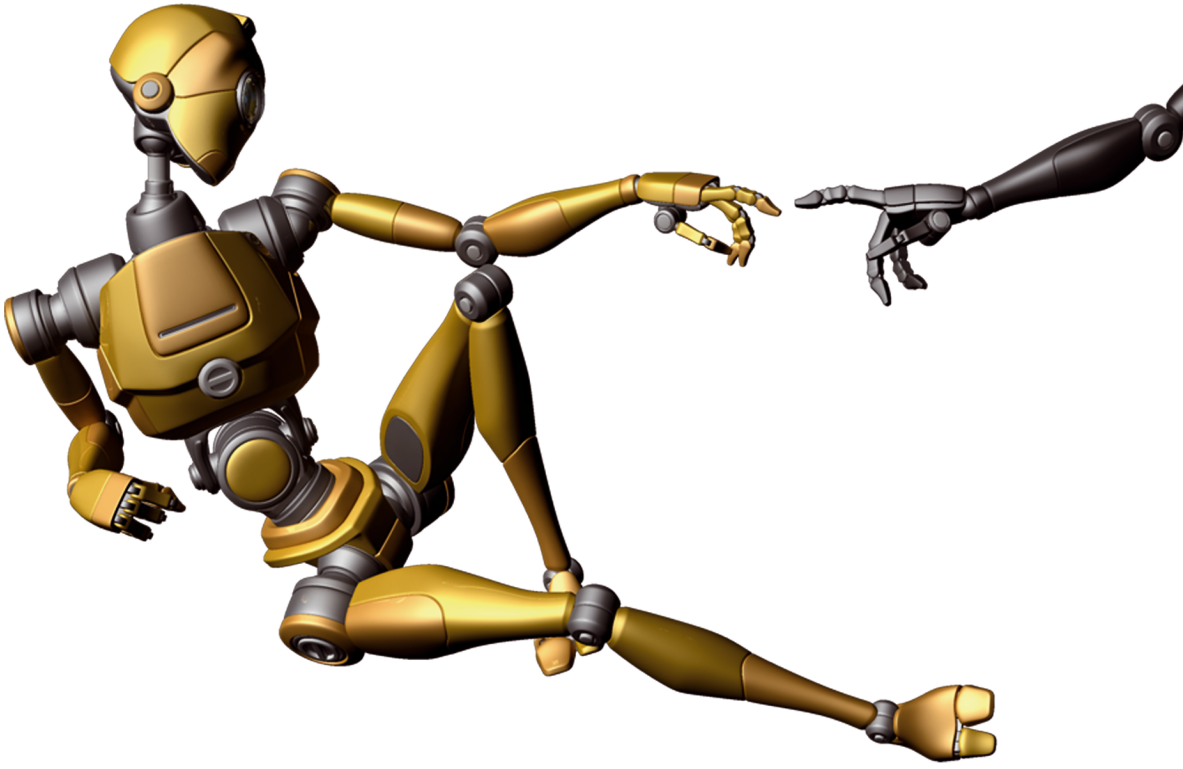
PARAMETER TUNING AND SCIENTIFIC TESTING IN EVOLUTIONARY ALGORITHMS



SELMAR KAGISO SMIT

VRIJE UNIVERSITEIT, AMSTERDAM

VRIJE UNIVERSITEIT

Parameter Tuning and Scientific Testing in Evolutionary Algorithms

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. L.M. Bouter,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de Faculteit der Exacte Wetenschappen
op woensdag 17 oktober 2012 om 15.45 uur
in de aula van de universiteit,
De Boelelaan 1105

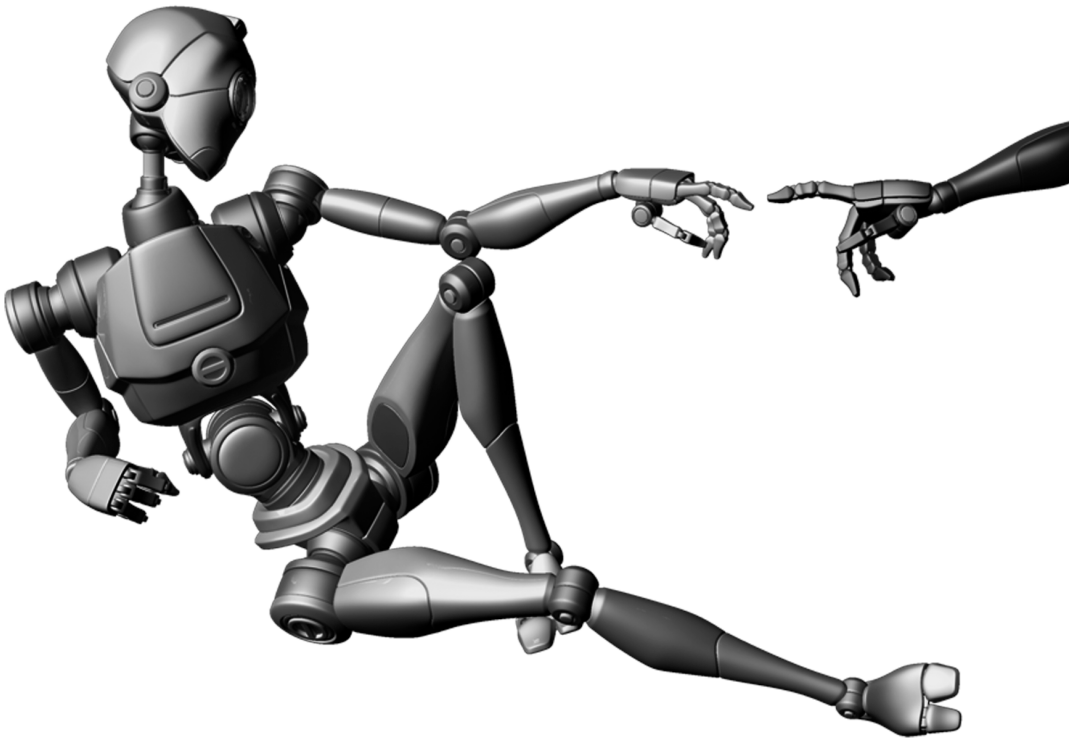
door

Selmar Kagiso Smit

geboren te Ramotswa, Botswana

promotor: prof.dr. A.E. Eiben

PARAMETER TUNING AND SCIENTIFIC TESTING IN EVOLUTIONARY ALGORITHMS



SELMAR KAGISO SMIT

VRIJE UNIVERSITEIT, AMSTERDAM

Cover illustrated by Mitra Smit

ISBN: 978-94-6191-420-0

SIKS Dissertation Series No. 2012-38
The research reported in this thesis has been carried out under the auspices of SIKS,
the Dutch Research School for Information and Knowledge Systems.

Dedicated to

life

"Rather to whom these things are rarities, that is the man who, when some unfamiliar thing is put before him, will take his fill of it with pleasure"

– Xenophon [144]

Contents

Preface

Even though only my name is on the cover of this thesis, many people contributed to it. I would like to use this first section to acknowledge these contributions.

First I would like to thank Gusz, not only for supervising this PhD project, but also for the inspiring discussions and all the things about research I learned from him. To think outside, and inside a box, in the sense that by somehow squeezing everything into a n-dimensional box(matrix), you will find what is up for grabs.

Secondly, I want to thank Evert for time we spend together, for our discussions and brainstorms, our coffee and our beers. Furthermore I want to thank him for his contributions to Section 3.3 of this thesis, that clearly showed how our two lines of research could come together.

I would also like to thank my predecessor Volker and successor Giorgos (Γιώργος) for their contributions to Section 2.2 and Section 4.3, respectively, in which they set out their visions on parameter tuning and parameter control. Furthermore, I am very grateful to Jan, for the effort he put into the "big tuner experiment", without him, my experiments would still not be finished.

Next to those co-workers, I would like to thank my other roomies; Robert, Joeri, Eelco and Berend, and the other members of the CI group for the pleasant time I had here at the VU. I would also like to thank my family; my brother for designing the cover, my sister and my parents for their love and support, especially at times when I was in need of formal mathematical proofs and properties.

Finally, but foremost, I would like to thank my lovely wife for her support during these four years. The number of times she had to listen to my new ideas, findings, frustrations or just mad brainwaves is countless. And when I was fed up with writing, she was there to encourage me. I love you.

SECTION 1

Introduction

1.1 Testing Heuristics, we have it all wrong

Imagine, one day you want to buy a horse, and not just any horse, but the best horse in the world. You want that single horse, that will make you win the gold medal at the London 2012 Olympic games. Hence, you search all newspapers and reports of the last years, and you quickly find out that this special horse is called Secretariat. Secretariat is one of only two horses that ever broke the two minute frontier in the Kentucky Derby, and is widely acknowledged to be the fastest horse in the history of horse racing. And, more importantly, he is for sale.

A few weeks pass, and together with the best horse, you have invested in the best trainers, the best jockey, the best stables and the best food, such that all parameters that might influence the performance of your horse are optimal. Nothing can get between you and the gold medal, you are sure of that.

Finally the day is there, the Olympic anthem is played, and the flag is hoisted into the air. Your horse is next, its name resounds through the stadium. And then ..., the music starts. "Dance of Devotion"; the same song Anky van Grunsven had, when she won her gold medal in 2008, and now its your turn.

Of course, you became last; you bought a race horse and let it compete in a dressage. An attempt that is not only utterly useless, but also shows a lack of knowledge about horses. Still, in the field of evolutionary algorithms, this happens all the time. A horse race is the common practice; algorithms compete on different kinds of benchmarks and

the one that reaches the finish line first is the winner. And, just as in the example, this winner is then used in a wide range of applications that have nothing to do with the original benchmark. In other words, the race horse is put in the dressage stadium. Even if this special race horse is capable of performing a gold-medal dressage, it is often the case that it was fully prepared for running, while a dressage would require a different trainer, a different jockey and possibly a different diet and horseshoes.

Therefore, there is no such thing as the 'best horse'. Which one is the best fully depends on what you intent to do with it, and how it is prepared. The same thing holds for evolutionary algorithms; letting them compete in a horse race and awarding the champion as being 'the best horse' is just doing it wrong.

1.2 Parameter Tuning and Scientific Testing in Evolutionary Algorithms

In this thesis we address this issue of evaluating and comparing evolutionary algorithms, and more specific, the role of parameter tuning in this context. The subject of this thesis is heavily inspired by the 1995 publication by J.N Hooker titled "Testing Heuristics: We Have It All Wrong"[71], in which Hooker criticizes the current paradigm of testing heuristics (such as evolutionary algorithms). He concludes that we have confused research with development, and focused on showing *that* our new algorithm performs better than the current state-of-the-art, rather than *why*. In this thesis we take the same stance, but focus on how parameter tuning can be used for a more "scientific approach" to testing heuristics.

To this end, we first give a general introduction of designing evolutionary algorithms in Chapter 1. There we address which questions arise and what decisions need to be made in such a process. Furthermore, we introduce the different terms related to algorithms, parameters and parameter tuning that are used throughout this thesis. We conclude this chapter by identifying two different approaches for comparing algorithms, namely competitive and scientific testing, and relate these to the issues that arise when designing algorithms. Most parts of this chapter originate from [42] and [40].

In Chapter 2, we give an extensive overview of the current approaches to automated parameter tuning, and we introduce three taxonomies by which these can be classified. We classify each of the approaches by the taxonomies and compare them based on theoretical arguments. The performance of the state-of-the-art approaches are then thoroughly tested by means of a big experiment, in which each of them needs to tune an advanced algorithm on a well-known test suite. We end this chapter with some

conclusions and recommendations about the use of the parameter tuners in different situations. Most parts of this chapter originate from [42], [105], [131], and [132].

