Swarm and Evolutionary Computation 1 (2011) 19–31

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

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

Invited paper

Parameter tuning for configuring and analyzing evolutionary algorithms

A.E. Eiben*, S.K. Smit[1]

Department of Computer Science, Vrije Universiteit Amsterdam De Boelelaan 1081a 1081 HV, Amsterdam, Netherlands

ARTICLE INFO

Article history:
Received 8 November 2010
Received in revised form
24 January 2011
Accepted 2 February 2011
Available online 3 March 2011

Keywords:
Evolutionary algorithms
Parameter tuning
Algorithm performance
Robustness
Experimental methodology

ABSTRACT

In this paper we present a conceptual framework for parameter tuning, provide a survey of tuning methods, and discuss related methodological issues. The framework is based on a three-tier hierarchy of a problem, an evolutionary algorithm (EA), and a tuner. Furthermore, we distinguish problem instances, parameters, and EA performance measures as major factors, and discuss how tuning can be directed to algorithm performance and/or robustness. For the survey part we establish different taxonomies to categorize tuning methods and review existing work. Finally, we elaborate on how tuning can improve methodology by facilitating well-funded experimental comparisons and algorithm analysis.

© 2011 Elsevier B.V. All rights reserved.

1. Introduction

The main objectives of this paper are threefold. We want to present a conceptual framework behind parameter tuning, provide a survey of relevant literature, and argue for a tuning-aware experimental methodology. The conceptual framework is comprised of the pivotal notions regarding parameter tuning, arranged and presented in a certain logical structure. It also embodies a vocabulary that can reduce ambiguity in discussions about parameter tuning. However, we are not aiming at mathematical rigor, as we are not to present formal definitions and theorems. Our treatment is primarily practical.

We consider the design, or configuration, of an evolutionary algorithm (EA) as a search problem in the space of its parameters and, consequently, we perceive a tuning method as a search algorithm in this space. We argue that parameter tuning can be considered from two different perspectives, that of

- configuring an evolutionary algorithm by choosing parameter values that optimize its performance, and
- analyzing an evolutionary algorithm by studying how its performance depends on its parameter values.

To this end, it is essential that a search algorithm generates much data while traversing the search space. In our case, these data concern a lot of parameter vectors and corresponding values of algorithm performance. If one is only interested in an optimal EA configuration then such data are not relevant—finding a good parameter vector is enough. However, if one is interested in gaining insights into the EA at hand, then these data are highly relevant for they reveal information about the evolutionary algorithm's robustness, distribution of solution quality, sensitivity etc. Adopting the terminology of Hooker [1], we refer to these options as competitive and scientific testing, and discuss how scientific testing is related to the notion of algorithm robustness. We also show that there are multiple definitions of robustness obtained through extending the above list by

- analyzing an evolutionary algorithm by studying how its performance depends on the problems it is solving, and
- analyzing an evolutionary algorithm by studying how its performance varies when executing independent repetitions of its run,

and then we discuss how these definitions are related to parameter tuning.

The rest of this paper is organized as follows. We begin with an introductory treatment of EAs and their parameters in Section 2. Then the general conceptual framework for parameter tuning is outlined in Section 3. It is summarized by Fig. 3, that shows that the solutions of a tuning problem depend on (1) the problem(s) to be solved, (2) the EA used, (3) the utility function that defines how we measure algorithm quality, and (4) the tuning method. Section 4 elaborates on the third component, the utility function, discussing various notions of algorithm performance and robustness, followed by a discussion of robustness and EA design in Section 5. The fourth component, the tuner, is the main

* Corresponding author. Tel.: +31 0 20 5987758; fax: +31 0 20 5987653.
E-mail address: gusz@cs.vu.nl (A.E. Eiben).
URLs: http://www.cs.vu.nl/~gusz (A.E. Eiben), http://www.cs.vu.nl/~sksmit (S.K. Smit).
1 Tel.: +31 20 59 87865; fax: +31 0 20 5987653.

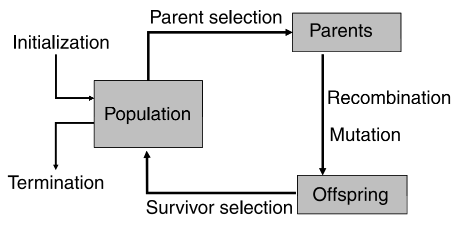

Fig. 1. General framework of an evolutionary algorithm.

subject of two sections. Section 6 describes three different ways to distinguish and classify tuning methods. Two of these focus on algorithmic details, the third one also takes the intended usage into account. The actual survey in Section 7 is structured by this latter one. Finally, Section 8 makes the link between parameter tuning and research methodology, giving recommendations towards a more tuning-aware practice.

Let us finish this introduction with a note on the scope of this paper. The title above indicates evolutionary algorithms as the subject matter of this paper. However, most of our considerations hold in the broader class of stochastic, heuristic search algorithms. In fact, several ideas behind our discussion are taken from or based on publications regarding heuristic methods in general, not restricted to evolutionary computing, e.g., [1,2]. In turn, many of the concepts introduced here for EAs can be directly transformed to various others methods, including swarm algorithms, particle-based optimizers, and the like.

2. Evolutionary algorithms and their parameters

Evolutionary algorithms form a class of heuristic search methods based on a particular algorithmic framework whose main components are the variation operators (mutation and recombination) and the selection operators (parent selection and survivor selection), cf. [3]. The general evolutionary algorithm framework is shown in Fig. 1.

A decision to use an evolutionary algorithm implies that the user adopts the main design decisions that led to the general evolutionary algorithm framework. That is, the decisions to use a population, manipulated by selection, recombination, and mutation operators follow automatically, the user only needs to specify "a few" details. In the sequel we use the term *parameters* to denote these details.

In these terms, designing an EA for a given application amounts to selecting good values for the parameters. For instance, the definition of an EA might include setting the parameter `crossoveroperator` to `onepoint`, the parameter `crossoverrate` to 0.5, and the parameter `populationsize` to 100. In principle, this is a sound naming convention, but intuitively there is a difference between choosing a good crossover operator from a given list of three operators and choosing a good value for the related crossover rate $p_c \in [0, 1]$. This difference can be formalized if we distinguish parameters by their domains. The parameter `crossoveroperator` has a finite domain with no sensible distance metric or ordering, e.g., {`onepoint`, `uniform`, `averaging`}, whereas the domain of the parameter p_c is a subset of $\Re$ with the natural structure for real numbers. This difference is essential for searchability. For parameters with a domain that has a distance metric, or is at least partially ordered, one can use heuristic search and optimization methods to find optimal values. For the first type of parameters this is not possible because the domain has no exploitable structure. The only option in this case is sampling.

The difference between two types of parameters has already been noted in evolutionary computing, but various authors use various naming conventions. For instance, [4] uses the names qualitative, and quantitative parameters, [5] distinguishes symbolic and numeric parameters, [6] calls them categorical and numerical, while [7] refers to structural and behavioral parameters. Furthermore, [8] calls unstructured parameters components and the elements of their domains operators. In the corresponding terminology, a parameter is instantiated by a value, while a component is instantiated by allocating an operator to it. In the context of statistics and data mining one distinguishes two types of variables (rather than parameters) depending on the presence of an ordered structure, but a universal terminology is lacking here too. Commonly used names are nominal vs. ordinal and categorical vs. ordered variables.

From now on we will use the terms *qualitative parameter* and *quantitative parameter*. For both types of parameters the elements of the parameter's domain are called *parameter values* and we *instantiate* a parameter by allocating a value to it. In practice, quantitative parameters are mostly numerical values, e.g., the parameter crossover rate uses values from the interval [0, 1], and qualitative parameters are often symbolic, e.g., `crossoveroperator`. In theory, a set of symbolic values can be ordered too, for instance, we could sort the set {`averaging`, `onepoint`, `uniform`} alphabetically. However, in practice it would not make much sense to make `crossoveroperator` a quantitative parameter by such a trick.

It is important to note that the number of parameters of EAs is not specified in general. Depending on particular design choices one might obtain different numbers of parameters. For instance, instantiating the qualitative parameter `parentselection` by `tournament` implies a new quantitative parameter `tournamentsize`. However, choosing for `roulettewheel` does not add any parameters. This example also shows that there can be a hierarchy among parameters. Namely, qualitative parameters may have quantitative parameters "under them". If an unambiguous treatment is required, then we can call such parameters *sub-parameters*, always belonging to a qualitative parameter.

2.1. EAs and EA instances

The distinction between qualitative and quantitative parameters naturally supports a distinction between EAs and EA instances. This view is based on considering qualitative parameters as high-level ones that define the main structure of an evolutionary algorithm, and look at quantitative parameters as low-level ones that define a specific variant of this EA. Following this naming convention an *evolutionary algorithm* is a partially specified algorithm, fitting the framework shown in Fig. 1, where the values to instantiate qualitative parameters are defined, but the quantitative parameters are not. Hence, we consider two EAs to be different if they differ in one of their qualitative parameters, for instance, use different mutation operators. If the values for all parameters are specified then we obtain an *evolutionary algorithm instance*. Table 1 illustrates this matter by showing three EA instances belonging to just two EAs.

This terminology enables precise formulations (which we will need in Section 4), meanwhile it enforces care with phrasing. Observe, that the distinction between EAs and EA instances is similar to distinguishing problems and problem instances. If rigorous terminology is required then the right phrasing is "to apply an EA instance to a problem instance". However, such rigor is not always needed, and formally inaccurate but understandable phrases like "to apply an EA to a problem" are acceptable if they cannot lead to confusion.

Table 1
Three EA instances specified by the qualitative parameters representation, recombination, mutation, parent selection, survivor selection, and the quantitative parameters mutation rate (p_m), mutation step size (σ), crossover rate (p_c), population size (μ), offspring size (λ), and tournament size (κ). In our terminology, the instances in columns A_1 and A_2 are just variants of the same EA. The EA instance in column A_3 belongs to a different EA, because it is different in its qualitative parameters.

