based search procedure based on an intertwined searching and learning loop. The search loop is a generate-and-test procedure that iteratively generates new vectors, pre-assesses their quality using a surrogate model
of the performance landscape, and finally tests the most promising vectors by executing an algorithm run with these specific values. In its turn, the learning loop uses the information obtained about the quality of the tested vectors to update the model of the performance surface. Furthermore, it uses a kernel filter to reduce the noise caused by the stochasticity of the algorithm.

5.3 Overall Setup

For each problem, the four versions of the controller (using the four different observables sets described in 4.2) are calibrated using Bonesa. The same tuner, and effort is also spent on finding good static values for the problem, which adheres to classical parameter tuning. The resulting EAs (with controlled and static parameters) are run 100 times to validate the outcomes, to fairly compare their performances. The same experiment is repeated for controlling each of the parameters individually, $\mu$ together with $g$ and all together. The experimental setup is summarized in Table 2.

<table>
<thead>
<tr>
<th>Table 2. Experimental Setup</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EA</strong></td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
</tr>
<tr>
<td><strong>Instantiation</strong></td>
</tr>
<tr>
<td><strong>Problems</strong></td>
</tr>
<tr>
<td><strong>Controller observables</strong></td>
</tr>
</tbody>
</table>

6. Results

Results are shown in Tables 3 to 7. For all tables, bold values denote a performance that is not significantly worse than the best on that specific problem, while underlined values denote performance that is significantly better than the EA with static parameter values for the specific problem. In all cases, significance is verified using a t-test with 95% confidence. Tuning and solving for the Corridor problem failed in all cases, so it will be completely disregarded in the subsequent analysis.
6.1 Performance

Tables 3, 4 and 6 show that attempting to control solely the population size, the generation gap or their combination was unsuccessful. But, controlling only $\sigma$ (Table 5) was successful for the controlled EA. It outperforms the static EA on three out of five problems, while not being significantly worse in the other two.

However, controlling all parameters (Table 7) at the same time, seems to be the most beneficial; the controlled EA outperforms the static in four problems while it is not significantly worse in the fifth. Even better (although not shown here), on most of the problems, the controller using all parameters outperform those for a single parameter or $\mu, g$ combination.

In general, for the EA and problems tested, the controller performs better or at least as good as the tuned, but static parameter values.

6.2 Parameter behavior

Analyzing the behavior of the controlled parameters provides some important insight.

**Controlling $\mu$ and $g$:** As the performance results show (Tables 3, 4 and 6) there is no improvement when controlling $\mu$ and $g$. In fact, in most cases, the calibration of control creates controllers that maintain constant parameter values. When controlling $\mu$ alone, controllers almost always maintain a constant value, either set to 1 (Sphere, Ackley and Rosenbrock) or simply linearly increasing to its maximum (Rastrigin). Similarly, when controlling $g$ alone, in almost all cases values are kept constant either to its minimum so that each generation has only one offspring (Sphere, Ackley, Rosenbrock) or its maximum (Fletcher & Powell, Rastrigin). In the few cases where parameter values vary according to the inputs there is no improvement in performance. These findings could mean that controlling these parameters for the specific EA and problems does not have an intrinsic value or that it is very difficult for the tuning process to find good controller weights.
Controlling $\sigma$: A much more interesting behavior is observed in the values of $\sigma$. In most cases, $\sigma$ shows an increase in the beginning and, subsequently, drops and stabilizes or oscillates around a value. Such a gradual decrease is a good strategy for climbing a hill and approximating an optimum. An example of this $\sigma$ behavior is shown in Figure 2. Of course, a preferable behavior would be to re-increase $\sigma$ when a population is stuck to a local optimum. Such a situation can be detected based on the diversity and $\Delta f$ observables used, however, this desired behavior does not occur in our results. A different control strategy is observed in Figure 3: $\sigma$ oscillates rapidly while the range of this oscillation seems related to current diversity. Despite its apparent oddity, this controller performs better than tuned static.

Controlling all combined: Combined control of all parameters produces the best results with calibrated controllers generally demonstrating a more complex behavior. Values of $\mu$ and $g$ still often remain constant, especially for the Rosenbrock and Sphere problems (this is not necessarily a drawback considering that these two problems are unimodal and can be solved with a $(1+1)$ES). Control of $\sigma$ shows a behavior similar to what is described in the previous paragraph. Examples of parameters’ behavior are illustrated in Figures 4 and 5. In one case $\mu$ is set to a constant value, however, $g$ and $\sigma$ are controlled according to the inputs. In the first case, $\sigma$ shows one of the two behaviors described in the previous paragraph while in the second it simply follows diversity.