Chapter 3 contains the results of three case studies that are conducted, each of which had a different goal. They are used to illustrate how parameter tuning can be used to improve the current practice of competitive testing for benchmarking, competitive testing for comparison, and the use of parameter tuning in scientific testing. Most parts of this chapter originate from [127], [129], [128], and [58].

In the fourth chapter, we discuss possible future use of parameter tuning methods in different areas. We show how it can be used for numerical optimization of noisy fitness landscapes, and the role that parameter tuning methods can have in parameter control. Most parts of this chapter originate from [42] and [85].

The thesis ends with a general discussion and conclusions about parameter tuning and scientific testing. It summarizes the thesis, and addresses the question that is fundamental to it: if we are all doing it wrong now, what is the correct way?

I

ALGORITHM DESIGN IN EXPERIMENTAL RESEARCH

"Countless hours are spent crafting the fastest possible code and finding the best possible parameter setting in order to obtain results that are suitable for publication"

– *J.N. Hooker*

Algorithm Design in Experimental Research

ABSTRACT

In this chapter we present a conceptual framework for parameter tuning based on a three-tier hierarchy consisting 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. Finally, we elaborate on the differences between competitive testing and scientific testing and discuss how both of them can be used to improve the common practice when comparing and testing evolutionary algorithms.

SECTION 1.1

Evolutionary Algorithm Design

1.1.1 Introduction

Evolutionary algorithms (EAs) form a class of heuristic search methods based on a particular algorithmic framework whose main components are variation operators (mutation and recombination) and selection operators (parent selection and survivor selection), cf. [43]. The general evolutionary algorithm framework is shown in Figure 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 following we use the term *parameters* to denote these details.

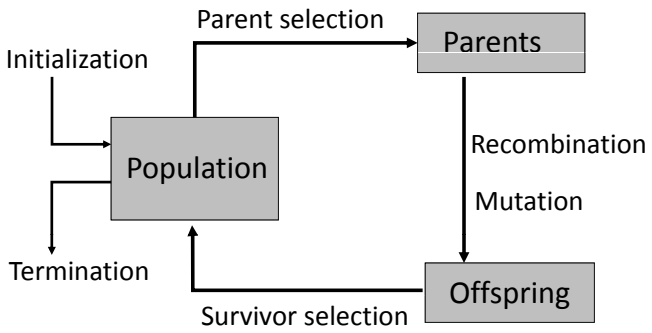


Figure 1: *General framework of an evolutionary algorithm.*

1.1.2 Evolutionary Algorithms and Parameters

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 $\mathbb{R}$ 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 (Table 1). For instance, [18] uses the names qualitative and quantitative parameters, [147] distinguishes symbolic and numeric parameters, in [21, 76] categorical and numerical is used, while [94] refers to structural and behavioral parameters. Furthermore, [105] 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

Table 1: *Pairs of terms used in the literature to distinguish two types of parameters (variables).*

Parameter with an unordered domain	**Parameter with an ordered domain**
qualitative	quantitative
symbolic	numeric
categorical	numerical
structural	behavioral
component	parameter
nominal	ordinal
categorical	ordered

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. In [76] such parameters are called 'conditional parameters'.

The distinction between qualitative and quantitative parameters naturally supports a distinction between algorithms and algorithm 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 method. Following this naming convention an *evolutionary algorithm* is a partially specified algorithm, fitting the framework shown in Figure 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 2 illustrates this matter by showing three EA instances belonging to just two EAs. Mind that, although here we focus on evolutionary algorithm, this is true for each algorithm with a deep parameter hierarchy.

This terminology enables precise formulations, 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,

	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

Table 2: *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 id different in its qualitative parameters.*

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.

Furthermore, we can distinguish so-called *structural tuning* and *(quantitative) parameter tuning* [20] in a formal way: structural tuning takes place in the space of qualitative parameter values. In contrast, the term parameter tuning, for which we commonly leave out the term 'quantitative', refers to searching through space of quantitative parameter values.

1.1.3 Problem Solving vs. Parameter Tuning

To obtain a detailed view on tuning, we distinguish three layers: application layer, algorithm layer, and design layer, see Figure 2. As this figure indicates, the whole scheme can be divided into two optimization problems that we refer to as problem solving and parameter tuning. The problem solving part consists of a problem at the application layer and an EA on the algorithm layer trying to find an optimal solution

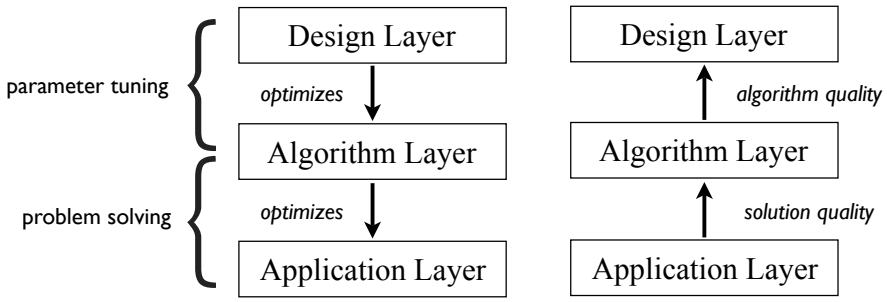


Figure 2: *Control flow (left) and information flow (right) through the three layers in the hierarchy of parameter tuning.*

	Problem solving	**Parameter tuning**
Method at work	evolutionary algorithm	tuning procedure
Search space	solution vectors	parameter vectors
Quality	fitness	utility
Assessment	evaluation	testing

Table 3: *Vocabulary distinguishing the main entities in the context of problem solving and parameter tuning.*

for this problem. Simply put, the EA is iteratively generating candidate solutions seeking one with maximal quality. The parameter tuning part contains a 'tuner' that is trying to find optimal parameter values for the EA on the algorithm layer. This could either be a human, configuring the algorithm by hand, or some kind of automated approach. Similarly to the problem solving part, this tuner is iteratively generating parameter vectors seeking one with maximal quality, where the quality of a given parameter vector is based on some kind of measure of the EA performance using the values of it. To avoid confusion we use distinct terms to designate the quality function of these optimization problems. Following 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 3 provides a quick overview of the related vocabulary.

With this nomenclature we can formalize the problem to be solved by the algorithm designer if we denote the qualitative parameters and their domains by $q_1, \ldots, q_m$ and $Q_1, \ldots, Q_m$, likewise using the notation $r_1, \ldots, r_n$ and $R_1, \ldots, R_n$ for the quantitative parameters.[1] The problem of parameter tuning can then be seen as a search problem

[1]Observe that by the possible presence of sub-parameters the number of quantitative parameters n depends on the instantiations of $q_1, \ldots q_m$. This makes the notation somewhat inaccurate, but we use

$\langle S, u \rangle$ in the parameter space

$$S = Q_1 \times \cdots \times Q_m \times R_1 \times \cdots \times R_n \tag{1.1.1}$$

using a utility function u, where the utility $u(\bar{p})$ of a given parameter vector $\bar{p} \in S$ reflects the performance of $EA(\bar{p})$, i.e., the evolutionary algorithm instance using the values of $\bar{p}$, on the given problem instance(s). Solutions of the parameter tuning problem can then be defined as parameter vectors with maximum utility. Given this conceptual framework, it is easy to distinguish the so-called "structural" and "parameter" tuning [20] in a formal way: structural tuning takes place in the space $S = Q_1 \times \cdots \times Q_m$, while parameter tuning refers to searching through $S = R_1 \times \cdots \times R_n$.

Now we can define the *utility landscape* as an abstract landscape where the locations are the parameter vectors of an EA and the elevation reflects utility. It is obvious that fitness landscapes –commonly used in EC– have a lot in common with utility landscapes as introduced here. To be specific, in both cases we have a search space (candidate solutions vs. parameter vectors), a quality measure (fitness vs. utility) that is conceptualized as a measure of height, and a method to assess the quality of a point in the search space (fitness evaluation vs. utility testing). Finally, we have a search method (an evolutionary algorithm vs. a tuning procedure) that is seeking for a point with maximum height.

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 quality 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. 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 metrics used to define the quality of EA, which in its turn depends on the preferences of the user. Namely, the definition of the quality of an algorithm is most likely different per person, and consist of two main ingredients: preferences about algorithm performance and algorithm robustness.

Summarizing, the core of our terminology is as follows:

1. Solution vectors have fitness values, based on the objective function related to the given problem instance to be solved.

2. EA instances have performance values, based on information regarding fitness and running time on one or more problem instances (objective functions).

it for sake of simplicity.

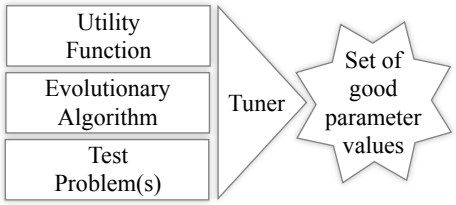


Figure 3: *Generic scheme of parameter tuning showing how good parameter values depend on four factors: the problem(s) to be solved, the EA used, the utility function, and the tuner itself.*

3. Parameter vectors have utility values, defined by some function that aggregates the performance values of the corresponding EA instance.

Figure 3 illustrates the matter in a graphical form. It shows that the solutions of a tuning problem 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.

SECTION 1.2

Algorithm Quality

1.2.1 Algorithm Performance

For researchers and practitioners, performance is probably the most commonly used quality indicator of an algorithm, setup, or set of parameter values. In most cases, one needs to find the set of parameter values for a particular algorithm on a particular (set of) problem(s) with the highest possible performance. In a business context this means, that 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. Adopting the terminology of Hooker [71], we refer to this as competitive testing, since it amounts to configuring some evolutionary algorithm driven by only a quality indicator. Simply put, competitive testing is solely aiming at obtaining an algorithmic design that meets some success criterion, for instance, beat some actual benchmark. How the corresponding parameter values are found, or the reason behind this choice of values is irrelevant in such a purely competitive approach.

This makes algorithm design, and parameter tuning easy. It only amounts to defining an appropriate quality indicator before the actual search for good configurations and parameter values is started. After that, by comparing the qualities of different settings one obtains a single design as output. Namely, the design that resulted in the best possible utility. This can then be reused to solve very similar problems it was tuned for, over and over again. One can for example, think of a delivery company that uses an evolutionary algorithm to schedule the delivery of packages throughout the country every day. Since each day a problem has to be solved, that is very similar to the

previous, such a company is solely interested in the algorithm setup that delivers the best possible result. Or, from an academic application, the setup that makes you win a competition or get your paper published.

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.