	A_1	A_2	A_3
Qualitative parameters			
Representation	Bitstring	Bitstring	Real-valued
Recombination	1-point	1-point	Averaging
Mutation	Bit-flip	Bit-flip	Gaussian $N(0, \sigma)$
Parent selection	Tournament	Tournament	Uniform random
Survivor selection	Generational	Generational	(μ, λ)
Quantitative parameters			
p_m	0.01	0.1	0.05
σ	n.a.	n.a.	0.1
p_c	0.5	0.7	0.7
μ	100	100	10
λ	n.a.	n.a.	70
κ	2	4	n.a.

3. Tuning evolutionary algorithms

In the broad sense, algorithm design includes all decisions needed to specify an algorithm (instance) for solving a given problem (instance). Throughout this paper we perceive parameter tuning as a special case of algorithm design. The principal challenge for evolutionary algorithm designers is caused by the fact that the design details, i.e., parameter values, largely influence the performance of the algorithm. For instance, an EA with good parameter values can be orders of magnitude better than one with poorly chosen parameter values. Hence, algorithm design in general, and EA design in particular, is an optimization problem itself. In the field of evolutionary computing one traditionally distinguishes two approaches to choosing parameter values, following the scheme offered in [9]:

- *Parameter tuning*, where (good) parameter values are established before the run of a given EA. In this case, parameter values are fixed in the initialization stage and do not change while the EA is running.
- *Parameter control*, where (good) parameter values are established during the run of a given EA. In this case, parameter values are given an initial value when starting the EA and they undergo changes while the EA is running.

There has been much research into parameter control during the last decade. It has been successfully applied in various domains of metaheuristics, including Evolution Strategies [10], Genetic Algorithms [11], Differential Evolution [12,13] and Particle Swarm Optimization [14]. However, the topic of parameter control is beyond the scope of this paper; for a good overview of the area we recommend [9,15].

To obtain a detailed view on parameter tuning, we distinguish three layers: application layer, algorithm layer, and design layer, see Fig. 2. As this figure indicates, the whole scheme can be divided

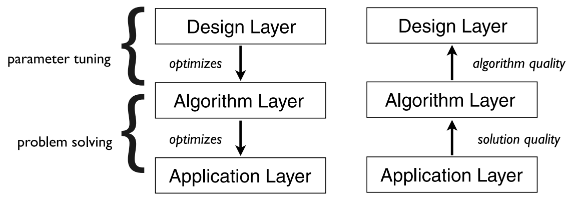

Fig. 2. Control flow (left) and information flow (right) through the three layers in the hierarchy of parameter tuning.

Table 2
Vocabulary distinguishing the main entities in the context of problem solving and parameter tuning.

	Problem solving	Parameter tuning
Method at work	Evolutionary algorithm	Tuning procedure
Search space	Solution vectors	Parameter vectors
Quality	Fitness	Utility
Assessment	Evaluation	Test

into two optimization problems that we refer to as problem solving and parameter tuning. The problem solving part consists of a problem on the application layer and an EA on the algorithm layer trying to find an optimal solution for this problem. Simply put, the EA is iteratively generating candidate solutions seeking one with maximal quality. The parameter tuning part contains a tuning method that is trying to find optimal parameter values for the EA on the algorithm layer. Similarly to the problem solving part, the tuning method is iteratively generating parameter vectors seeking one with maximal quality, where the quality of a given parameter vector is based on the performance of the EA using the values of it. To avoid confusion we use distinct terms to designate the quality function of these optimization problems. Conform the usual EC terminology we use the term *fitness* for the quality of candidate solutions of the problem on the application layer, and the term *utility* to denote the quality of parameter vectors. Table 2 provides a quick overview of the related vocabulary.

With this nomenclature, the problem to be solved by the algorithm designer can be seen as a search problem in the space of parameter vectors given some utility function. Solutions of the parameter tuning problem can then be defined as parameter vectors with maximum utility. Furthermore, we can distinguish the so-called "structural tuning" and "parametric tuning" [16] in a formal way: structural tuning takes place in the space of qualitative parameters, while parametric tuning refers to searching through quantitative parameters.

Now we can define the *utility landscape* as an abstract landscape where the locations are the parameter vectors of an EA and the height reflects utility. It is obvious that fitness landscapes – commonly used in EC – have a lot in common with utility landscapes as introduced here. However, despite the obvious analogies, there are some differences we want to note here. First of all, fitness values are most often deterministic—depending, of course, on the problem instance to be solved. However, the utility values are always stochastic, because they reflect the performance of an EA which is a stochastic search method. This implies that the maximum utility sought by tuning needs to be defined in some statistical sense. In fact, even comparing utility values could be difficult if the underlying data shows a big variance. In Section 4.1 we will return to this issue. Second, the notion of fitness is usually strongly related to the objective function of the problem on the application layer and differences between suitable fitness functions mostly concern arithmetic details. The notion of utility, however, depends on the performance metrics used to define EA quality, thus ultimately on the preferences of the user.

Fig. 3 illustrates the generic scheme of parameter tuning in a graphical form. It shows that the solutions of a tuning problem

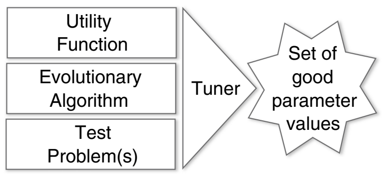

Fig. 3. Generic scheme of parameter tuning showing how good parameter values depend on four factors: the problem instance(s) to be solved, the EA used, the utility function, and the tuner itself.

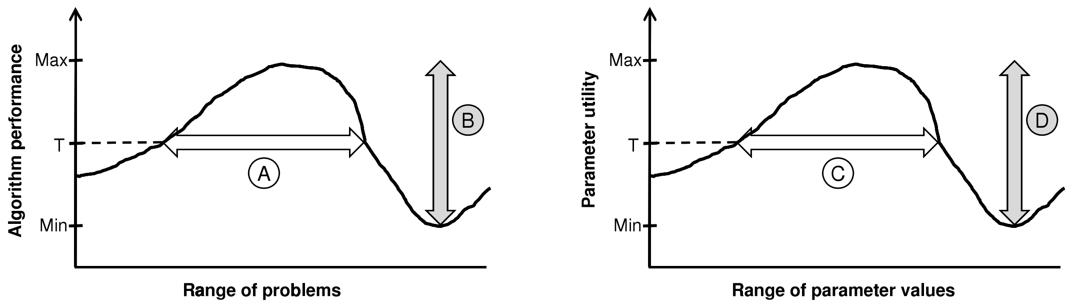

Fig. 4. Illustration of algorithm applicability (A), fallibility (B), tolerance (C), tuneability (D). See the text for detailed explanation.

depend on the problem(s) to be solved, the EA used, and the utility function. Adding the tuner to the equation, we obtain this picture showing how a set of good parameter values obtained through tuning depends on four factors. In the next section we discuss algorithm quality, the basis of utility functions, in more detail.

4. Algorithm quality: performance and robustness

Next we take a closer look on (evolutionary) algorithm quality. To structure our discussion we identify two main factors that determine algorithm quality: performance and robustness.

4.1. Performance measures

In general, there are two atomic performance measures for EAs: one regarding solution quality and one regarding algorithm speed. Most, if not all, performance metrics used in EC are based on variations and combinations of these two. Solution quality can be naturally expressed by the fitness function the EA is using. As for algorithm speed, time or search effort needs to be measured. This can be done by, for instance, the number of fitness evaluations, CPU time, wall-clock time, etc. In [17] we discussed pro's and con's of various time measures. Here we do not go into this issue, just assume that one of them has been chosen. Then there are different combinations of fitness and time that can be used to define algorithm performance in one single run. For instance:

- Given a maximum running time (computational effort), algorithm performance is defined as the best fitness at termination.
- Given a minimum fitness level, algorithm performance is defined as the running time (computational effort) needed to reach it.
- Given a maximum running time (computational effort) and a minimum fitness level, algorithm performance is defined through the Boolean notion of success: a run succeeds if the given fitness is reached within the given time, otherwise it fails.

Obviously, by the stochastic nature of EAs, multiple runs on the same problem are necessary to get a good estimation of performance. By aggregating the measures mentioned above over a number of runs, we obtain the performance metrics commonly used in evolutionary computing, cf. [3, Chapter 14]:

- MBF (mean best fitness).
- AES (average number of evaluations to solution).
- SR (success rate),

respectively. Straightforward as they are, these measures are not always appropriate. The most prominent problem in here is the possibly large variance in the data, i.e., the performance results of the EA in question. If this is the case, then using the mean (and standard deviation) may not be meaningful and the use of the median or the best fitness can be preferable [18]. Visualizations like boxplots and plots of the Empirical Cumulative Distribution Function (ECDF) [19] can be also be useful in such cases.

When designing a good EA, one may tune it to maximize either of these performance measures or a combination of them such as Success Performance (SP) [20]. Obviously, the actual performance metrics determines the utility landscape, and therefore the choice of the best parameter vector. In a recent case study, we have shown that tuning for different performance measures (MBF, SR, or rank) can yield parameter values that differ in orders of magnitude, cf. [21]. This demonstrates that any claim about good parameter values in general, without a reference to the performance measure, should be taken with a grain of salt.

4.2. Robustness

Regarding robustness, the first thing to be noted is that there are different interpretations of this notion in the literature. The existing (informal) definitions do have a common feature: robustness is related to the variance of algorithm performance across some dimension. However, they differ in what this dimension is. To this end, there are indeed more options, given the fact that the performance of an EA (instance) depends on (1) the problem instance it is solving, (2) the parameter vector it uses, (3) the random seed used to realize stochasticity. Therefore, the variance of performance can be considered along three different dimensions: parameter values, problem instances, and random seeds, leading to three different types of robustness.