6.3 Observables

The best results are acquired using only fitness based observables, either $\{f_N\}$ or $\{\Delta f, \bar{p}\}$. Between these two we cannot distinguish a clear winner.

Though choosing between $f_N$ or $\Delta f$ 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 yield better performance.
even for multimodal problems (including the irregular and asymmetric Fletcher & Powell function). In fact, keeping all other factors fixed, using diversity as an input produces a significant improvement only in 2 out of 50 cases.

6.4 Discussion on Selection

Three important points were observed in the results and analysis of this section:

- the failure to calibrate a meaningful controller for \( \mu \),
- the absence of control strategies that increase \( \sigma \) late in the run to escape local optima,
- and the fact that using diversity as an observable does not improve performance even for multimodal problems.

A plausible assumption is that these points might be due to the survivor selection used by the ES in the experiments (“plus” selection). All above points are related to maintaining a diverse population, either by accommodating enough individuals, by performing enough exploration when
Table 3. Performance results for $\mu$. For each column, bold values denote performance not significantly different to the best and underlined values denote performance significantly better than static.

<table>
<thead>
<tr>
<th></th>
<th>Sph</th>
<th>Crdr</th>
<th>Rsbk</th>
<th>Ack</th>
<th>Reg</th>
<th>F&amp;P</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(f_N, D)$</td>
<td>0.09607</td>
<td>9.303</td>
<td>4.868</td>
<td>5.75</td>
<td>36.98</td>
<td>7602</td>
</tr>
<tr>
<td>$(f_N)$</td>
<td>0.09607</td>
<td>9.303</td>
<td>4.868</td>
<td>5.75</td>
<td>36.77</td>
<td>6731</td>
</tr>
<tr>
<td>$(\Delta f, D, \vec{p})$</td>
<td>0.09607</td>
<td>9.303</td>
<td>4.868</td>
<td>5.75</td>
<td>38.97</td>
<td>5528</td>
</tr>
<tr>
<td>Static</td>
<td>0.09901</td>
<td>9.303</td>
<td>4.329</td>
<td>5.776</td>
<td>34.29</td>
<td>1335</td>
</tr>
</tbody>
</table>

Table 4. Performance results for $g$. For each column, bold values denote performance not significantly different to the best and underlined values denote performance significantly better than static.

<table>
<thead>
<tr>
<th></th>
<th>Sph</th>
<th>Crdr</th>
<th>Rsbk</th>
<th>Ack</th>
<th>Reg</th>
<th>F&amp;P</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(f_N, D)$</td>
<td>0.1009</td>
<td>9.303</td>
<td>5.115</td>
<td>5.76</td>
<td>55.15</td>
<td>1.113e+04</td>
</tr>
<tr>
<td>$(f_N)$</td>
<td>0.1009</td>
<td>9.303</td>
<td>5.115</td>
<td>5.76</td>
<td>55.15</td>
<td>1.065e+04</td>
</tr>
<tr>
<td>$(\Delta f, D, \vec{p})$</td>
<td>0.1009</td>
<td>9.303</td>
<td>6.142</td>
<td>5.752</td>
<td>54.4</td>
<td>1.065e+04</td>
</tr>
<tr>
<td>Static</td>
<td>0.1003</td>
<td>9.303</td>
<td>5.226</td>
<td>5.778</td>
<td>79.35</td>
<td>5770</td>
</tr>
</tbody>
</table>

Table 5. Performance results for $\sigma$. For each column, bold values denote performance not significantly different to the best and underlined values denote performance significantly better than static.