Then there are three main approaches in defining algorithm performance in one single run, namely:

- 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 three performance metrics commonly used in evolutionary computing, cf. [43, Chapter 14]:

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

respectively. As an example, the previously mentioned delivery company would be interested in the cheapest possible schedule that can be found within the small time-frame between collecting the orders, and the first car leaving the building. Hence, the design with the best mean best fitness (the costs of the schedule) is preferred. The second kind of approach is most commonly used in scientific applications. Here, the

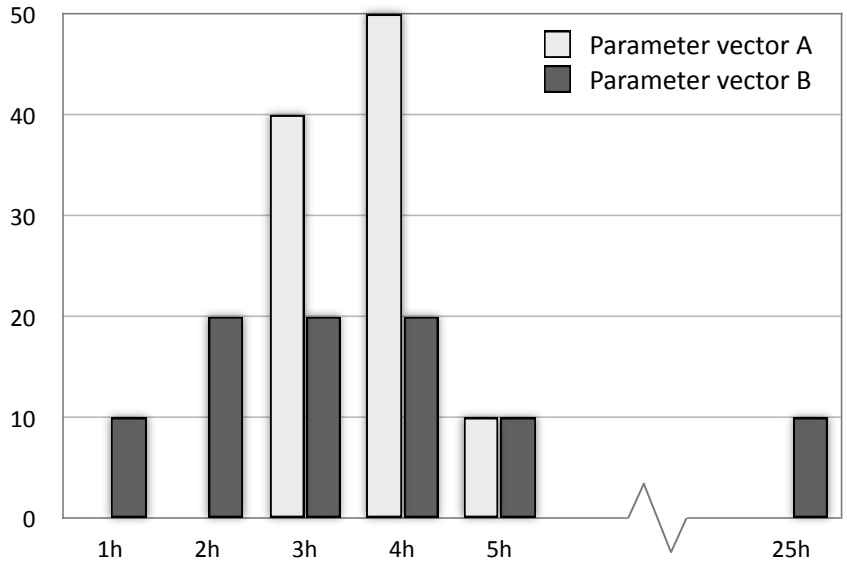


Figure 4: *Hypothetical results of the runtime distribution of an algorithm using two different parameter vectors. Y-axis shows the time needed to find a solution.*

interest is often in algorithm speed, rather than solution quality. The third one is again commonly used in industry, and, to be more precise, in reoccurring satisfaction problems. In terms of the delivery company, think of a problem in which there are many houses that need visiting, and products that have to be delivered that specific day. The main interest is now to find a schedule that satisfies all these constraints, rather than finding the fastest or cheapest one. Therefore, the design with the highest 'success rate' is preferred.

Obviously, the chosen performance metrics influences the shape of the utility landscape, and therefore the choice of the best design. However, straightforward as they are, these measures are not always appropriate. For example, in Figure 4 we have depicted hypothetical results for one algorithm on one application, but with two different parameter vectors.

It is immediately clear that both show completely different results, but which one of them is best is not so obvious. This depends on the preferences of the user, and the particular environment where the algorithm is used. Using the parameter vector A, the algorithm never reaches the level of speed that is reached using vector B. However, it always terminates within a reasonable amount of time, in contrast to vector B that sometimes requires more than a day to finish. Depending on the application, this could be acceptable or not.

The most prominent problem in this context 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[12]. Furthermore, in some applications the algorithm is not even required to perform good on average, but should have a high peak performance[38]. For instance, rather than a single run each day, the delivery company could run the algorithm 10 times, and only use the best found schedule. In this case, an algorithm setup with the highest expected maximum performance over 10 runs is preferred over a configuration with a high average. This demonstrates that one should be careful when adopting results that are optimized using a different quality indicator. Furthermore, it also shows that in order to reach the desired outcome, the performance metric has to reflect the specific preferences in order to select the 'best' EA instance for that particular application. In general, such a measure can be rarely defined as just an average, and often requires certain properties related to the robustness of the algorithm (instance).

1.2.2 Algorithm 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.

Table 4 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') as shown in Figure 5. In the following three subsections, we will describe each of the rows in more detail.

Despite the fact that in many real-world applications robustness is very important, it is in general, except for the success rate, less used as an objective of algorithm design than the previously mentioned performance measures that are simply averages.

ROBUSTNESS AS VARIANCE ACROSS	LARGE VALUE FOR WIDTH	HEIGHT
Problem instances	widely applicable	fallible
Parameter values	tolerant	tuneable
Random seeds	successful	unstable

Table 4: *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 and Figure 5 for more details.*

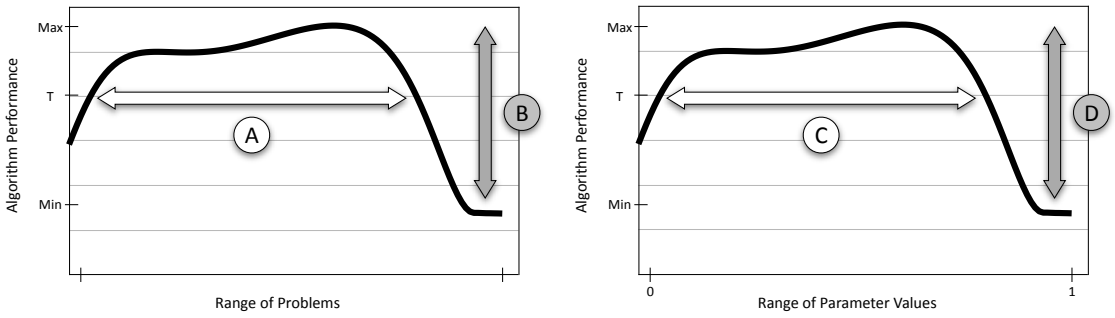


Figure 5: *Illustration of algorithm applicability (A), fallibility (B), tolerance (C), tuneability (D). See the text for detailed explanation.*

1.2.2.1 Robustness to Changes in the Problem Specification

In the simplest case, we are tuning an evolutionary algorithm A on one problem type f. Then the utility of a parameter vector $\bar{p}$ is measured by the performance of the EA instance $A(\bar{p})$ on different instances of f, or different seeds for the algorithm that is solving a single instance of f (Section 1.2.2.3). In this case, tuning delivers a *specialist*, that is, an EA instance that is very good in solving type f problems, with no claims or indications regarding its performance on other problems. This can be a satisfactory result if one is only interested in solving problems of type f. However, algorithm designers in general, and evolutionary computing experts in particular, are often interested in EA instances that work well on many, sometimes very different, objective functions. In such a case, tuning is performed on a test suite consisting of many test functions $f_1, \ldots, f_n$ and is aimed at finding a *generalist*.

The left diagram of Figure 5 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 [54, pg. 6], and Michalewicz, [99, pg. 292] refer to this kind of robustness.

Selecting designs based on their on applicability (width) means that one is looking for a design that makes the algorithm reach an acceptable performance on a wide range of problems. Although such an approach would deliver a true generalist, it is highly dependent on the choice of thresholds. Furthermore, one can imagine that a certain design solves 95% of the problems perfectly, but completely fails in solving the other 5%. Such a vector would have a worse average performance than a design that performs reasonably good on all problems. This might in some applications not be a desired property, hence manual inspection would be needed for selecting based on applicability.

A preference for a low fallibility (height) means that the difference between the performance on different problems has to be as small as possible. Obviously, the fact that solutions to different problems often have different scales of quality raises issue here, but also the lack of any minimal performance conditions requires a bit of attention. A design that always terminates with the worst possible value of 0 has a height of zero, and therefore the best possible fallibility. So it is clear that preferences about fallibility must coincide with applicability or other performance indicators.

Furthermore, it should be noticed that these notions of robustness are applicable to EA instances, and not to EAs. The reason is simple: to measure performance one needs to have a fully specified EA instance $A(\bar{p})$. This $A(\bar{p})$, and/or the parameter vector $\bar{p}$ are robust (fallible, widely applicable), rather than the EA itself. However, it can be argued that it is possible to extend this notion of robustness to EAs. To illustrate this, assume a diverse set of problem instances F and two evolutionary algorithms A and B. Furthermore, assume that A is able to solve, or reach a certain threshold T, all instances from F by just using two different instances $A(\bar{p}_1)$ and $A(\bar{p}_2)$, specified by the parameter vectors $\bar{p}_1$ and $\bar{p}_2$. Now, if B needs three of more instances (parameter vectors) for the same results F, then this makes B less robust in the parameter space given a certain set of problems. Since this measure aggregates over all possible parameter values, optimizing this quality requires a search through the qualitative parameters and is therefore applicable to EAs rather than instances.

1.2.2.2 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 all possible parameter vectors. The right diagram of Figure 5 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.[1] 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, on groups of parameters, and on the algorithm as a whole.

However, measuring robustness to changing parameters is not straightforward, since the parameter-space is often high dimensional. One approach is to measure the area of the parameter-space that leads to a certain minimal performance, or, in other words, the percentage of parameter vectors that lead to a certain minimal performance. Parameter entropy [130] is such a measure that can indicate the robustness of numerical parameters, qualitative parameters, and algorithms as a whole.

Algorithms with a high tolerant parameters are very useful for users that are willing to invest some, but limited, tuning effort. In many cases, a specific parameter needs different settings for each problem, therefore it will not be very widely applicable. However, it could still be that a large area of the parameter space leads to success, although this area is different for each specific problem. In such a case, investing just a little bit of time is enough to find this specific area with high performing parameter vectors. Especially if running the algorithm is computational expensive, such characteristics are desired. Tunable algorithms on the other hand, are extremely useful for users that can invest a huge amount of tuning. A large difference in performance, combined with a small applicability, means that such an algorithm can be specifically tailored to a problem.

[1] This notion is very similar to the one used by Preuss [112].

1.2.2.3 Robustness to Changes in Random Seeds

EAs are stochastic algorithms, as they rely on random choices in several steps. Therefore, all experimental investigations should be statistically sound. This requires either a number of independent repetitions of a run with the same setup with different random seeds, or (for example in case of a deterministic algorithm) a number of independent repetitions of a run using different instances of the same problem. Hereby we obtain information over the third kind of robustness, namely the fluctuations in the results of comparable runs. Because it is hard to define a meaningful ordering of random seeds or instances, we cannot use an equivalent of the diagrams of Figure 5 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 1.2.1 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 such as in Figure 4. However more commonly, height and width are combined in a single measure by means of the standard deviation.

Selecting algorithms, qualitative parameters or parameter vectors based on this kind of robustness is mainly interesting for those users who require certain specific properties of an algorithm during the actual use. For example, in most business settings more than enough time is available for fine-tuning the algorithm beforehand. In order to acquire the best possible design for a specific problem, robustness in the space of parameter values, or problems, is not required. However, it has to have a certain robustness in the sense that it delivers a good solution every single time it is run during production.

SECTION 1.3

Parameter Tuning and Experimental Research Methodology

1.3.1 Introduction

Parameter tuning, experimental research methodology and user-preferences form three closely bonded layers (Figure 6). User preferences are the lowest level, and form the base of the pyramid. There are thousands of different user preferences and corresponding quality indicators. Some of those are about performance, others about different kinds of robustness, but the largest portion is a combination of all of them. One could prefer a good performance on problem A, fast on solving B, or has a set of parameter values that are widely applicable, etc...

On top of these user preferences, the parameter tuning approaches reside. In the current EC practice, tuning adheres to select the parameter-values by conventions, ad hoc choices, or experimental comparisons on a limited scale. Such methods incorporate underlying assumptions about user-preferences and the influence of parameter values. For example: the algorithm should be fast, therefore `populationsize` should be low; let's test 10 different values for `mutationrate` and choose the one with the highest mean best fitness; or using $\mathtt{p}_c = 1$ makes the algorithm more widely applicable. Automated parameter tuning approaches (Section 2) are even more closely tied to the preferences layer. Typically, these methods iteratively search through the space of algorithm designs based on the specific predefined indicator that measures the quality of the design.