4.2.1. Robustness to changes in problem specification

In the simplest case, we are tuning an evolutionary algorithm A on one function f. Then the utility of a parameter vector $\bar{p}$ is measured by the performance of the EA instance $A(\bar{p})$ on f. In this case, tuning delivers a *specialist*, that is, an EA instance that is very good in solving f, with no claims or indications regarding its performance on other problem instances. This can be a satisfactory result if one is only interested in solving f. However, algorithm designers in general, and evolutionary computing experts in particular, are often interested in EAs (EA instances) that work well on many objective functions. In such a case, tuning is performed on a test suite consisting of many test functions $f_1, \ldots, f_n$ and it delivers a *generalist*.

The left diagram of Fig. 4 illustrates this matter, exhibiting EA performance across a range of problems. Based on this performance curve we define two properties. If the height of the curve (Max–Min, shown by the arrow B) is large, we call the EA *fallible*, because it can fail greatly on some problems. If the width (shown by the arrow A) is large, we call the EA *widely applicable*. Note that the length of arrow A depends on the performance threshold T. This means that we consider an EA applicable to a problem if its performance exceeds this threshold. Other measures of width or height, or even the two combined (such as standard deviation) are also possible. For the historically inclined readers, the famous figures in the classic books of Goldberg [22, pg. 6], and Michalewicz, [23, pg. 292], refer to this kind of robustness.

It should be noticed that this notion of robustness is applicable to EA instances, and not to EAs. The reason is simple, to measure

Table 3
Six notions related to robustness, based on the variance of algorithm performance across different spaces and directions (width vs. height). The notions tolerant and tuneable apply to EAs, the other four to EA instances or parameter vectors. See text for more details.

Robustness as variance across	Large value for	
	Width	Height
Problem instances	Widely applicable	Fallible
Parameter values	Tolerant	Tuneable
Random seeds	Successful	Unstable

performance one needs to have a fully specified EA instance $A(\bar{p})$. Then it is $A(\bar{p})$, and/or the parameter vector $\bar{p}$, that is robust (fallible, widely applicable).

4.2.2. Robustness to changes in parameter values

Another popular interpretation of algorithm robustness is related to performance variations caused by different parameter values. This notion of robustness is defined for EAs. Of course, it is again the EA instance $A(\bar{p})$ whose performance forms the basic measurement, but here we aggregate over parameter vectors. The right diagram of Fig. 4 shows the performance of $A(\bar{p})$ for different values of $\bar{p}$. In other words, it is a plot of the utility values belonging to different parameter vectors. Based on this curve we define two properties. If the height of the curve (Max–Min, shown by the arrow D) is large, then we call the EA *tuneable*, because it can be made much better or worse by selecting different parameter values.[2] If the size of the parameter space leading to acceptable performance (shown by the arrow C) is large enough, then we call the EA *tolerant*. Note that the length of arrow C depends on the performance threshold T that defines what acceptable performance is. Furthermore, it can be measured per parameter individually, and on the parameter space as a whole. Using such a definition, it is EAs (specified by a particular configuration of the qualitative parameters) that can be compared by their robustness (tuneability or tolerance). Optimizing this quality requires a search through the qualitative parameters.

4.2.3. Robustness to changes in random seeds

EAs are stochastic algorithms, because they rely on random choices in several steps. Therefore, all experimental investigations should be statistically sound, requiring a number of independent repetitions of a run with the same setup, but with different random seeds. Hereby we obtain information over the third kind of robustness. Because it is hard to define a meaningful ordering of random seeds we cannot use an equivalent of the diagrams of Fig. 4 and have to redefine width and height.

Instead of width, here we can use the ratio of runs ending with a good result above some threshold T. This measure is well-known in the literature, in Section 4.1 it was mentioned as success rate. We call an EA instance *successful* if this success rate is high. The equivalent of a height-based notion describes the difference between the worst and best runs among all repetitions using different random seeds. If the difference between the best and worst run is big, then we call this EA instance *unstable*. The height is not often used on its own, although it can support a worst-case analysis. More commonly, height and width are combined in a single measure by means of the standard deviation.

Table 3 summarizes our terminology regarding robustness showing six adjectives that can be used as context specific replacements of the term robust. For a good understanding, it is important to note that the concepts in the left column (those under 'Width') are not the opposites of the ones in the right column (those under 'Height').

2 This notion is very similar to the one used by Preuss [24].

5. Configuring and analyzing EAs by tuning

Based on the previous section we can identify different purposes behind tuning an (evolutionary) algorithm:

1. To obtain an algorithm instance with high performance, given a performance measure or a combination of measures (on one or more problems).
2. To obtain an algorithm instance that is robust to changes in problem specification.
3. To indicate the robustness of the given algorithm to changes in parameter values.
4. To obtain an algorithm instance that is robust to random effects during execution.

Note, that the case of robustness to changes in parameter values is somewhat of an outlier in this list, because we cannot optimize a given EA for this. This is a straightforward consequence of restricting tuning to parametric tuning. That is, if the object to be tuned is an EA with all qualitative parameters specified, then tuning stands for searching for a good vector of quantitative parameters. Then it is easy to see that a good vector can be defined by (1) or (2), or (4) in the above list, but a parameter vector that is robust to changes in parameter values does not make sense. Of course, it is possible to do structural tuning (i.e., searching for a good vector of qualitative parameters) such that we obtain an EA robust to changes in quantitative parameter values. Doing so can reveal interesting and sometimes unexpected relationships between qualitative and quantitative parameters. For instance, in [8] we demonstrated that using crossover in a GA makes the mutation parameters more tolerant. This illustrates that structural tuning can increase robustness to changes in certain quantitative parameter values.

As explained in [1], there is a big difference between competitive and scientific testing of heuristic methods. Simply put, competitive testing is solely aiming at obtaining an algorithmic setup that meets some success criterion, for instance, beats some actual benchmark. Scientific testing is more concerned with gaining insights into an algorithm through controlled experimentation. In general, such a controlled experimentation should aim to study the effects of problem characteristics and algorithm characteristics on algorithm behavior (performance).

In evolutionary computing, tuning usually amounts to competitive testing to configure some evolutionary algorithm in a performance driven manner. In a business context one seeks an algorithm instance that provides a good solution at low computational costs for a given practical problem. In an academic context, a researcher is typically after an instance of his/her newly invented EA that outperforms some benchmark (evolutionary) algorithms on some benchmark problems. In the current EC practice parameter values are mostly selected by conventions, ad hoc choices, and experimental comparisons on a limited scale. In previous publications we discussed the drawbacks of this practice, explained that there are better alternatives in the form of existing tuning algorithms and argued for using them.

For instance, in [8] we experimented with structural and parametric tuning, addressing the costs and gains of tuning quantitatively. We found that EAs differ greatly in the amount of tuning needed to reach a given performance, and this tuning cost depends on the overall setup of the EA (the values of the qualitative parameters), rather than the number of (quantitative) parameters. In [25] we compared three methods, (meta-)evolution strategies, REVAC and SPO for tuning a GA.[3] We studied their differences in finding superior parameter values and the amount

3 See Section 7 and the references therein for details on REVAC and SPO.

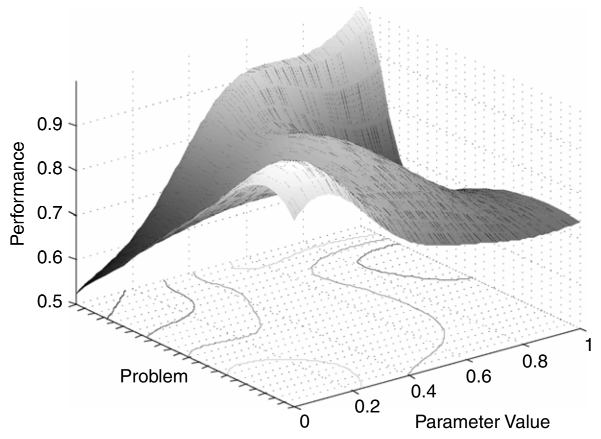

Fig. 5. Illustration of the grand utility landscape showing the performance (z) of EA instances belonging to a given parameter vector (x) on a given problem instance (y). Note: The "cloud" of repeated runs is not shown.

of information they provide. These experiments showed how common wisdom about good parameter values can be easily refuted by algorithmic tuning. In [21] we demonstrated the benefits of tuning in a challenging case study. We applied REVAC to an EA that had been carefully designed and optimized approximating its best possible performance: the winner of the CEC-2005 Special Session on Real-Parameter Optimization, [20]. In just a few days, the world champion has been improved and much has been learned about its parameters. Most recently, we have proposed the use of entropy in a paper focusing on the analysis, rather than the optimization of EAs through parameter tuning [26]. We discussed different notions of entropy and explained in detail how our REVAC method collects entropy data while tuning an EA. We illustrated the matter with experiments that showed how such data can be used to indicate the relevance (tuneability) of qualitative and quantitative parameters and the amount of tuning needed per parameter.

There are two main messages emerging from our experience. First, using tuning algorithms is highly rewarding. The efforts are moderate and the gains in performance can be very significant. Second, by using tuning algorithms one does not only obtain superior parameter values, but also much information about parameter values and algorithm performance. This information can be used to obtain a deeper understanding of the algorithm in question. In other words, using tuning algorithms helps not only in calibrating, but also in analyzing evolutionary algorithms. In terms of testing heuristics, this means a transition from competitive to scientific testing. It is easy to see that a detailed understanding of algorithm behavior has a great practical relevance. Knowing the effects of problem characteristics and algorithm characteristics on algorithm behavior, users can make well-informed design decisions regarding the (evolutionary) algorithm they want to use. Obviously, the biggest impact in this direction is achieved if the tuning data and the aggregated knowledge are shared among users and members of the scientific community, such that individual users/researchers do not have to execute their own tuning sessions.