<table>
<thead>
<tr>
<th></th>
<th>Sph</th>
<th>Crdr</th>
<th>Rsbk</th>
<th>Ack</th>
<th>Reg</th>
<th>F&amp;P</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(f_N, D)$</td>
<td>0.04848</td>
<td>9.303</td>
<td>7.473</td>
<td>0.26</td>
<td>29.06</td>
<td>6848</td>
</tr>
<tr>
<td>$(f_N)$</td>
<td>0.01609</td>
<td>9.303</td>
<td>6.995</td>
<td>0.1085</td>
<td>25.32</td>
<td>5622</td>
</tr>
<tr>
<td>$(\Delta f, D, \vec{p})$</td>
<td>0.0528</td>
<td>9.303</td>
<td>8.05</td>
<td>0.3153</td>
<td>27.08</td>
<td>6089</td>
</tr>
<tr>
<td>Static</td>
<td>0.05745</td>
<td>9.303</td>
<td>8.111</td>
<td>0.617</td>
<td>25.85</td>
<td>8238</td>
</tr>
</tbody>
</table>

necessary or by screening the current status of diversity. However, using a “plus” selection could negate an effort to increase diversity since new and “diverse” individuals would be of inferior fitness and, thus, discarded while newly grown populations would be taken over by the existing best individuals.

7. 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
Table 6. Performance results for \( \mu.g \). For each column, bold values denote performance not significantly different to the best and underlined values denote performance significantly better than static.

<table>
<thead>
<tr>
<th></th>
<th>Sph</th>
<th>Cdr</th>
<th>Rsbk</th>
<th>Ack</th>
<th>Reg</th>
<th>F&amp;P</th>
</tr>
</thead>
<tbody>
<tr>
<td>((f_N, D))</td>
<td>0.09869</td>
<td>9.303</td>
<td>6.89</td>
<td>5.633</td>
<td>38.26</td>
<td>5841</td>
</tr>
<tr>
<td>((f_N))</td>
<td>0.09382</td>
<td>9.303</td>
<td>6.89</td>
<td>5.591</td>
<td>38.66</td>
<td>6432</td>
</tr>
<tr>
<td>((\Delta f, D, \vec{p}))</td>
<td>0.09869</td>
<td>9.303</td>
<td>6.89</td>
<td>5.633</td>
<td>38.19</td>
<td>4299</td>
</tr>
<tr>
<td>((\Delta f, \vec{p}))</td>
<td>0.09382</td>
<td>9.303</td>
<td>6.89</td>
<td>5.756</td>
<td>36.22</td>
<td>3465</td>
</tr>
<tr>
<td>Static</td>
<td>0.09475</td>
<td>9.303</td>
<td>3.834</td>
<td>5.575</td>
<td>34.08</td>
<td>762.3</td>
</tr>
</tbody>
</table>

Table 7. Performance results for all. For each column, bold values denote performance not significantly different to the best and underlined values denote performance significantly better than static.

<table>
<thead>
<tr>
<th></th>
<th>Sph</th>
<th>Cdr</th>
<th>Rsbk</th>
<th>Ack</th>
<th>Reg</th>
<th>F&amp;P</th>
</tr>
</thead>
<tbody>
<tr>
<td>((f_N, D))</td>
<td>1.102</td>
<td>9.303</td>
<td>26.81</td>
<td>3.337</td>
<td>29.28</td>
<td>7824</td>
</tr>
<tr>
<td>((f_N))</td>
<td>0.007457</td>
<td>9.303</td>
<td>11.54</td>
<td><strong>0.5488</strong></td>
<td><strong>23.99</strong></td>
<td>5999</td>
</tr>
<tr>
<td>((\Delta f, D, \vec{p}))</td>
<td>5.387</td>
<td>9.303</td>
<td>82.6</td>
<td>18.5</td>
<td>36.52</td>
<td>2.055e+05</td>
</tr>
<tr>
<td>((\Delta f, \vec{p}))</td>
<td><strong>0.005169</strong></td>
<td>9.303</td>
<td>7.358</td>
<td>3.275</td>
<td>30.48</td>
<td><strong>2028</strong></td>
</tr>
<tr>
<td>Static</td>
<td>0.03565</td>
<td>9.303</td>
<td>7.025</td>
<td>2.024</td>
<td>35.9</td>
<td>2828</td>
</tr>
</tbody>
</table>

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, either using the best fitness value directly, or the combination of \( \delta \) fitness and the current parameter values, outperforms the tuned but static parameter values in most problems.

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 \( \sigma \).

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 other algorithms.
Tuning Evolutionary Algorithms and Tuning Evolutionary Algorithm Controllers

and applications. If these could be controlled with the same ease, this opens up the possibilities for a problem-tailed and high-performing algorithm for any particular problem.

Notes

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

References