Finally, on top of the pyramid, the experimental research methodologies are located. While there are thousands of preferences, and a couple of of parameter tuners, only

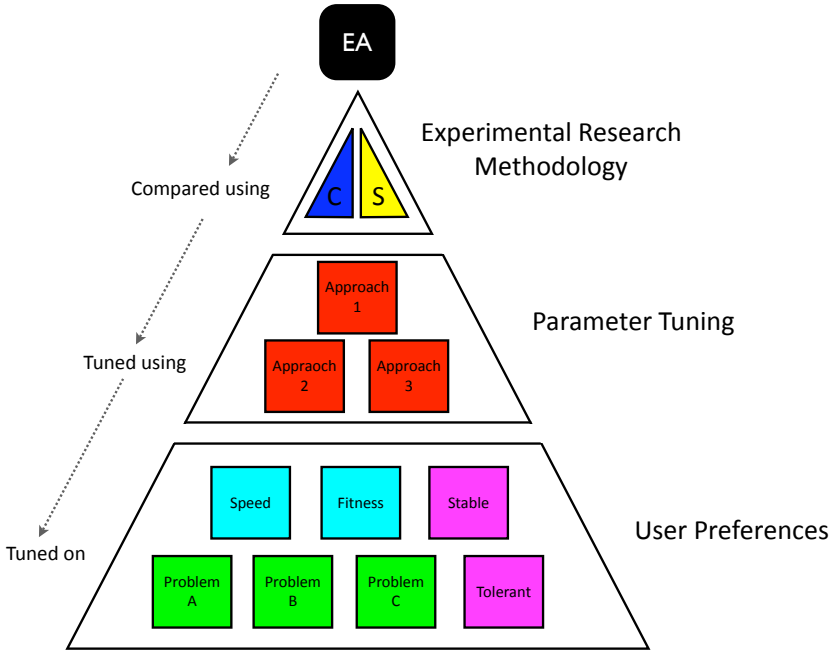


Figure 6: *The pyramid of experimental research methodology*

two main components form this layer: competitive testing and scientific testing. Since competitive testing amounts to selecting the algorithm or design with the highest score, a selection from the lower layers is required to define the 'contest'. The selected user preferences define the score-function used to compare different instances. As shown earlier, comparing EA instances based on speed is different from comparing them on best fitness. Furthermore, since competitive testing adheres to comparing EA instances rather than EAs themselves, one or more parameter tuning approaches need to be selected from the second layer in order to acquire an instance. Note that comparing instances always requires some form of parameter tuning, even if the 'default' parameters are used, one inherently chooses the method and preferences of the original author.

We adopt the term "scientific testing" from Hooker [71], as the second main approach in experimental research methodologies. Rather than simply selecting the best possible setup using some parameter tuning approach based on some indicator of user preferences, scientific testing amounts to describing how, why, and when a certain design, algorithm or set of parameter values works. It is more concerned with gaining insights into an algorithm through controlled experimentation. In general, such controlled experimentation should aim to study the effects of problem characteristics and algorithm characteristics on algorithm behavior (performance). In most cases, still a selection is needed from the lower layers. However, rather than using them as a

justification of the algorithm, they are used as an ingredient of a much wider study.

1.3.2 Competitive Testing and Parameter Tuning

To begin with, let us address concerns questioning the benefits of the whole tuning approach for real-world applications due to the computational overhead costs. For a detailed treatment it is helpful to distinguish two types of applications, one-off problems and repetitive problems, cf. [43, Chapter 14]. In case of one-off problems, one has to solve a given problem instance only once and typically the solution quality is much more important than the total computational effort. Optimizing the road network around a new airport is an example of such problems. In such cases it is a sound strategy to spend the available time on running the EA several times –with possibly long run times and different settings– and keep the best solution ever found without willing to infer anything about good parameters. After all, this problem instance will not be faced again, so information about the best EA parameters for solving it is not really relevant. It is in these cases, i.e., one-off problems, where the added value of tuning seems questionable. Applying some parameter control technique to adjust the parameter values on-the-fly, while solving the problem, looks like a better approach here. However, even such algorithms with controlled parameter need some initial settings and parameter-values related to the control method. In case of one-off problems, the obvious approach is then to use 'robust' values for them, that always work. But this requires information about the effect of parameters on the performance, given a certain problem, that might not be available. Therefore, spending effort on tuning and algorithm analysis by either the user, or the algorithm designer is inevitable.

In case of repetitive problems, one has to solve many different instances of a given problem, where it is assumed that the different instances are not too different. Think, for example, on routing vehicles of a parcel delivery service in a city. In this case it is beneficial to tune the EA to the given problem, because the computational overhead costs of tuning only occur once, while the advantages of using the tuned EA are enjoyed repeatedly. Given these considerations, it is interesting to note that academic research into heuristic algorithms is more akin to solving repetitive problems, than to one-off applications. This is especially true for research communities that use benchmark test suites and/or problem instance generators.

A typical use-case is the publication of a research paper showing that a newly invented EA (NI-EA) is better than some carefully chosen benchmark EA (BM-EA). In this case, tuning NI-EA can make the difference between comparable-and-sometimes-better and convincingly-better in favor of NI-EA. In this respect, using a tuner for

NI-EA and reporting the tuning efforts alongside the performance results is a bit more fair than the huge majority of publications at present, since it at least makes the tuning effort visible. A typical claim in a paper following the usual practice sounds like

> "NI-EA is better than BM-EA."

with possible refinements concerning the number or type of test functions where this holds. This means that from the whole pyramid, only the top is shown, namely, the competitive testing approach. How and why the parameter values are chosen remains unseen. Using tuners and reporting the tuning effort would improve the present practice by making things public and transparent. A typical claim of the new style would sound like

> "Spending effort X on tuning NI-EA on metric M, we obtain an instance that is better than the BM-EA."

To this end, the tuning effort could be measured by the number of parameter vectors tested (A), the number of utility tests executed ($A \times B$)[1], the computing time spent on running the tuner, etc. By reporting on the metric M that defines 'better' which is used for tuning, the whole pyramid of the NI-EA becomes visible.

Although an improvement over the current practice, this is only a fair comparison if the same choices are made for the BM-EA. Such an assumption is for example, fair in cases of benchmark competitions. One can assume that all competitors used the same problems, the same utility function and spend the maximum effort in finding the best design of their algorithm. In other words, they share the same base of the pyramid: the user preferences are equal, since those are defined in the competition rules, and the tuning approach is similar, namely 'maximum effort'.

However, just as often, a newly invented EA is compared to a benchmark EA on a new specific problem, using specific quality indicators to measure its performance. In the experimental setup of such work, one would often find terms like 'using the parameter values described in [1]', or 'with the default setup' describing the setup of the benchmark algorithm. On the other hand, the parameter values of the NI-EA are often carefully chosen to match the specific choices in the lowest layer of the pyramid. NI-EA therefore is hand-crafted to solve a specific problem very accurately, has a high success rate, or the described EA instance is widely applicable. It is obvious that such a comparison is rather unfair. If the choices of preferences are different, so will be the 'default parameters' and the corresponding performance (Figure 4).

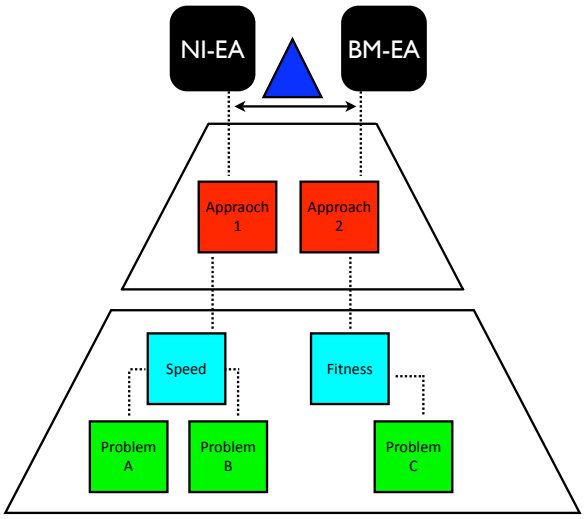


Figure 7: *Is this a fair competitive testing approach?*

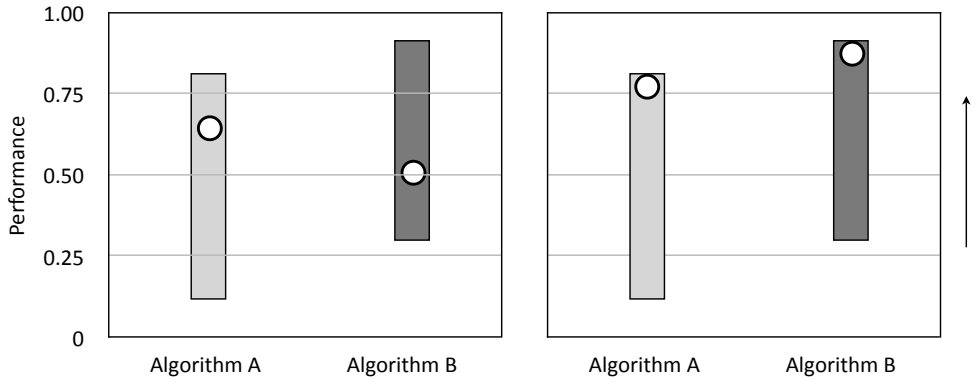


Figure 8: *The effect of parameter tuning on competitive testing. Left: the traditional situation, where the reported EA performance is an "accidental" point on the scale ranging between worst and best performance (as determined by the used parameter values). Right: the improved situation, where the reported EA performance is a near-optimal point on this scale, belonging to the tuned instance. This indicates the full potential of the given EA, i.e., how good it can be with using the right parameter values.*

This stance would imply that both NI-EA and BM-EA need equal tuning for a fair comparison. A typical claim of this style would sound like

"The best instance of NI-EA is better than the best instance of BM-EA,

[1]If the number of tests per parameter vector is not constant, then A × B is not the total number of utility tests executed, but for the present argument this is not relevant.

where for both EAs the best instance is the one obtained through spending effort X on tuning on metric M."

Note, that this new philosophy eliminates the rather accidental parameter values as a factor when comparing EAs and focuses on the type of EA operators (recombination, mutation, etc.) instead. In other words, we would obtain true, valid and fair statements about EAs, rather than specific EA instances.

1.3.3 Scientific Testing and Parameter Tuning

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. However, the only thing that is commonly known, is that on the specific problems, specific parameter values lead to a certain value for a specific performance measure as described in the publication that introduced that new algorithm. However, this EA user probably has different preferences about quality, especially since the problem on which it will be applied is different. But, little is known or reported about the effects of changing either the problems or performance metrics. Insights about the robustness of an algorithm are therefore particularly interesting to this kind of users.

We have adopted the term 'scientific testing' from [71] for such an approach that is aimed at gaining insights into the algorithm rather than optimizing on a quality indicator.

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 (or has a very high peak performance).

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 our definition of an EA. That is, since an EA is defined as having all qualitative parameters specified, 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 [105] 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.

Second, using parameter 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 learn about their effects and interactions, and thereby can lead to 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.

Figure 9 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" or density distribution.

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

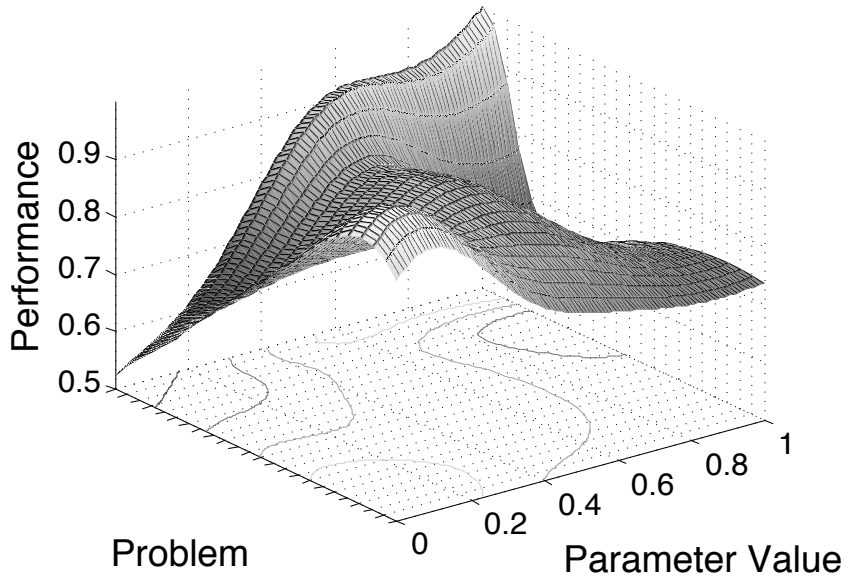


Figure 9: *Illustration of the grand utility landscape showing the performance (z) of EA instances belonging to a given parameter vector (x) on a given problem instance (y). Note: The "cloud" of repeated runs is not shown.*

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 1.2.2.3. Such data is 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)[72, 28].