Fig. 5 shows the kind of data that can be gathered by systematic tuning. For the sake of this illustration, we restrict ourselves to parameter vectors of length $n = 1$. That is, we only take a single parameter into account. Thus, we obtain a 3D landscape with one axis x representing the values of the parameter and another axis y representing the problem instances investigated. (In the general case of n parameters, we have $n+1$ axes here.) The third dimension z shows the performance of the EA instance belonging to a given parameter vector on a given problem instance. It should be noted that for stochastic algorithms, such as EAs, this landscape is blurry if the repetitions with different random seeds are also taken into account. That is, rather than one z-value for a pair $\langle x, y\rangle$, we have one z for every run, for repeated runs we get a "cloud".

Although this 3D landscape gives the best complete overview of performance and robustness, lower-dimensional hyperplanes are also interesting and sometimes more clear. To begin with, let us mention the 1D hyperplane, or slice, corresponding to one pair $\langle x, y\rangle$. In the full picture including the "cloud", such a slice contains the outcomes of all repetitions, thus data about robustness to changes in random seeds as discussed Section 4.2.3. Such data are often reported in the EC literature in the form of the average performance and the corresponding standard deviation for a given EA instance (x) on a given problem instance (y), or they are visualized by means of boxplots and graphs of the Empirical Cumulative Distribution Function (ECDF) [19].

The left-hand-side of Fig. 6 shows 2D slices corresponding to specific parameter vectors, thus specific EA instances. Such a slice shows how the performance of a specific EA instance varies over the range of problem instances. This provides information on robustness to changes in problem specification, for instance if the given EA instance is fallible or widely applicable, cf. Section 4.2.1. Such data are often reported in the EC literature, be it with a different presentation. A frequently used option is to show a table containing the experimental outcomes (performance results) of one or more EA instances on a predefined test suite, e.g., the five DeJong functions, the 25 functions of the CEC-2005 contest, etc. Notice that the two EA instances depicted on left-hand-side of Fig. 6 are quite different regarding their behavior. For instance, the foremost EA instance is rather fallible, while the second one is widely applicable (assuming a threshold $T = 0.75$).

The right-hand-side of Fig. 6 shows 2D slices corresponding to specific problem instances. On such a slice we see how the

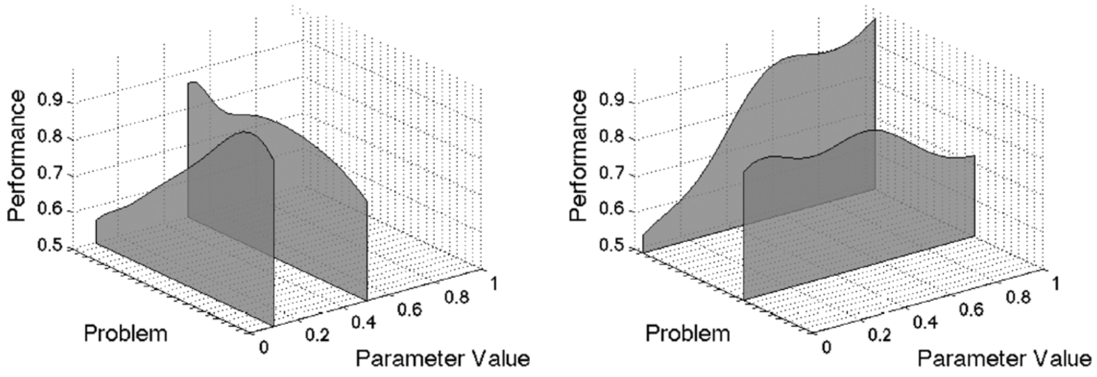

Fig. 6. Illustration of parameter-wise slices (left) and problem-wise slices (right) of the grand utility landscape shown in Fig. 5. (The "cloud" of repeated runs is not shown.) See the text for explanation and interpretation.

Table 4
Tuning methods distinguished by taxonomy T_1 & T_2.

Described in section	Method	Taxonomy T_1	Taxonomy T_2
Section 7.1	Latin-Square [29]	Single stage	A
	Taguchi Orthogonal Arrays [30]	Single stage	A
Section 7.1.1	CALIBRA [40]	Multi stage	A
	Empirical Modelling of Genetic Algorithms [29]	Two stage	A
Section 7.2	Sequential Experiment Designs [63]	Single stage	A
	François–Lavergne [45]	Single stage	A
	Logistic Regression [44]	Single stage	A
	ANOVA [34]	Single stage	A
	Design of Experiments with Regression Tree [64]	Single stage	A
Section 7.2.1	Coy's procedure [46]	Multi stage	A
	Sequential Parameter Optimization (SPO) [36]	Multi stage	A
	SPO + OCBA [47]	Multi stage	A
Section 7.3	Interactive Analysis [49]	Single stage	B
	Ranking and Selection [50]	Single stage	B
	Multiple Comparison Procedures [51]	Single stage	B
	Sequential indifference-zone selection [52]	Single stage	B
	Racing [35]	Single stage	B
	F-RACE [16]	Single stage	B
Section 7.3.1	Iterative F-RACE [39]	Multi stage	A & B
Section 7.4	Meta-Plan [54]	Multi stage	A
	Meta-Algorithm [32]	Multi stage	A
	Meta-GA [65]	Multi stage	A
	Meta-GA + Racing [5]	Multi stage	A & B
	FocusedILS [55]	Multi stage	A & B
	Meta-ES [66]	Multi stage	A
	Meta-CMA-ES [25]	Multi stage	A
	OPSO [67]	Multi stage	A
Section 7.4.1	Local Unimodal Sampling [59]	Multi stage	A
	REVAC [68]	Multi stage	A
	REVAC ++ [25]	Multi stage	A & B
Section 7.4.2	M-FETA [60]	Multi stage	A & B
	Performance Fronts [61]	Multi stage	A

performance of the given EA depends on the parameter vectors it uses. This discloses information regarding robustness to changes in parameter values (e.g., tuneability and tolerance), as discussed in Section 4.2.2. In evolutionary computing such data is hardly ever published. This is a straightforward consequence of the current practice, where parameter values are mostly selected by conventions, ad hoc choices, and very limited experimental comparisons. In other words, usually such data is not even produced, let alone stored and presented. By the increased adoption of tuning algorithms this practice could change and knowledge about EA parameterization could be collected and disseminated.

For tuning problems involving multiple parameters, similar, but higher dimensional, planes exist which provide the same information as in this simplified scenario. However, visualizing such hyperplanes is much harder and requires dimension-reduction approaches, such as contour-plots [27], aggregations such as boxplots and ECDFs. There are also visualizations that are specifically designed for analyzing utility landscapes such as entropy [26] plots that can be used to assess the tolerance and tuneability of an algorithm.

6. Positioning tuning methods

In this section we describe three different ways to distinguish and classify tuning methods. The first two are strongly oriented to the internal algorithmic differences between tuners. In [28] we used the resulting taxonomies to structure a survey of tuning algorithms. The overview in this paper is organized differently, led by a new taxonomy, taking into account the different notions of robustness, or, more generally, various aspects of scientific testing.

6.1. *Tuning algorithms: taxonomy T_1*

In essence, all tuning algorithms work by the GENERATE-and-TEST principle, i.e., through generating parameter vectors and testing them to establish their utility. Tuners can be then divided into two main categories:

1. *non-iterative* and
2. *iterative* tuners.

All non-iterative tuners execute the GENERATE step only once, during initialization, thus creating a fixed set of vectors. Each of those vectors is then tested during the TEST phase to find the best vector in the given set. Hence, one could say that non-iterative tuners follow the INITIALIZE-and-TEST template. Initialization can be done by random sampling, generating a systematic grid in the parameter space, or some space filling set of vectors. Examples of such methods are Latin-Square [29] and Taguchi Orthogonal Arrays [30].

The second category of tuners is formed by iterative methods that do not fix the set of vectors during initialization, but start with a small initial set and create new vectors iteratively during execution. Common examples of such methods are meta-EAs (Section 7.4) and Iterative Sampling Methods (Section 7.1.1). Section 7 and Table 4 provide a more elaborate overview of algorithms and their classification based on this taxonomy.

Given the stochastic nature of EAs, a number of tests is necessary for a reliable estimate of utility. Following [31,4], we distinguish

1. *single-stage* and
2. *multi-stage procedures.*

Single-stage procedures perform the same number of tests for each given vector, while multi-stage procedures use a more sophisticated strategy. In general, they augment the TEST step by adding a SELECT step, where only promising vectors are selected for further testing, deliberately ignoring those with a low performance.

6.2. Tuning algorithms: taxonomy T_2

The taxonomy presented in this section is based on the search effort perspective. Obviously, good tuning algorithms try to find a good parameter vector with the least possible effort. In general, the total search effort can be expressed as $A \times B \times C$, where

- A is the number of parameter vectors to be tested by the tuner.
- B is the number of tests, e.g., EA runs, per parameter vector to establish its utility. The product $A \times B$ represents the total number of algorithm runs used by the tuners.
- C is the number of function evaluations performed in one run of the EA.

Based on this perspective, we divide existing tuning methods into four categories: those that try to allocate search efforts optimally by saving on A, B, C, respectively. In addition, there are tuners that try to allocate search efforts optimally by saving on A and B.

Methods for optimizing A are trying to allocate search efforts efficiently by cleverly generating parameter vectors. Strictly speaking they might not always minimize A, but try to "optimize the spending", that is, get the most out of testing A parameter vectors. Such tuners are usually iterative methods. The idea behind most tuners in this category is to start with a relatively small set of vectors and iteratively generate new sets in a clever way, i.e., such that new vectors are likely to be good. Well-known examples in this category are the classical meta-GA [32] and the more recent REVAC method [33]. These tuners are only appropriate for quantitative parameters, because qualitative parameters do not have an ordering that could be exploited by a search algorithm, cf. Section 2. Formally, a meta-EA could work on parameter vectors that contain qualitative parameters too. But even then, its working would boil down to random sampling in the (sub)space of qualitative parameters.

Methods for optimizing B are trying to reduce the number of tests per parameter vector. The fundamental dilemma here is that fewer tests yield less reliable estimates of utility. If the number of tests is too low, then the utilities of two parameter vectors might not be statistically distinguishable. More tests can improve (sharpen) the reliability of the utility estimates such that the superiority of one vector over the other can be safely concluded. However, more tests come with a price in the form of longer runtimes. The methods in this group use the same trick to deal with this problem: initially they perform only a few tests per parameter vector and increase this number to the minimum level that is enough to obtain statistically sound comparisons between the given parameter vectors. Such methods are known as *statistical screening, ranking and selection*. Well-known examples in this category are ANOVA [34] and racing [35]. These tuners are in principle applicable for quantitative and qualitative parameters.

Obviously, the greatest benefit in terms of reduced tuning effort would come from optimizing both A and B. Currently there are a few tuning methods based this idea, for example, Sequential Parameter Optimization (SPO) [36–38] and REVAC++ [25].

Optimizing/reducing the number of fitness evaluations per EA run (C) amounts to terminating a run before the maximum number of fitness evaluations is reached. This could be done, for instance, by a mechanism which detects that the given run is not worth to be continued. In principle, this could lead to a great reduction of tuning effort too. However, as of summer 2010 we are not aware of any parameter tuners in this category.

An overview of algorithms and their classification based on this taxonomy is shown in Table 4.

6.3. Tuning algorithms: taxonomy T_3

Section 4 specifies four different quality indicators that can be used to evaluate evolutionary algorithms. In order to find values for these indicators, knowledge of the grand utility landscape over the set of all possible parameter values and the set of all possible problems is required. Since both sets are infinitely large, just enumerating is not feasible. However, many parameter tuning algorithms are not only able to find the best parameter values, but also supply information about various robustness indicators.

The quality and quantity of these indicators varies per tuning method. In general, there are two main tasks for parameter tuning. The first task is to find the parameter vector(s) with the highest possible performance. This could be perceived as *exploitation* of knowledge about the parameter values. The second one is the task of acquiring much information on robustness. This requires *exploration* of the parameter space. Each of the tuning methods has a different balance between exploitation and exploration, and focuses on different types of robustness. Based on this distinction, we can identify four main approaches, namely an approach solely for finding the best performing vectors (meta-EA), an approach mainly aimed at providing information (sampling), and two approaches that are a combination of both (screening and model-based).

Although each of the main approaches uses distinct techniques and has a different goal, within each of those approaches specialized methods have emerged that are hybrids, and incorporate ideas and objectives from the other approaches. Approaches designed for finding the best possible parameter vector are enhanced with features that provide more information about robustness (such as REVAC [33]), while approaches from the informative type are improved for delivering high performing parameter values (such as I/F-RACE [39] and CALIBRA [40]). Most of these methods even allow for a smooth transition between exploitation and exploration by defining suitable parameters values.

7. Survey of tuning methods

In this section we provide an overview of search methods for tuning evolutionary algorithms. To organize this overview, each of the tuning methods is assigned to a certain category according to taxonomy T_3. Hence, the four main categories are: sampling methods, model-based methods, screening methods, and meta-evolutionary algorithms. Furthermore, the different branches are shown in which an approach from the main category is altered to specialize more on one specific task.

7.1. Sampling methods

Sampling methods can be described as methods that reduce the search effort by cutting the number of parameter vectors tested (A) with respect to a full factorial design. The two most commonly used sampling methods are Latin-Square [29] and Taguchi Orthogonal Arrays [30]. The outcome of a sampling method session needs to be analyzed afterward to predict which parameter values work best and which are the most robust. Therefore, most sampling methods are mainly used as a starting point for model-based Methods or as an initialization method, rather than as independent parameter tuners. The lack of search refinement leads to parameter vectors of low quality and causes that the information that can be acquired is rather limited, especially on algorithms with a low tolerance.

7.1.1. Iterative sampling methods

CALIBRA [40] and Empirical Modelling of Genetic Algorithms [29] are examples of iterative sampling methods. Unlike the single-stage sampling methods, they refine the area from which new points are sampled in each iteration. Hence, they can be used as independent tuners. The method in [29] is a two-stage procedure that starts with a graeco-latin square over the whole parameter space and proceeds with a fully crossed factorial design with narrowed ranges. CALIBRA starts with a full factorial experiment, based on the 1st and 3rd quantile within the range of each parameter. Using these outcomes, new vectors for the next iteration are generated based on a Taguchi Orthogonal Array with three (narrowed) levels and this procedure is repeated until the maximum number of tests is reached.

Since both methods require a quality measure to define a promising area for resampling, they can only be used to tune for optimizing EA performance, not for robustness. However, if such a choice is made, the thorough parameter sweep that is performed allows for a detailed analysis of the robustness in the parameter space and solutions of reasonable quality.

7.2. *Model-based methods*

Using meta models or surrogate models is a well established approach for the optimization of computationally expensive problems by (evolutionary) search [41,42]. Applied for parameter tuning, such a method constructs a model of the utility landscape and reduces the number of tests (B) by replacing some of the real tests by using the model estimates. The model is constructed based on data about parameters and their utility, delivered by testing. A common approach is to use a regression method to predict the utility of an unknown parameter vector [43–45]. The model is then a formula that maps parameter values to an estimated utility value $\hat{u}$. Eq. (1) is an example of such a mapping for two parameters p_1, p_2.

$$\hat{u}(p_1, p_2) = \beta_0 + \beta_1 \cdot p_1 + \beta_2 \cdot p_1^2 + \beta_3 \cdot p_2 + \beta_4 \cdot p_2^2 \cdot \beta_5 \cdot p_1 \cdot p_2. \quad (1)$$

In such a formula, the β values indicate the relevance of the associated parameter. A high β value leads to large deviations in utility when varied, and therefore indicates an EA with a high tuneability. Based on these values, it is also possible to estimate the tolerance of the algorithm on a given level of performance. If the model-based method is applied to multiple problems, then the difference in β values also indicates a lower robustness to changes in problem definition. However, as model-based methods rely on a single-stage sampling method for generating a population, the quality of the best vector found and the quality of the model is relatively low.

7.2.1. *Iterative model-based methods*

Iterative model-based methods are developed to overcome this issue and allow for a more fine-grained search of the parameter space. Coy's procedure [46] is one of the most basic extensions, where the standard Model-Based Method is followed by a local search procedure to optimize on the parameter values. It describes a two-stage procedure in which the first stage is used to find a model, and the second stage is specifically aimed at identifying the best parameter vector. The first stage consists of a full factorial design over the whole parameter space. The outcomes are used to fit a linear regression model, and to determine the path of steepest descent. In the second stage, this path is followed and new vectors are generated and tested until the best solution found has not changed for a specified number of steps. As the model is not updated in this second stage, the quality of the best parameter vector found heavily depends on the correctness of the model in the first stage.

Unlike the two-stage procedure of Coy, Sequential Parameter Optimization (SPO) [36,47] performs a true multi-stage procedure where the model is constantly updated. Each iteration starts with generating a set of new vectors and predicting their utility using the model. The vectors with the highest predicted utility are then tested to determine their 'true' utility, and these measured utility values are used to update the model for the next iteration. After reaching the maximum number of tests, the procedure terminates with an accurate model of the most promising areas. Obviously, both the accuracy of the model and the quality of the best parameter vector depend on the type of model used. Although this can be any model, such as regression trees or logistic regression, the authors advocate the use of Kriging to model the utility landscape. In a comparative study it has been shown that SPO is able to find high quality parameter vectors comparable to the ones found by meta-evolutionary algorithms [25]. This, combined with the information that can be extracted from the resulting models, makes it a high quality parameter tuner.

7.3. *Screening methods*

The idea behind screening methods is to identify the best parameter vector from a given a set of vectors with a minimum number of tests. They are trying to save on B by iteratively testing only those vectors that deserve further investigation. The chosen vectors are (re-)tested and the whole process is repeated until no further testing is needed. Therefore, they can either identify the best parameter vector with less computational effort than sampling methods, or investigate a larger set of parameter vectors with the same computational effort. In the latter case, the quality of best parameter vector found is likely to be higher and there is also more information to estimate the robustness to changes in parameter values.

Screening methods are one of the oldest approaches to parameter tuning and are heavily influenced by the field of system selection, where the objective is to select the best option from a range of competing systems with as few stochastic simulations as necessary [48]. As parameter vectors can be seen as competing systems and a run of the algorithm as a stochastic simulation, methods from the field of system selection can be seen as parameter tuning approaches. Interactive Analysis (IA) [49], Ranking and Selection (R&S) [50], Multiple Comparison Procedures (MPC) [51] and Fully sequential indifference-zone Selection Procedure (FSP) [52] are the four main approaches from this field [53]. Their main differences are in the guarantees that can be given about the selection of the best system and the required number of repetitions. As with most screening methods, all four rely on the assumption that the outcomes of the simulation are normally distributed. The extent to which this assumption holds is of course doubtful, although by means of batching [53], these assumptions can be met. The main advantage is that such methods guarantee that the system indicated as the best, is either within a certain range (R&S and FSP) or has a certain confidence level (MPC). A more detailed overview of the differences between these algorithms is in [48,53].

For the specific application of parameter tuning, the term racing is adopted from [35]. Although in essence it is not much different from the system selection mechanisms, racing methods are aimed at selecting the best system from a very large set, while most system selection methods only deal with relatively few competing systems. Furthermore, both Hoeffding Races [35] (that uses Hoeffding's bound) and F-RACE [16] (that uses the Friedman's two-way ANOVA by ranks statistical test) use tests that work without any assumptions about the underlying distribution, giving them an advantage over system selection methods.