The left-hand-side of Figure 10 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 1.2.2.2. Such data are often reported in the EC literature, be it with a different presentation, namely 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. In [72] the same is captured using visualizations of the solution cost distribution. Notice that the two EA instances depicted on left-hand-side of Figure 10 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 $T = 0.75$).

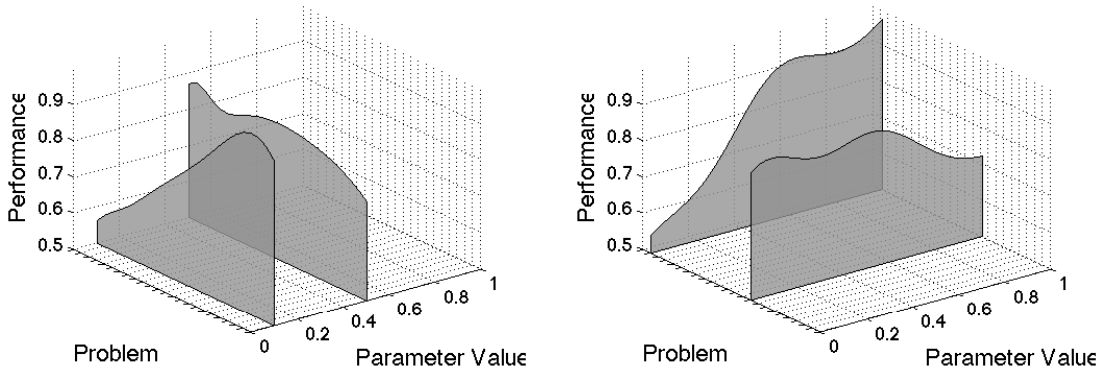


Figure 10: *Illustration of parameter-wise slices (left) and problem-wise slices (right) of the grand utility landscape shown in Figure 9. (The "cloud" of repeated runs is not shown.) See the text for explanation and interpretation.*

The right-hand-side of Figure 10 shows 2D slices corresponding to specific problem instances. On such a slice we see how the 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 1.2.2.1. 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 [15], aggregations such as boxplots and ECDFs. There are also visualizations that are specifically designed for analyzing utility landscapes such as entropy [130] plots that can be used to assess the tolerance and tuneability of an algorithm.

II

AUTOMATED APPROACHES TO PARAMETER TUNING

"We are all victims of a double-edged evolutionary process that favors a narrow selection of problems and algorithms"

– J.N. Hooker

Automated Approaches to Parameter Tuning

ABSTRACT

In this chapter we first establish different taxonomies to categorize tuning methods, and provide an extensive overview of the current literature. Next, we introduce two new tuning methods, namely REVAC and Bonesa, each with its own specialties and characteristics. Finally, we consider a set of the most modern tuners, including REVAC and Bonesa, and compare them on both the conceptual level, describing them in a uniform algorithmic framework, as well as experimentally, by using them to tune the same EA on a set of test functions. The results show clear differences in tuning performance (EA quality) and the amount of information gained through the tuning process.

SECTION 2.1

A Survey of Automated Tuning Methods

2.1.1 Introduction

The main objective of this part is to provide a thorough treatment of parameter tuning in evolutionary computing (EC). The treatment we have in mind has a twofold character. On the one hand, it is a survey of relevant approaches, algorithms and publications, including our own. On the other hand, we present the kind of horse-race, we criticized earlier. Rather than comparing algorithms based on their performance in the ultimate race for 'the best algorithm', we are comparing parameter tuners. We are well aware of this, and it indeed raises methodological issues. However, we tried to take a more neutral stance by emphasizing the good and bad qualities of each horse without firing the start-signal. Only at the end of this part, we will start the race in order to compare the usability of those tuners in a fight for the title 'best tuner for competitive testing', while emphasizing the need for scientific testing.

2.1.2 Positioning Tuning Algorithms

In this section we describe three different ways to distinguish and classify tuning methods. The first two are strongly focused on the internal algorithmic differences between tuners. In [41] we used the resulting taxonomies to structure a survey of tuning algorithms. In these two subsections we often refer to tuners as examples of a certain type of tuner, however for clarity we kept the descriptions of all tuners together in the third subsection. The overview in this subsection is organized differently, led

by a new taxonomy, taking into account the different notions of robustness, or, more generally, various aspects of scientific testing.

2.1.2.1 Tuning algorithms: taxonomy $NI/I - SS/MS$

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(NI) 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[102] and Taguchi Orthogonal Arrays [138] (Section 2.1.3.1).

The second category of tuners is formed by iterative(I) 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 2.1.3.4) and Iterative Sampling Methods (Section 2.1.3.1.1). Section 2.1.3 and Table 8 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 [19] and [18], we distinguish

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

Single-stage(SS) procedures perform the same number of tests for each given vector, while multi-stage(MS) 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.

2.1.2.2 Tuning algorithms: taxonomy $A/B/C$

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 [56] (Section 2.1.3.4) and the more recent REVAC method (Section 2.2). 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. Chapter 1.1. Formally, a meta-EA could work on parameter vectors that contain qualitative parameters too. But even then, its working would often boil down to random sampling in the (sub)space of qualitative parameters, due to the often unpredictable/unrelated results when changing qualitative parameter values.

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 [123] and racing [92] (Section 2.1.3.3). 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 on this idea, for example, Sequential Parameter Optimization (SPO) [10, 11, 17] (Section 2.1.3.2) and REVAC++ [127] (Section 2.1.3.4).

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. For the moment, there is only some work performed by Hutter et al. [78, 77], that extended their ParamILS and SMAC tuning methods (see Section 2.1.3) with adaptive censoring/capping. By terminating runs based on the time is was already running, they achieved substantial speedups up to a factor of 126. However, this is currently only limited to tuning on speed, and not for tuning on solution quality.

An overview of algorithms and their classification based on this taxonomy is shown in Table 8.

2.1.2.3 Tuning algorithms: taxonomy $MR/SA/SC/MB$

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 which obviously has a negative effect on the search for the best possible parameter-vector. Therefore, each of the tuning methods have a different balance between exploitation and exploration. Based on this distinction, we can identify four main approaches, namely an approach solely for finding the best performing vectors (meta-randomized search methods, MR), an approach mainly aimed at providing information (sampling, SA), and two approaches that are a combination of both (screening SC, and model-based MB).

Figure 12 shows the general outline of these four basic approaches. For clarification, we put the method that is used to generate new parameter-vectors in its own diagram

(Figure 13). Each block represents a group of parameter vectors, and their color shows the quality. These blocks are connected by arrows to indicate either a steam of vectors, or a stream of information (Figure 11). For meta-randomized search methods (Meta-RSM) for example, the main distinctive feature is obviously the recurrent loop from *Test* to *Generate*, indicating an iterative multi-stage approach.

Figure 13 is therefore also different, since this describes how the new parameter vectors are generated based upon the knowledge (population) in the current stage. Sampling and model-based approaches are very similar, however the output differs. Sampling methods deliver a 'winner', namely the parameter vector that performed best. Model-based methods on the other hand, deliver a model that describes the mapping from parameter-vector to expected performance. Although they can obviously also produce a winner, namely, the vector for which the best performance was measured.

Screening approaches are again a completely different, the recurrent loop is not between *Test* and *Generate*, but within *Test* itself. It shows that within screening methods, not all generated vector use the same number of tests, since only the best(green) vectors are tested again in the next stage.

Although each of the main approaches uses distinct techniques and have different goals, 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 , Section 2.2), while approaches from the informative type are improved for delivering high performing parameter-values (such as I/F-RACE [8] and CALIBRA [1]). Most of these methods even allow for a smooth transition between exploitation and exploration by defining suitable parameters values. As we will see later, most tuning methods are a composite

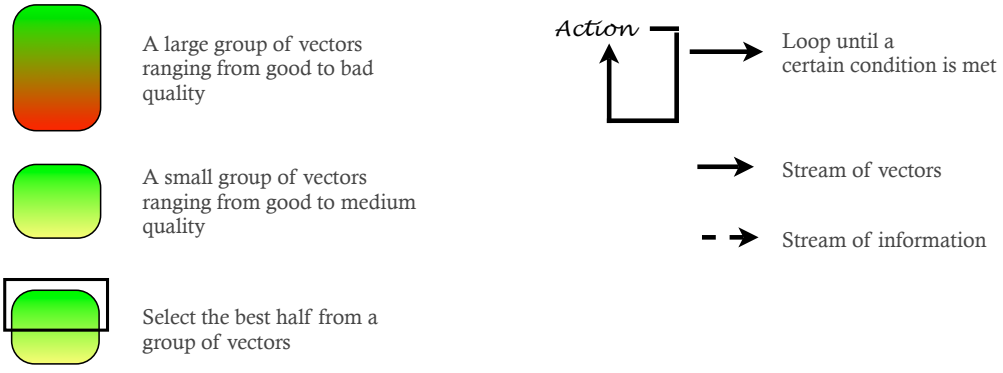


Figure 11: *Legend of tuner descriptions*

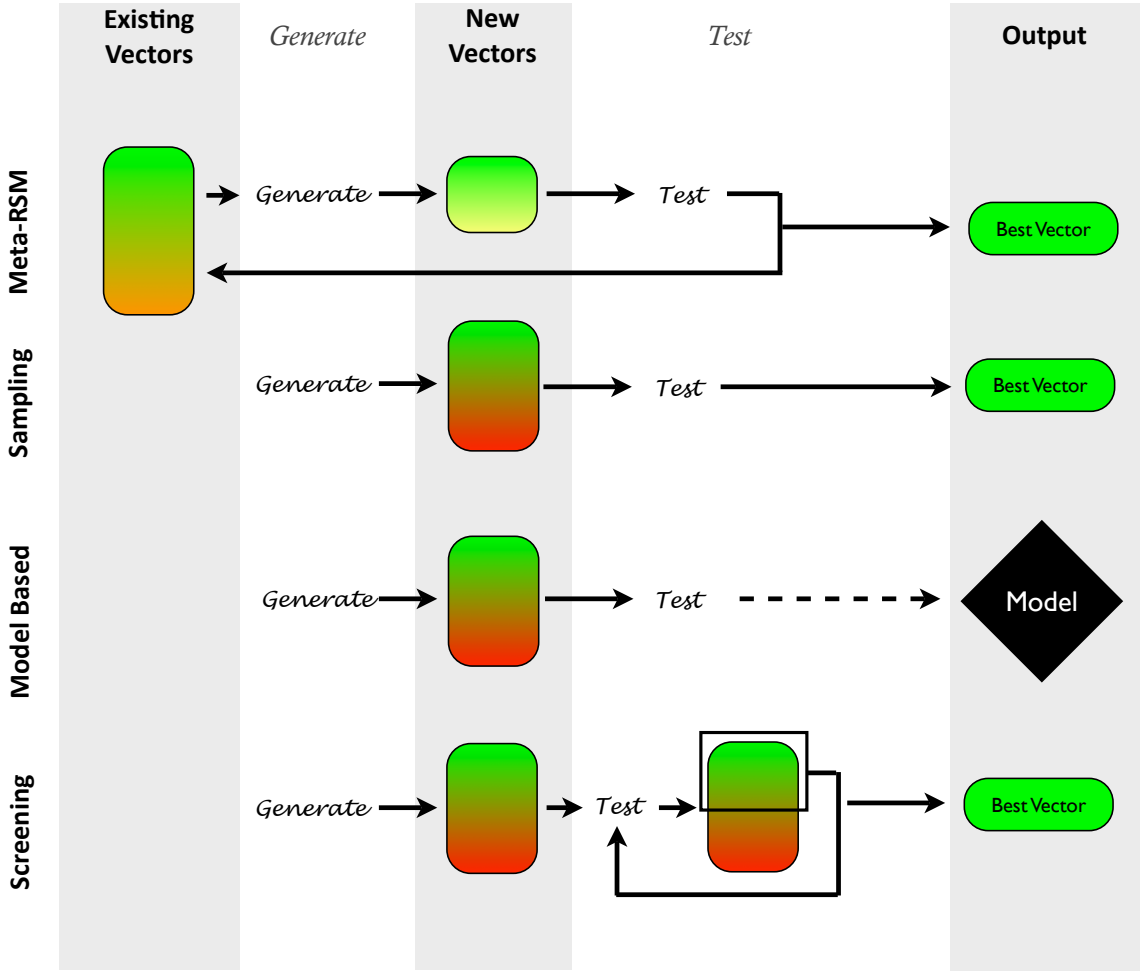


Figure 12: *The general outline of the four basic tuning approaches*

of the elements from these four basic approaches.