7.3.1. *Iterative screening methods*

Iterative F-RACE (I/F-RACE) [39] is an extension to F-RACE [16] with the specific goal of combining screening methods and a fine-grained search. Initially, an I/F-RACE starts with a region as big as the parameter space that is used to sample a relatively small population of vectors. Using the racing techniques from F-RACE, the number of vectors in this population is reduced until a certain condition is met. However, unlike standard F-RACE, this is only the start of the procedure. Namely, a multi-variate normal distribution is fit on the surviving vectors, which is then used as a probability density function to sample points for a new population. The whole procedure of screening and generating new points can be repeated again, until the maximum number of tests is reached. Because I/F-RACE is a multi-stage method that samples from a distribution, it can be seen as a basic form of an iterative model-based method, or a special form of meta-evolutionary algorithm, and therefore has many of the same characteristics, such as good performance and valuable information about parameter robustness.

7.4. Meta-evolutionary algorithms

Finding parameter vectors with a high utility is a complex optimization task with a nonlinear objective function, interacting variables, multiple local optima, noise, and a lack of analytic solvers. Ironically, this is exactly the type of problem where EAs are very competitive heuristic solvers. Therefore, it is a natural idea to use an evolutionary approach to optimize the parameters of an evolutionary algorithm.

The idea of a meta-EA was already introduced in 1978 by Mercer and Sampson [54], but due to the large computational costs, their research was very limited. Greffenstette [32] conducted more extensive experiments with his Meta-GA and showed its effectiveness. In general, the individuals used in a meta-EA are parameter vectors of the baseline EA to be tuned and the (meta) fitness of such a vector is its utility, determined by running the baseline EA with the given parameter values. Using this representation and utility as (meta) fitness, any evolutionary algorithm can be used as a meta-EA, if only it can cope with the given vector representation. However, the tuning problem has two challenging characteristics, the noise in (meta) fitness values and the very expensive (meta) evaluations. This gave rise to more tuning-specific algorithms that use the same techniques as screening methods.

FocusedILS [55] is such a method, that adds a screening approach to an existing technique. It is an extension to the ParamILS framework [56] that describes a work-flow very similar to a $(1 + 1)$ evolution strategy. It differs with respect to the variation operation, which only changes a single parameter value, rather than applying a Gaussian perturbation to the whole vector. Furthermore, it requires the abstract procedure '$\bar{x}$ *is better than* $\bar{y}$' to be defined. The basic implementation of this procedure is based on comparing the average utilities over N runs, however in FocusedILS this is replaced by racing. A similar enhancement is proposed in [5] which adds racing to a Meta-GA for tuning both numerical and symbolic parameters. Both show the added value of such an approach in terms of the number of parameters that is tested, speed, and the quality of the best parameter vector.

Although meta-EAs turn out to be excellent in finding high quality vectors [25], they do not provide any model of the utility landscape, nor insights into the different types of robustness of the baseline EA.

7.4.1. Enhanced meta-evolutionary algorithms

To enhance meta-EAs with such features, Nannen and Eiben introduced a method for Relevance Estimation and Value Calibration of parameters (REVAC) [57,33]. In essence, REVAC is a specific type of meta-EA where the population approximates the probability density function of the most promising areas of the utility landscape, similar to Iterative F-RACE. This function is rather simple –decomposed by coordinates (parameters), hence blind for parameter interactions– but can be used for analyzing the sensitivity and relevance of the different parameters and the costs of tuning each parameter [8]. Furthermore, REVAC has been extended with racing and sharpening techniques [25] in order to deal with the stochasticity of the utility values more effectively.

A third extension to REVAC, introduced in [58], aims at finding parameter values that can be considered as widely applicable. The approach is similar to that of Local Unimodal Sampling [59] in which the utility of a parameter vector is defined as the average utility over a range of problems. This approach is further extended in [21] to allow for more sophisticated methods of aggregating the performance of multiple runs. Results show that such an enhanced meta-EA is able to find much better robust parameter values than one could do by hand, as it greatly improved the performance of the winner of the competition on the CEC-2005 test suite, cf. [21].

7.4.2. Multi-objective meta-evolutionary algorithms

Sometimes the application and/or user requirements imply that multiple problems or performance indicators are taken into account. A straightforward approach then is to aggregate these into one measure in order to optimize all of them. However, [58] shows that such an approach runs into several problems. For instance, if the test suite contains objective functions with different levels of difficulty (as most test suites do), then the tuner favors parameter values that make the baseline EA good on the hard test functions, because this leads to higher overall gains than improving performance on problems that are solved very well already. This introduces a (hidden) bias that is not intended by the user.

Alternatives to the simple aggregation approach can be sought by approaching the multi-function tuning problem as a multi-objective optimization problem. This view is quite natural as each performance measure and fitness function in the test suite can be regarded as one objective. Tuning along this line of thought can leverage on existing knowledge regarding multi-objective EAs. These create a Parameter Pareto Front that can be used to evaluate robustness to changes in problem definition, as well as performance using multiple performance criteria [60].

The method presented by Dréo in [61] used only multiple performance measures, as the test suite consisted of a single problem. To estimate the utility using each of these performance measures, a fixed number of repetitive runs were performed and averaged. The tuning algorithm itself was NSGA-II, a well-known multi-objective optimization algorithm [62]. Although in [61] only speed and accuracy are defined as objectives, it can be easily extended to tune for stability too. In [60], the authors took a completely different approach and built a tuning algorithm specifically designed for multi-objective parameter tuning. The main advantage of the new algorithm, M-FETA, is the presence of special operators that reduce the number of tests per vector (B). By assuming that the utility landscape is fairly smooth, the utility of neighboring parameter vectors can be used for estimating the utility of a vector. Thus, similarly to model-based tuning, the need for expensive real tests is reduced. During the run of the tuner, these estimations are sharpened by generating new vectors close to the ones that need more investigation, thus narrowing down the neighborhood-size in those areas. Vectors with a high performance, or a reasonable performance and high variance (due to a large neighborhood-size), are regarded as the ones that need more investigation, due to the fact that M-FETA adopts statistical tests to evaluate dominance.

Inherently to their multi-objective character, these tuners return a diverse set of parameter vectors. Compared to the standard meta-EA approach, this usually leads to a better estimation of parameter robustness at the cost of a lower utility of the best parameter vector found. However, a substantial added value lies in the insights regarding applicability and fallibility, which are quite unique and hard to get using other approaches.

Table 5
Tuning methods distinguished by taxonomy T_3.

Described in section	Method	Performance		Seeds		Parameters		Problems	
		Quality	Speed	Success	Stability	Tolerance	Tuneability	Applicability	Fallibility
Section 7.1	Latin-Square [29]	−−	−−	−−	□	−−	−−	−−	−−
	Taguchi Orthogonal Arrays [30]	−−	−−	−−	□	−−	−−	−−	−−
Section 7.1.1	CALIBRA [40]		□		□	+	+	−−	−−
	Empirical Modelling of Genetic Algorithms [29]		□		−	+	+	−−	−−
Section 7.2	Sequential Experiment Designs [63]	−	−	−	−	□	□	□	□
	François–Lavergne [45]	−	−	−	−	□	□	□	□
	Logistic Regression [44]	−	−	−	−	□	□	□	□
	ANOVA [34]	−	−	−	−	□	□	□	□
	Design of Experiments with Regression Tree [64]	−	−	−	−	□	□	□	□
Section 7.2.1	Coy's Procedure [46]		+		−	□	□	□	□
	Sequential Parameter Optimization (SPO) [36]		++		−	++	++	+	+
	SPO + OCBA [47]		++		−	++	++	+	+
Section 7.3	Interactive Analysis [49]	−			−−	□	+	−−	−−
	Ranking and Selection [50]	−			−−	□	+	−−	−−
	Multiple Comparison Procedures [51]	−			−−	□	+	−−	−−
	Sequential indifference-zone selection [52]	−			−−	□	+	−−	−−
	Racing [35]	−			−−	□	+	−−	−−
	F-RACE [16]	−			−−	□	+	−−	−−
Section 7.3.1	Iterative F-RACE [39]		+		−−	++	+	−−	−−
Section 7.4	Meta-Plan [54]		++		−	−−	−−	−−	−−
	Meta-Algorithm [32]		++		−	−−	−−	−−	−−
	Meta-GA [65]		++		−	−−	−−	−−	−−
	Meta-GA + Racing [5]		++		−	−−	−−	−−	−−
	FocusedILS [55]		++		−	−−	−−	−−	−−
	Meta-ES [66]		++		−	−−	−−	−−	−−
	Meta-CMA-ES [25]		++		−	−−	−−	−−	−−
	OPSO [67]		++		−	−−	−−	−−	−−
Section 7.4.1	Local Unimodal Sampling [59]		+		−	−−	−−	□	□
	REVAC [68]		+		−	++	+	−−	−−
	REVAC ++ [25]		++		−−	++	+	□	□
Section 7.4.2	M-FETA [60]	+	+	+	+	□	□	++	++
	Performance Fronts [61]	+	+	+	+	−	−	+	+

Algorithms that span multiple columns can optimize/give information on one of the columns at the time.
++ excellent quality, + good quality, □ reasonable quality, − poor quality, −− very poor quality.

7.5. Overview of tuner capabilities

Our overview is summarized by Table 5 listing the parameter tuning approaches in this survey, showing to what extent they are able to:

1. Tune an EA for high performance, given a performance measure.
2. Tune an EA to be robust to random variations, or at least indicate this kind of robustness.
3. Indicate the robustness of an EA to changes in parameter values.
4. Tune an EA to be robust to changes in problem specification, or at least indicate this kind of robustness.

On each aspect, the algorithms are rated varying from ++ to −−. Grades are given based on the limits imposed by techniques that are used. Algorithms for optimizing B are generally worse in finding the optimal performance, as they depend on an initial static set of vectors. On the other hand, algorithms for optimizing A are generally worse in generating models and assessing robustness, because they are aimed at finding the best parameter vector. In case an algorithm optimizes both on A and B, then the grade is given based on the focus of the algorithm. In general, methods receive a high grade on the aspects they are designed for and a low grade on the aspects not taken into account in the algorithm design.

Furthermore, if an algorithm is only able to optimize, or to give information, on one aspect at the time (for example only a single performance measure), a combined grade is given.

8. Parameter tuning and research methodology

As explained in the Introduction, there are three main subjects treated in this paper: a conceptual framework, a survey, and a tuning-aware experimental methodology. In this section we reconsider all these matters and conclude the paper by summarizing the principal messages.

Let us begin with noting that parameter tuning in EC has been a largely ignored issue for a long time. Over the last couple of years, there are promising developments (tuning related publications and software), but still, in the current EC practice parameter values are mostly selected by conventions, ad hoc choices, and very limited experimental comparisons. To this end, our main message is that there are well-working alternatives in the form of existing tuning algorithms. These can deliver good parameter values at moderate costs. This observation is not new per se. In fact, all publications about tuning methods, including ours from the past, carry the same basic message. A new angle here is to consider the impact of widely using algorithmic parameter tuners that enable measuring tuning efforts. Such a new practice will enable better founded experimental comparisons.

To illustrate the methodological improvement let us compare the kind of claims a traditional paper and a tuning-aware paper would make when assessing a new EA (*NEA*) by comparing it experimentally to a benchmark EA (*BEA*). In a traditional paper, *NEA* and *BEA* are presented by explaining their qualitative parameter values (operators for selection, variation, population management, etc.), followed by the specification of the values for their quantitative parameters, say $\bar{p}$ and $\bar{q}$, without any information on why and how $\bar{p}$ and $\bar{q}$ are selected. Then the results obtained with $NEA(\bar{p})$ and $BEA(\bar{q})$ are presented to underpin the main findings. These findings are typically claims about $NEA(\bar{p})$ being better than $BEA(\bar{q})$, most often also inferring that *NEA* is better than *BEA* (e.g., that the new crossover in *NEA* is superior). Following the new tuning-aware methodology, *NEA* and *BEA* are presented and tuned with the same tuning method spending the same tuning effort X, thus arriving to a motivated and documented choice for $\bar{p}$ and $\bar{q}$. Then the results obtained with $NEA(\bar{p})$ and $BEA(\bar{q})$ are presented to underpin the claim that the practical best of *NEA* is better than the practical best of *BEA*. (Obviously, ‘‘practical best’’ stands for ‘‘after spending effort X on tuning it’’.) In [28] we discuss tuning methods from the competitive testing perspective and elaborate on related methodological issues in more details.

An important aspect to be noted is the impact of the tuner and the tuning budget (the effort X that can be spent on tuning). Obviously, the practical best of any EA, as we regard it here, depends on both of them. Thus, claims about the practical best of EAs are subject to these two choices—hyper-parameters, if you wish. It might be argued that the problem of arbitrary parameters is now repeated at a higher level. Yet, the tuning-aware comparisons as we advocate here are preferable to the old practice, because they support informed design choices and accumulate much information about EAs.

Further to improved experimental comparisons, the wide adoption of parameter tuners would enable better evolutionary algorithm design. As mentioned before, using tuning algorithms one cannot only obtain superior parameter values, but also much information about problem instances, parameter values, and algorithm performance. This information can be used to analyze EAs and to obtain a deeper understanding of them. This is obviously a long term benefit, but such information is also useful on the short and mid term as it can serve as empirical evidence to justify design decisions. To illuminate this matter, let us consider some examples.

In practice, there might be (and in our experience: there is) a group of EA users who do not have the resources and the willingness to tune an algorithm for their specific problem. Rather, they are interested in a good EA off-the-shelf. Therefore, available knowledge about variations of EA performance along the problem-dimension is particularly interesting for them. If the given problem (instance) can be related to a known type of problems then the user can make an informed choice for an algorithm setup with a high performance on that specific type. On the other hand, if the problem at hand does not belong to a known problem type, or if there is not enough information on that type, then right choice is a widely applicable EA, especially if its parameters are very tolerant. This increases the chances that the untuned instantiation will find good solutions. Yet another case concerns users with repetitive problems, for example, a parcel delivery company that needs to solve almost the same routing problem instance every day. As explained in [3, Chapter 14], the appropriate algorithm in this case is one that is robust to changes in random seeds, as this increases the chances that a decent solution will be found every day with a low probability of making big mistakes. In all these examples, the user depends on the availability of information about the robustness of EA parameters.

Finally, let us touch upon the conditions for benefiting EA design through tuning. Obviously, it is required that data obtained by tuning is preserved and made available for analysis. This can be done on different scales, ranging from one single user (academic or industrial), through a group of users (research group or R&D department), up to the whole evolutionary computing community. Furthermore, the tuning methods themselves need some revision too. They need to work with a minimum user effort and provide the functionalities to store and present (visualize) all information necessary for a thorough analysis. In essence, this represents a requirement for the developers of tuning methods. The acceptance by the community and the obtainable benefits will be realized only if tuning methods are available online, are user-friendly, and assist the user in taking a more scientific testing approach.

References

[1] J. Hooker, Testing heuristics: we have it all wrong, Journal of Heuristics 1 (1995) 33–42.
[2] R. Barr, B. Golden, J. Kelly, M. Rescende, W. Stewart, Designing and reporting on computational experiments with heuristic methods, Journal of Heuristics 1 (1995) 9–32.
[3] A.E. Eiben, J. Smith, Introduction to Evolutionary Computing, Springer-Verlag, London, 2003.
[4] T. Bartz-Beielstein, Experimental Research in Evolutionary Computation—The New Experimentalism, in: Natural Computing Series, Springer, Berlin, Heidelberg, New York, 2006.
[5] B. Yuan, M. Gallagher, Combining meta-EAs and racing for difficult EA parameter tuning tasks, in: [15], pp. 121–142.
[6] M. Birattari, Z. Yuan, P. Balaprakash, T. Stützle, F-race and iterated F-race: an overview, in: T. Bartz-Beielstein, M. Chiarandini, L. Paquete, M. Preuss (Eds.), Experimental Methods for the Analysis of Optimization Algorithms, Springer, 2010, pp. 311–336.
[7] J. Maturana, F. Lardeux, F. Saubion, Controlling behavioral and structural parameters in evolutionary algorithms, in: International Conference on Artificial Evolution, EA'2009, in: Lecture Notes in Computer Science, vol. 5926, Springer, 2009, pp. 110–121.
[8] V. Nannen, S.K. Smit, A.E. Eiben, Costs and benefits of tuning parameters of evolutionary algorithms, in: G. Rudolph, T. Jansen, S.M. Lucas, C. Poloni, N. Beume (Eds.), Parallel Problem Solving from Nature – PPSN X, in: Lecture Notes in Computer Science, vol. 5199, Springer, 2008, pp. 528–538.
[9] A.E. Eiben, R. Hinterding, Z. Michalewicz, Parameter control in evolutionary algorithms, IEEE Transactions on Evolutionary Computation 3 (1999) 124–141.
[10] O. Kramer, Evolutionary self-adaptation: a survey of operators and strategy parameters, Evolutionary Intelligence 3 (2010) 51–65. On-line first.
[11] A. Fialho, Adaptive operator selection for optimization, Ph.D. Thesis, Université Paris-Sud XI, Orsay, France, 2010.
[12] A.K. Qin, V.L. Huang, P.N. Suganthan, Differential evolution algorithm with strategy adaptation for global numerical optimization, Transactions on Evolutionary Computation 13 (2009) 398–417.
[13] R. Mallipeddi, P. Suganthan, Differential evolution algorithm with ensemble of parameters and mutation and crossover strategies, in: B. Panigrahi, S. Das, P. Suganthan, S. Dash (Eds.), Swarm, Evolutionary, and Memetic Computing, in: Lecture Notes in Computer Science, vol. 6466, Springer, Berlin, Heidelberg, 2010, pp. 71–78.
[14] Z.-H. Zhan, J. Zhang, Adaptive particle swarm optimization, in: M. Dorigo, M. Birattari, C. Blum, M. Clerc, T. Stützle, A. Winfield (Eds.), Ant Colony Optimization and Swarm Intelligence, in: Lecture Notes in Computer Science, vol. 5217, Springer, Berlin, Heidelberg, 2008, pp. 227–234.
[15] F. Lobo, C. Lima, Z. Michalewicz, Parameter Setting in Evolutionary Algorithms, Springer, 2007.
[16] M. Birattari, Tuning Metaheuristics, Springer, 2005.
[17] A.E. Eiben, M. Jelasity, A critical note on experimental research methodology in experimental research methodology in EC, in: 2002 Congress on Evolutionary Computation, CEC'2002, IEEE Press, Piscataway, NJ, 2002, pp. 582–587.
[18] T. Bartz-Beielstein, New experimentalism applied to evolutionary computation, Ph.D. Thesis, Universität Dortmund, 2005.
[19] M. Chiarandini, L. Paquete, M. Preuss, E. Ridge, Experiments on metaheuristics: methodological overview and open issues, Technical Report DMF-2007-03-003, The Danish Mathematical Society, 2007.
[20] P. Suganthan, N. Hansen, J. Liang, K. Deb, Y.-P. Chen, A. Auger, S. Tiwari, Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization, Technical Report, Nanyang Technological University, 2005.
[21] S.K. Smit, A.E. Eiben, Beating the ‘world champion’ evolutionary algorithm via REVAC tuning, in: IEEE Congress on Evolutionary Computation, IEEE Computational Intelligence Society, IEEE Press, Barcelona, Spain, 2010, pp. 1–8.
[22] D. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.
[23] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, 3rd, extended ed., Springer-Verlag New York, Inc., New York, NY, USA, 1996.
[24] M. Preuss, Adaptability of algorithms for real-valued optimization, in: M. Giacobini, et al. (Eds.), Applications of Evolutionary Computing, EvoWorkshops 2009. Proceedings, in: Lecture Notes in Computer Science, vol. 5484, Springer, Berlin, 2009, pp. 665–674.
[25] S.K. Smit, A.E. Eiben, Comparing parameter tuning methods for evolutionary algorithms, in: IEEE Congress on Evolutionary Computation, IEEE Press, Trondheim, 2009, pp. 399–406.
[26] S.K. Smit, A.E. Eiben, Using entropy for parameter analysis of evolutionary algorithms, in: T. Bartz-Beielstein, M. Chiarandini, L. Paquete, M. Preuss (Eds.), Experimental Methods for the Analysis of Optimization Algorithms, in: Natural Computing Series, Springer, 2010, pp. 287–310.