2.1.3 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 $MR/SA/SC/MB$. Hence, the four main categories are: sampling methods, model-based methods, screening methods, and meta-randomized search algorithms. Furthermore, the different branches are shown in which an approach from the main category is altered to specialize more on one specific task.

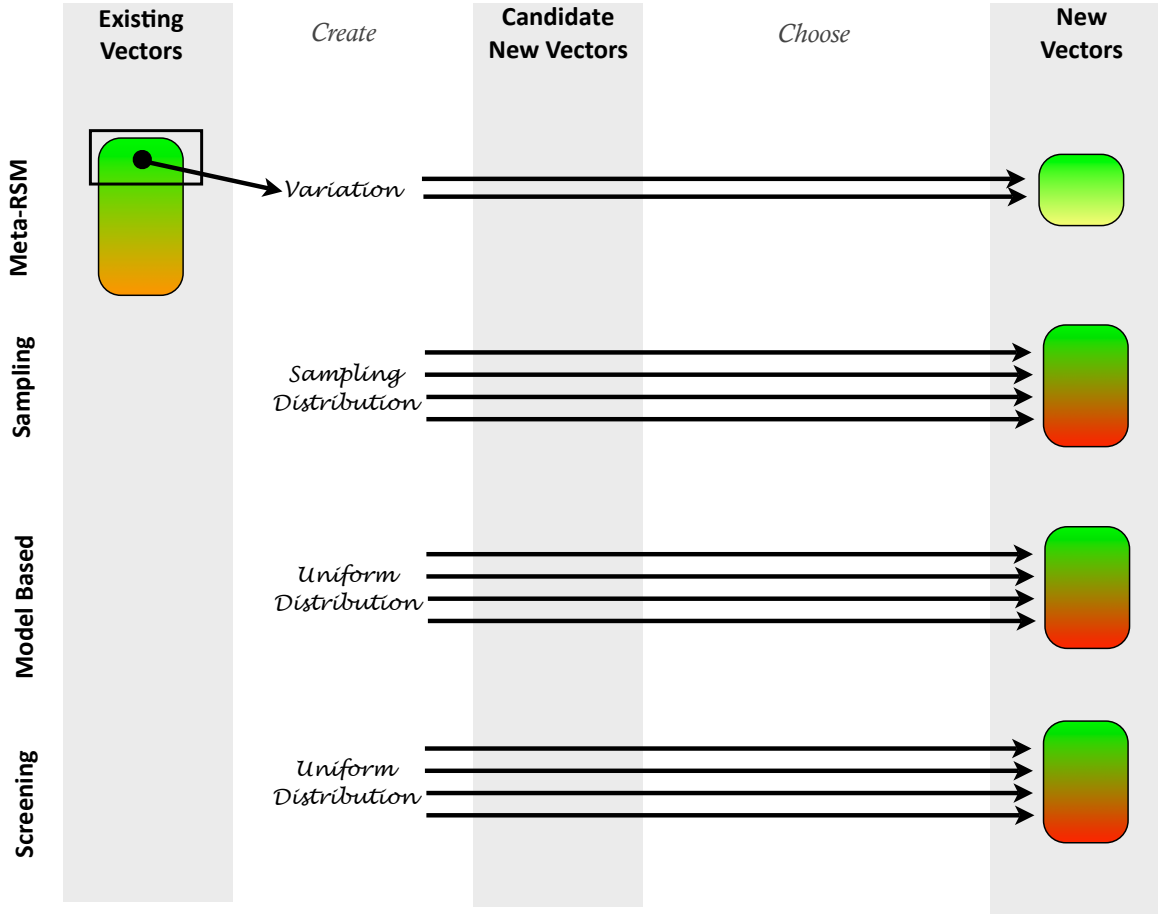


Figure 13: *The generate stage of the four basic tuning approaches*

2.1.3.1 Sampling Methods

Sampling methods can be described as methods that reduce the search effort by cutting down the number of parameter vectors tested (A) with respect to a full factorial design. The two most commonly used sampling methods are Latin-Square [102] and Taguchi Orthogonal Arrays [138]. 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.

2.1.3.1.1 Iterative Sampling Methods

CALIBRA [1] and Empirical Modelling of Genetic Algorithms [102] 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 [102] 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.

2.1.3.2 Model-Based Methods

Using meta models or surrogate models is a well established approach for the optimization of computationally expensive problems by (evolutionary) search [45, 80]. 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 [31, 115, 51]. The model is then a formula that maps parameter values to an predicted utility value $\hat{u}$. Equation 2.1.1 is an example of such a mapping for two parameters p_1, p_2.

$$\hat{u}(p_1, p_2) = \beta_0 + \beta_1 \cdot p_1 + \beta_2 \cdot p_1^2 + \beta_3 \cdot p_2 + \beta_4 \cdot p_2^2 \cdot \beta_5 \cdot p_1 \cdot p_2 \qquad (2.1.1)$$

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 for 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 these 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.

2.1.3.2.1 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 [30] 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) [10, 88] 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' performance, and these measured utility values are used to update the model for the next iteration. Furthermore, it also has a new feature, which we will call sharpening. If the procedure detects that the current number of tests per vector is inadequate to estimate it performance, it increases this number to sharpen the estimates.

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. An experimental investigation of the different types of models, and the methods to build and exploit the knowledge from these models can be found in [74].

2.1.3.3 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 [55]. 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) [124], Ranking and Selection (R&S) [120], Multiple Comparison Procedures (MPC) [69] and Fully sequential indifference-zone selection procedure(FSP) [86] are the four main approaches from this field [24]. 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 [24], 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 [55] and [24].

For the specific application of parameter-tuning, often the term "racing" is used, which was first described in [92]. 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 [92] (which uses Hoeffding's bound) and F-RACE [20] (which 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.

2.1.3.3.1 Iterative Screening Methods

Iterative F-RACE (I/F-RACE) [8] is an extension to F-RACE [20] with the specific goal of combing 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.

2.1.3.4 Meta Randomized Search Methods

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 randomized search methods (RSM) are very competitive heuristic solvers. Therefore, it is a natural idea to use such an approach to optimize the parameters of an evolutionary algorithm.

The idea of a meta-RSM was already introduced in 1978 by Mercer and Sampson [97] that created a meta-EA. But due to the large computational costs, their research was very limited. Greffenstette [56] conducted more extensive experiments with his Meta-GA and showed its effectiveness. In general, the individuals used in a meta-RSM 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 randomized search method can be used, 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.

2.1.3.4.1 Enhanced Randomized Search Methods

FocusedILS [76] is such a method, that adds a screening approach to an existing technique. It is an implementation of the ParamILS framework that describes a workflow very similar to a $(1+1)$ evolution strategy combined with local search. In the outer loop, it differs with respect to the variation operation, which only s random moves for perturbation, rather than applying a Gaussian perturbation to the whole vector, and the use of random restarts. However, it also has, as the name suggests, an iterated locale search procedure to fine-grain the search on the perturbed vector. Finally to turn ParamILS into an executable conguration procedure, 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 a screening approach. A similar enhancement is proposed in [147] 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.

In Section 2.2 we show how meta-EAs can not only be extended with screening, but also with model-based approaches, and the sharpening approach of SPO.

2.1.3.4.2 Multi-Problem Randomized Search Methods

Multi-problem tuning introduces a new dimension to the parameter-search problem, since a specific parameter vector needs to perform well on each problem (or at least on one of then, see Section 2.1.3.4.2). The benefit of multi-problem testing is that this allows for a better analysis of both applicability and fallibility. Doing multiple single-objective runs on each of the problems would not yield into the same results. A certain parameter can be sub-optimal for each of the problems, and would therefore not be found with a single-objective approach, but 'optimal' if you are in need of a parameter vector that "always works". The problem is to find this robust parameter-vector. Mind that there is an important difference between multi-instance and multi-problem tuning. For multi-instance tuning, a single parameter vector is often sufficient to solve all possible problem instances reasonably well (since they are very similar) and therefore it is sufficient to average the performances over multiple instances. In multi-problem tuning on the other hand, such a generalist is often non-existent, or has a much lower

overall performance. Hence, simple averaging of the performance over multiple problems does not always deliver the expected results and has to be done with care.

Sequential Model-based Algorithm Conguration (SMAC) [75] is an extension of FocusedILS that combines it with Model-Based techniques. It adopts a similar screening technique as FocusedILS, however, again introduces a different method of generating a new vector to challenge the currently known best. It exhibits a local search procedure that searches for the vectors with the best predicted value, based on a random forest model. Furthermore, it is specifically aimed at multi-problem tuning.

In SMAC, a user-defined cost function is introduced that aggregates the predicted performances and predicted variances over multiple problems, into a single value indicating the 'expected positive improvement'. This approach is therefore similar to that of Local Unimodal Sampling [109] in which the utility of a parameter vector is defined as the average utility over a range of problems. For further enhancements regarding multi-problem tuning, it incorporates information about specific features of those problems(or instances) into the model. The quality of such predictions is obviously highly correlated with the informativeness of the problem-features. Although such feature extraction has not been very successful previously, so called 'exploratory landscape analysis' recently gained more attention when it showed to be very successful on identifying classes of problems in the BBOB'09 testsuite [98].

In Section 2.2 we introduce a third approach similar to SMAC and Local Unimodal Sampling, that is used for one of the case studies in this thesis.

Although, a straightforward approach is to aggregate the performance on each problem into one value to optimize all of them (like in SMAC and Local Unimodal Sampling), in Section 3.1.3 we show 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 intrinsically 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 therefore introduces a (hidden) bias that is not intended by the user. This bias can be lowered by using the geometric mean, but not avoided completely.

Alternatives to a 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 problem type in the test suite can be regarded as one objective [1]. Tuning along this line of thought can leverage on existing knowledge regarding multi-objective RSMs. These create a Parameter Pareto Front

[1]Mind that instances of the same type can obviously be averaged, since the previously mentioned bias will not occur there

that can be used to evaluate robustness to changes in problem definition, as well as performance using multiple performance criteria [131].

The method presented by Dreo in [37] 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 [35]. Although in [37] only speed and accuracy are defined as objectives, it can be easily extended to tune for stability too. In [131], 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.

In Section 2.3 we introduce a third multi-problem approach that merges the model-based approach from SPO with the multi-objective and sharpening approach from M-FETA.