[27] T. Bartz-Beielstein, C. Lasarczyk, M. Preuss, The sequential parameter optimization toolbox, in: T. Bartz-Beielstein, M. Chiarandini, L. Paquete, M. Preuss (Eds.), Empirical Methods for the Analysis of Optimization Algorithms, Springer, Berlin, Heidelberg, New York, 2009, pp. 337–360.

[28] A.E. Eiben, S.K. Smit, Evolutionary algorithm parameters and methods to tune them, in: E.M.Y. Hamadi, F. Saubion (Eds.), Autonomous Search, Springer, 2011.

[29] R. Myers, E.R. Hancock, Empirical modelling of genetic algorithms, Evolutionary Computation 9 (2001) 461–493.

[30] G. Taguchi, T. Yokoyama, Taguchi Methods: Design of Experiments, ASI Press, 1993.

[31] R.E. Bechhofer, C.W. Dunnett, D.M. Goldsman, M. Hartmann, A comparison of the performances of procedures for selecting the normal population having the largest mean when populations have a common unknown variance, Communications in Statistics B19 (1990) 971–1006.

[32] J. Greffenstette, Optimisation of control parameters for genetic algorithms, in: A. Sage (Ed.), IEEE Transactions on Systems, Man and Cybernetics, vol. 16 (1), IEEE Press, Piscataway, NJ, 1986, pp. 122–128.

[33] V. Nannen, A.E. Eiben, Relevance Estimation and Value Calibration of evolutionary algorithm parameters, in: M. M. Veloso (Ed.), Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI, Hyderabad, India, 2007, pp. 1034–1039.

[34] J. Schaffer, R. Caruana, L. Eshelman, R. Das, A study of control parameters affecting online performance of genetic algorithms for function optimization, in: Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989, pp. 51–60.

[35] O. Maron, A. Moore, The racing algorithm: model selection for lazy learners, in: Artificial Intelligence Review, vol. 11, Kluwer Academic Publishers, Norwell, MA, USA, 1997, pp. 193–225.

[36] T. Bartz-Beielstein, K. Parsopoulos, M. Vrahatis, Analysis of particle swarm optimization using computational statistics, in: Chalkis (Ed.), Proceedings of the International Conference of Numerical Analysis and Applied Mathematics, ICNAAM 2004, Wiley, 2004, pp. 34–37.

[37] T. Bartz-Beielstein, Experimental analysis of evolution strategies: overview and comprehensive introduction, Technical Report Reihe CI 157/03, SFB 531, Universität Dortmund, Dortmund, Germany, 2003.

[38] T. Bartz-Beielstein, M. Preuss, Considerations of budget allocation for sequential parameter optimization, SPO, in: L. Paquete, et al. (Eds.), Workshop on Empirical Methods for the Analysis of Algorithms, Proceedings, Online Proceedings, Reykjavik, Iceland, 2006, pp. 35–40.

[39] P. Balaprakash, M. Birattari, T. Stutzle, Improvement strategies for the F-race algorithm: sampling design and iterative refinement, in: T. Bartz-Beielstein, M. Blesa Aguilera, C. Blum, B. Naujoks, A. Roli, G. Rudolph, M. Sampels (Eds.), Hybrid Metaheuristics, in: Lecture Notes in Computer Science, vol. 4771, Springer, Berlin, Heidelberg, 2007, pp. 108–122.

[40] B. Adenso-Diaz, M. Laguna, Fine-tuning of algorithms using fractional experimental designs and local search, Operations Research 54 (2006) 99–114.

[41] M. El-Beltagy, P. Nair, A. Keane, Metamodeling techniques for evolutionary optimization of computationally expensive problems: promises and limitations, in: W. Banzhaf, J. Daida, A.E. Eiben, M. Garzon, V. Honavar, M. Jakiela, R. Smith (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-1999, Morgan Kaufmann, San Francisco, 1999, pp. 196–203.

[42] Y. Jin, A comprehensive survey of fitness approximation in evolutionary computation, Soft Computing 9 (2005) 3–12.

[43] A. Czarn, C. MacNish, K. Vijayan, B. Turlach, R. Gupta, Statistical exploratory analysis of genetic algorithms, IEEE Transactions on Evolutionary Computation 8 (2004) 405–421.

[44] I. Ramos, M. Goldbarg, E. Goldbarg, A. Neto, Logistic regression for parameter tuning on an evolutionary algorithm, in: Proceedings of the 2005 IEEE Congress on Evolutionary Computation IEEE Congress on Evolutionary Computation, vol. 2, IEEE Press, Edinburgh, UK, 2005, pp. 1061–1068.

[45] O. François, C. Lavergne, Design of evolutionary algorithms—a statistical perspective, IEEE Transactions on Evolutionary Computation 5 (2001) 129–148.

[46] S.P. Coy, B.L. Golden, G.C. Runger, E.A. Wasil, Using experimental design to find effective parameter settings for heuristics, Journal of Heuristics 7 (2001) 77–97.

[47] C.W.G. Lasarczyk, Genetische programmierung einer algorithmischen chemie, Ph.D. Thesis, Technische Universiteit Dortmund, 2007.

[48] D. Goldsman, B.L. Nelson, B. Schmeiser, Methods for selecting the best system, in: WSC'91: Proceedings of the 23rd Conference on Winter Simulation, IEEE Computer Society, Washington, DC, USA, 1991, pp. 177–186.

[49] B. Schmeiser, Simulation experiments, Handbooks in Operations Research and Management Science 2 (1990) 295–330.

[50] Y. Rinott, On two-stage selection procedures and related probability-inequalities, Communications in Statistics—Theory and Methods 7 (1978) 799–811.

[51] Y. Hochberg, A.C. Tamhane, Multiple Comparison Procedures, John Wiley & Sons, Inc., New York, NY, USA, 1987.

[52] S.-H. Kim, B.L. Nelson, A fully sequential procedure for indifference-zone selection in simulation, ACM Transactions on Modeling and Computer Simulation 11 (2001) 251–273.

[53] J. Branke, E. Chick, C. Schmidt, New developments in ranking and selection: an empirical comparison of the three main approaches, in: WSC'05: Proceedings of the 37th Conference on Winter Simulation, Winter Simulation Conference, 2005, pp. 708–717.

[54] R. Mercer, J. Sampson, Adaptive search using a reproductive metaplan, Kybernetes 7 (1978) 215–228.

[55] F. Hutter, H.H. Hoos, T. Stützle, Automatic algorithm configuration based on local search, in: Proc. of the Twenty-Second Conference on Artifical Intelligence, AAAI'07, pp. 1152–1157.

[56] F. Hutter, H.H. Hoos, K. Leyton-Brown, T. Stutzle, ParamILS: an automatic algorithm configuration framework, Journal of Artificial Intelligence Research 36 (2009) 267–306.

[57] V. Nannen, A.E. Eiben, A method for parameter calibration and relevance estimation in evolutionary algorithms, in: M. Keijzer (Ed.), Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'06, Morgan Kaufmann, San Francisco, 2006, pp. 183–190.

[58] S.K. Smit, A.E. Eiben, Parameter tuning of evolutionary algorithms: generalist vs. specialist, in: C. Di Chio, et al. (Eds.), Applications of Evolutionary Computation, in: Lecture Notes in Computer Science, vol. 6024, Springer, 2010, pp. 542–551.

[59] M. Pedersen, A. Chipperfield, Simplifying particle swarm optimization, Applied Soft Computing 10 (2010) 618–628.

[60] S.K. Smit, A.E. Eiben, Z. Szlávik, An MOEA-based method to tune EA parameters on multiple objective functions, in: J. Filipe, J. Kacprzyk (Eds.), Proceedings of the International Joint Conference on Computational Intelligence, IJCCI-2010, SciTePress, Valencia, Spain, 2010, pp. 261–268.

[61] J. Dréo, Using performance fronts for parameter setting of stochastic metaheuristics, in: GECCO'09: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference, ACM, New York, NY, USA, 2009, pp. 2197–2200.

[62] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2002) 182–197.

[63] E. Ridge, D. Kudenko, Sequential experiment designs for screening and tuning parameters of stochastic heuristics, in: Workshop on Empirical Methods for the Analysis of Algorithms, Reykjavik, Iceland, Online Proceedings, 2006, pp. 27–34.

[64] T. Bartz-Beielstein, S. Markon, Tuning search algorithms for real-world applications: a regression tree based approach, Technical Report of the Collaborative Research Centre 531 Computational Intelligence CI-172/04, University of Dortmund, 2004.

[65] B. Freisleben, M. Hartfelder, Optimization of genetic algorithms by genetic algorithms, in: R. Albrecht, C. Reeves, N. Steele (Eds.), Artificial Neural Networks and Genetic Algorithms, Springer, 1993, pp. 392–399.

[66] T. Bäck, Parallel optimization of evolutionary algorithms, in: Y. Davidor, H.-P. Schwefel, R. Männer (Eds.), Proceedings of the 3rd Conference on Parallel Problem Solving from Nature, in: Lecture Notes in Computer Science, vol. 866, Springer, Berlin, Heidelberg, New York, 1994, pp. 418–427.

[67] M. Meissner, M. Schmuker, G. Schneider, Optimized particle swarm optimization (OPSO) and its application to artificial neural network training, BMC Bioinformatics 7 (2006) 125.

[68] V. Nannen, A.E. Eiben, Efficient Relevance Estimation and Value Calibration of evolutionary algorithm parameters, in: IEEE Congress on Evolutionary Computation, IEEE, 2007, pp. 103–110.