SECTION **2.2**

REVAC

2.2.1 Introduction

In this section, we introduce a new method for parameter tuning called REVAC. Technically, REVAC is a heuristic generate-and-test method that is iteratively adapting a set of parameter vectors of a given EA. It can therefore be classified as a multi-stage, iterative, meta-randomized search method. Testing a parameter vector is, as always, done by executing the EA with the given parameters and measuring the EA performance. EA performance can be defined by any appropriate performance measure, or combination of performance measures (such as their sum). Because of the stochastic nature of EAs, in general running multiple repetitions with the same parameter vector is advisable to obtain better estimates of its quality.

For a good understanding of the REVAC method it is helpful to distinguish two views on a given set of parameter vectors as shown in Table 5. Taking a *horizontal* view on the table, each row shows the name of a vector (first column), the k parameter values of this vector, and the utility u^i of this vector i (last column), defined as the

Table 5: *Two views on a table of parameter vectors.*

	$\mathcal{D}(x_1)$	$\cdots$	$\mathcal{D}(x_i)$	$\cdots$	$\mathcal{D}(x_k)$	Utility
$\vec{x}^1$	$\{x_1^1$	$\cdots$	x_i^1	$\cdots$	$x_k^1\}$	u^1
$\vdots$		$\ddots$				$\vdots$
$\vec{x}^n$	$\{x_1^n$	$\cdots$	x_i^n	$\cdots$	$x_k^n\}$	u^n
$\vdots$				$\ddots$		$\vdots$
$\vec{x}^m$	$\{x_1^m$	$\cdots$	x_i^m	$\cdots$	$x_k^m\}$	u^m

average performance over n runs of the EA in question. However, taking a *vertical* view on the table, the i^{th} column in the inner box shows m values from the domain of parameter i.

To understand how REVAC is generating these parameter vectors the horizontal view is more helpful. From this perspective, REVAC can be described as an evolutionary algorithm, in the style of EDAs [101], working on a population of m parameter vectors. This population is updated by selecting parent vectors, which are then recombined and mutated to produce one child vector that is then inserted into the population. The exact details are as follows.

- **Parent selection** is deterministic in REVAC, as the best n ($n < m$) vectors of the population, i.e., those with the highest utility, are selected to become the parents of the new child vector. For further discussion we denote the set of parents by $\{\vec{y}^1, \ldots, \vec{y}^n\} \subset \{\vec{x}^1, \ldots, \vec{x}^m\}$.

- **Recombination** is performed by a multi-parent crossover operator, uniform scanning. In general, this operator can be applied to any number of parent vectors and the ith value in the child $\langle c_1, \ldots, c_k \rangle$ is selected uniformly random from the $i$the values, $y_i^1, \ldots, y_i^n$, of the parents. Here, we create one child from the selected n parents.

- **Mutation**, applied to the offspring $\langle \vec{c}^1, \ldots, \vec{c}^k \rangle$ created by recombination, works independently on each parameter $i \in \{1, \ldots, k\}$ in two steps. First, a mutation interval $[a_i, b_i]$ is calculated, then a random value is chosen uniformly from this interval. The mutation interval for a given c_i is determined by all values $y_i^1, \ldots, y_i^n$ for this parameter in the selected parents as follows. First, the parental values are sorted in increasing order such that $y_i^1 \leq \cdots \leq y_i^n$. (Note, that for the sake of readability, we do not introduce new indices corresponding to this ordering.) Recall that the child $\langle c_1, \ldots, c_k \rangle$ is created by uniform scanning crossover, hence the value c_i comes from one of the parents. That is, $c_i = y_i^j$ for some $j \in \{1, \ldots, n\}$ and we can define the neighbors of c_i as follows. The first neighbors of c_i are y_i^{j-1} and y_i^{j+1}, the second neighbors are y_i^{j-2} and y_i^{j+2}, the third neighbors are y_i^{j-3} and y_i^{j+3}, etc. Now, the begin point a_i of the mutation interval is defined as the h-th lower neighbor of c_i, while the end point of the interval b_i is the h-th upper neighbor of c_i, where h is a parameter of the REVAC method (as there are no neighbors beyond the upper and lower limits of the domain, we extend it by mirroring the neighbors from the other side). The mutated value c_i' is drawn from this mutation interval $[a_i, b_i]$ with a uniform distribution and the child $\langle c_1', \ldots, c_k' \rangle$ is composed from these mutated values.

- **Survivor selection** is also deterministic in REVAC as the newly generated vector always replaces the oldest vector in the population.
- **Evaluation** The newly generated vector is tested by running the EA in question with the values it contains.

The above list describes one REVAC cycle that is iterated until the maximum number of vectors tested is reached.

By using these specific operators to generate the vectors in this table, it can be shown [104], that such a table represents the overall distribution of vectors that lead to a performance of at least u^m. In short: because above this 'line' there is no bias towards any area of the parameter-space, all values in the table are evenly spread. As a result they represent the overall distribution of parameter values that lead to a performance of at least u^m.

If, for example, all the values in column i are within the interval [0, 0.1], this means that of all possible vectors, only the vectors with a parameter value within this interval lead to a performance of at least u^m. The same holds, if there are twice as much values in the table, within the interval [0, 0.05] as within [0.05, 0.1]. Namely, this means that there are twice a much parameter vectors (from the set of all possible parameter vectors) with an i^{th} value within [0, 0.05] that lead to a performance of u^m than vectors with an i^{th} value within [0.05, 0.1].

2.2.2 Additional Components to REVAC

The different taxonomies of parameter tuners identifies that besides the principal approaches there are a number of useful 'add-ons', i.e., components that lead to a more hybrid version. These can either allow for to better competitive testing, i.e. improve the search capabilities by optimizing on $A \times B \times C$. For example by adding a screening component to BasicILS, we obtain FocussedILS that optimizes on B is much more efficient with respect to competitive testing. On the other hand, there are also components that allow for better scientific testing, i.e. they better aid algorithm analysis, such as multi-problem approaches.

Three of such components can also be applied to enhance REVAC:

- Racing
- Sharpening
- Multi-Problem utility aggregation

2.2.2.1 REVAC++: Racing and Sharpening

Racing was introduced into configuration selection by Maron and Moore [92]. The purpose of racing is to decrease the number of tests needed to estimate the quality of parameter vectors, and thereby the total runtime of a tuner. Hence, racing is a specific implementation of screening that acts as the selection phase of an iterative generate-and-test method.

The main idea is that the number of tests performed to estimate the utility of a parameter vector, n, is not used as a universal constant throughout the search, but as a variable maximum. Using racing we initially perform only a few tests for each vector, separate the ones that are clearly good, and continue with those vectors only that are not significantly worse or better than the good ones. Once we either obtain a set of significantly better vectors, equal to n, or we have reached the maximum number of repetitions n, the racing phase is terminated and the best n are used for recombination and mutation.

Sharpening was introduced as an separate technique for testing in [127], but was an intrinsic part of the SPO method by Bartz-Beielstein et al. [10]. The purpose of sharpening is to increase the quality of the estimation of the performance of all parameter vectors, as compared to the simple each-vector-n-tests approach, by increasing the number of tests per vector that are performed which transforms it into a each-vector-n_t-tests approach. However, this is only done if really needed, such that not to much of the tuning budged is spoiled.

The main idea is to start the tuning algorithm with a small number of tests per vector (which is lower than the n in a simple each-vector-n-tests approach), but when a certain threshold is reached, the amount of tests per vector is doubled (for all vectors). This means that the algorithm is initially able to explore the search space very quickly. If a promising area is found, the method focuses on improving the estimates by reducing the effect of possible outliers on the utility. Therefore, at the moment of termination, the current best vector is tested very often. This can lead to better results than algorithm that tests each vector only a couple of times.

Observe that racing and sharpening are opposing forces. Sharpening is increasing the number of tests during the tuning run for all vectors that need testing, while racing reduces the number of tests performed per individual vector. Nevertheless, they can be combined very easily. In a combined setup, sharpening will increase the maximum number of tests n that can be used by racing to select the best parameter vectors. In the beginning of the tuning-run the effect of racing will be very small, due to the small maximum 'budget' set by sharperning. However, during the run, when more and more

tests are required to sharpen the estimates, the role of racing will get more important. By using racing, not much effort is spent on vectors that are not very promising, even if sharpening already increased the number of tests. In principle, we can get the best of both worlds using this setup. By combining sharpening and racing much more effort is spent on promising vectors while the effort wasted on bad vectors is reduced.

2.2.2.2 Multi-Problem REVAC

In simplest case, the utility of a parameter vector $\bar{p}$ is the average performance of the EA using the values of $\bar{p}$ *on a given test function F, or set of instances representing a problem type F.* This notion is sufficient to find specialists for F. However, for generalists, a collection of functions $\{F_e1, \ldots, F_n\}$ should be used. This means that the utility is not a single number, but a vector of utilities corresponding to each of the test functions. Hence, finding a good generalist is a multi-objective problem, for which each test-function is one objective. One straightforward approach is to define the utility on a set $\{F_1, \ldots, F_n\}$ as the average of utilities on the functions F_i.

So in order to find the utility on a certain parameter set P, for each of the problems, a number a repetitions r, is executed. Let $F_{i,j}^{\bar{p}}$ be the utility of $\bar{p}$ on function number i and repetition number j, then the utility of $\bar{p}$ is:

$$F_{\bar{p}} = \sum_{j=1}^{r} \sum_{i \in P} \frac{F_{i,j}^{\bar{p}}}{|P| \cdot r}$$

However, REVAC (as many other tuners) can be ran with different aggregation functions, for example:

- MBF,
- SR,
- Rank (defined below),

For each of these utility functions it holds that $u(\vec{p})$ of a parameter vector $\vec{p}$ is calculated by running the algorithm r times independently on the whole test suite with the parameter values in $\vec{p}$. MBF and SR are commonly used ones, therefore we omit their definitions. Rank, however, is not that well known. Conceptually, rank is a lexicographic ordering such that the best parameter vector gets rank 1. Here we use it as an ordering based on SR (as primary measure) and MBF (secondary measure, in

case SR is equal). An important difference between MBF and SR on the one hand, and Rank on the other hand is that SR and MBF can be calculated for any parameter vector v in isolation, while Rank is a relative measure showing how good v is within a set of vectors V. Technically, given a set V of parameter vectors, the rank $R(v)$ of any $v \in V$ is calculated in two different ways, depending on its SR value:

- if $SR(v) > 0$, then $R(v)$ equals its SR-based rank in $\{u \in V : SR(u) > 0\}$
- if $SR(v) = 0$, then $R(v)$ equals its MBF-based rank in $\{u \in V : SR(u) = 0\}$ plus $|\{u \in V : SR(u) > 0\}|$

This definition allows us to calculate rank R for any set of parameter vectors. During a tuning session with REVAC, the set V is changing over time and at any time t it is the set of all parameter vectors that have been generated and tested till that time.

SECTION 2.3

Bonesa

2.3.1 Introduction

Bonesa is, in essence, an iterative model based search procedure much like Sequential Parameter Optimization [9], or in general, any search method using surrogate models [80, 45]. The generic scheme is illustrated in Figure 14 that shows an intertwined searching and learning procedure. The searching loop is a generate-and-test procedure that is iteratively generating new points in the search space and is assessing the quality of the most promising ones. The learning loop is using the information generated by the searching loop to build a model of the quality surface (here: utility landscape), which allows for estimating the quality of previously unseen points. Notice the two-way interaction between the two loops: information generated by the searching loop is used to update the model, while estimations based on the model are used to direct the search.

Such an iterative model based search procedure, in which a model is used to quickly pre-assess the generated points, can highly reduce the costs of the tuning session with respect to uniform sampling. Only parameter vectors that are, based on the model, expected to perform reasonably well are really tested. Furthermore, after running them, they are used to update the model. Therefore, if a vector from an area that looked promising beforehand turned out be be of low quality, the model is updated accordingly, and will therefore not be selected again. Hence, this approach does not only contribute to the reduction in costs, it also a natural way to derive a model of the grand performance landscape, since the procedure is iteratively increasing its accuracy.

However, the fact that the points that pass the model-based test need several expensive EA runs to establish their real performance still leads to high tuning-costs.

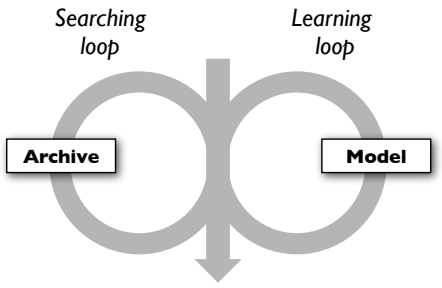


Figure 14: *The generic 'double loop' scheme of an intertwined searching and learning procedure. The searching loop is a generate-and-test procedure iteratively bulding an archive that contains all visited points and their quality. The learning loop is iteratively building a model of the quality surface, based on data in the archive.*

Therefore, we have chosen to use a Gaussian filter to reduce the effects of noise when assessing the performance of a parameter vector. A Gaussian filter is a common method of reducing noise in spatial data, which does not depend on repetitive testing of the same point but uses proximity data [66, 25]. Each parameter vector can be evaluated once rather than being retested, as long as the proximity data is relevant enough.

Furthermore, we adopt an implicit racing approach to determine which of two selected parameter vectors performs best [92]. The outcome of such a test is either 'A is better than B', or 'B is better than A', or that more data is needed to determine the order. Therefore, by iteratively increasing the number of data-points, only the minimal amount of tests are used, needed for comparing the two parameter-vectors.

Finally, the most distinguishing difference between the currently known iterative model based search procedures and Bonesa is its multi-objective approach. Rather than optimizing on a single problem or on a weighted sum of performance values [129, 108], Bonesa uses the Pareto-strength approach from SPEA2 [148] to optimize on a whole range of different problems in one go. Therefore, one is ultimately not only able to select the best parameter-values for a single problem, but also for a class of problems. It is important to note that this approach of multi-problem tuning differs from other multi-problems approaches [30, 76, 20] by means of the problems that are assessed. Rather than optimizing over a set of instances of the same type of problem, a true multi-objective approach allows for assessing instances of completely different problems. Since such a method terminates with a whole set of parameter-vectors, an experimenter can choose which of those suits his needs best. For example, certain parameter vectors could be well-suited for a subset of problems, rather than the whole set. Bonesa can identify these and deliver a (small) set of good parameter vectors, each with a distinct subset of covered problems. Such a set of vectors can support a much higher overall

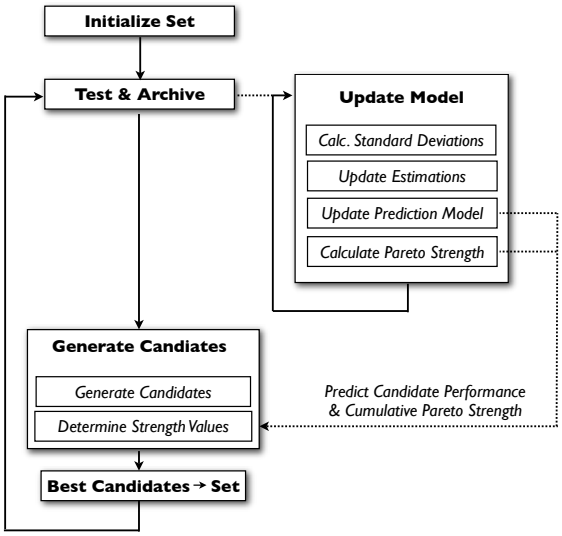


Figure 15: *The outline of the Bonesa algorithm*

performance than a single vector, delivered by usual tuners.

These four features form the core ingredients of Bonesa, in particular, of its learning loop. The searching loop on the other hand uses a straight-forward generate-and-test approach adopted from REVAC [104].

2.3.2 Learning Loop

The learning loop is iteratively updating the statistics and the prediction model of the utility landscape, using the set of currently known parameter vectors and corresponding performances. In each iteration, a new model is built that can be used to:

- Derive the relative distance between two parameter-vectors
- Predict the performance of an unknown parameter vector on each of the problems
- Estimate the cumulative Pareto strength of an unknown parameter vector

2.3.2.1 Relative Distance

In order to predict the utility values for unknown parameter vectors, a common approach is to model the utility landscape. There are two main approaches for modeling, surface

modeling and nearest-neighbor classification. The first one fits a model (a regression model, Kriging model, classification tree, etc.) to the data that can directly be used to predict the utility values. The second type estimates the utility of a new vector by a weighted average of the utilities of the closest known neighbors This method is very robust –for large neighborhood-sizes– even if measurements are very noisy. However, the problem with this approach is that there is no generic definition of being 'close to' a neighbor. In particular, it depends on the size of the sweet spot, i.e., the range where the good parameter values are. To illuminate this, let us consider two parameters, x and y, with the same range between 0 and 100. If the good values for parameter x are all between 50 and 100, then two values for x with a distance of 10 can be considered 'close'. However, if the good values for parameter y are all between 70 and 71, then two values for y with a distance of 10 should be considered 'far'. Therefore, we propose to use the standardized distance, which is obtained by dividing the distance between two points on a certain axis by the standard deviation of the parameter values of all 'good vectors' in the archive A. Here, 'good vectors' are defined as vectors that perform above average on at least one of the problems.

Definition 1 *Let GV be the set of l-dimensional 'good vectors' from the archive A and $\bar{\sigma}$ its standard deviation vector. Then the standardized distance between two l-dimensional vectors $\bar{x}$ and $\bar{y}$ is:*

$$r(\bar{x}, \bar{y}) \quad = \sqrt{1/l \cdot \sum_{i=1}^{l} \left((\bar{x}_i - \bar{y}_i)/\bar{\sigma}_i\right)^2} \tag{2.3.1}$$

2.3.2.2 Noise Reduction and Prediction

The quality of a specific parameter vector is defined by the performance of the EA on a collection of problems $P = \{p_1, \ldots, p_n\}$. By the stochastic nature of EAs, this performance is a noisy observable. In tuning terms, this means that the utility of a parameter vector $\bar{x}$ can only be estimated. The usual way of improving these estimates is to repeat the measurements [142, 73, 139, 47, 34], that is, to do multiple EA runs using $\bar{x}$, however, this is clearly an expensive way of gaining more confidence. Therefore, Bonesa tests each vector only once, and uses Gaussian filtering (or kernel smoother) to reduce the noise.

The main idea behind this technique is to perform just one run with each parameter vector $\bar{x}$ and to improve the confidence by looking at the utilities of similar parameter vectors in our archive assessed before. In fact, we heavily rely on a form of the strong causality principle, stating that small changes to a parameter vector cause only small changes in its utility. This property is true (or at least assumed) in the majority of

reproductive systems and can be popularized as "the apple does not fall far from its tree". In technical terms, this means that the utility of child vectors of $\bar{x}$ are likely to have a utility close to that of $\bar{x}$.

To this end, we use the concept of "perceived similarity", as proposed in [126]. The predicted utility $\hat{u}_{\bar{x},p}$ of a certain parameter vector $\bar{x} \in A$ on problem p is calculated by a weighted average of the test-results (u_p) of all vectors (including $\bar{x}$ itself) in the archive A on the same problem.

Definition 2 *The predicted utility $\hat{u}_{\bar{x},p}$ of a vector $\bar{x}$ on problem p is:*

$$\hat{u}_{\bar{x},p} = (\textstyle\sum_{\bar{y}\in A} u_{\bar{y},p} \cdot \omega_{\bar{x},\bar{y}})/(\sum_{\bar{y}\in A} \omega_{\bar{x},\bar{y}}) \quad , where \tag{2.3.2}$$

$$\omega_{\bar{x},\bar{y}} = e^{c \cdot r(\bar{x},\bar{y})} \qquad \forall\ \bar{y} \in A \tag{2.3.3}$$

The term c is a scaling factor, defined in such a way (Appendix A) that that the average value of $\sum_{\bar{y}} \omega_{\bar{x},\bar{y}}$ is equal to 50, if the $|A|$ points are random uniformly distributed in a l-dimensional hypercube. This means that the more points there are in the archive, the smaller the contribution is of points far from $\bar{x}$. Note that, since A increases over time, c also changes each time A is updated. The same approach of similarity based weighted average can be used to estimate the variance of the utility of $\bar{x}$ on problem p and the support (density) $\hat{\rho}$ of $\bar{x}$.

Definition 3 *The predicted variance $\hat{\sigma}^2_{\bar{x},p}$ of a vector $\bar{x} \in A$ on problem p is:*

$$\hat{\sigma}^2_{\bar{x},p} = (\textstyle\sum_{\bar{y}\in A} (u_{\bar{y},p} - \hat{u}_{\bar{x},p})^2 \omega_{\bar{x},\bar{y}})/((\sum_{\bar{y}\in A} \omega_{\bar{x},\bar{y}}) - 1) \tag{2.3.4}$$

Definition 4 *The support $\hat{\rho}$ of a vector $\bar{x} \in A$ is calculated using kernel density estimation:*

$$\hat{\rho}_{\bar{x}} = \textstyle\sum_{\bar{y}\in A} \omega_{\bar{x},\bar{y}} \tag{2.3.5}$$

Obviously, the same approach can be used to predict the utility, variance, and support of previously unseen vectors. Namely, the predicted utility $\hat{u}_{\bar{z}}$ of a certain seen or previously unseen vector $\bar{z}$ is derived using Gaussian interpolation over all vectors in the archive (Equation 2.3.3). The same can be done for the predicted variance (Equation 2.3.4) and support (Equation 2.3.5).

2.3.3 Searching Loop

As mentioned before, the task of the searching loop is to provide new vectors and performance values to the learning loop. The search loop of Bonesa is initialized with a uniformly randomly generated set of K parameter vectors. For each of them a single run of the EA to be tuned is executed on each problem from the test-suite, to determine its utility. Therefore, if the test-suite contains n problems, this yields into a total number of $K \cdot n$ tests executed in this step. After all tests are finished, the vectors are added to the archive A together with the utility values obtained by these tests. This first initial set is fed into the learning loop, and when that is finished, the iterative search procedure is started.

Each iteration starts with generating k candidate parameter-vectors. Half of them are generated using the REVAC approach, [104, 127]. To generate one new parameter vector, for each of the parameters ($i = 1, \ldots, l$) we have to randomly draw a value x_i from the corresponding parameter values in the archive. The second step is to generate a new parameter value, based on the one drawn from the archive. In Bonesa, this step is simplified w.r.t. the original REVAC scheme. Instead of calculating the so-called smoothening interval and drawing a random value from it, we simply add some Gaussian noise to x_i in such a way that the new value is expected to be within the top 50 closest values. To this end, the level of noise s_i has to be defined as follows.

$$s_i \;\; = \sqrt{12} \cdot \bar{\sigma}_i \cdot (\Gamma(0.5 \cdot l + 1) \cdot (50 \cdot 1/|A|))^{1/l} / \sqrt{\pi \cdot |A|} \tag{2.3.6}$$

where $\bar{\sigma}$ is the standard deviation vector of the archive A and Γ is the gamma function. This process is repeated until $0.5 \cdot k$ new parameter vectors are created, the other half is drawn uniformly random from the parameter space. All of the generated vectors are then fed into the model, and their quality is predicted based on their Pareto strength.

2.3.3.1 Pareto Strength

In general, the quality of a vector $\bar{z}$ (seen or unseen) in a multi-objective setting is based on the notion of Pareto dominance. In our case, parameter vectors must be compared based on their utility, which is an inherently noisy function. Therefore, we cannot simply compare the utility of $\bar{z}$ with that of $\bar{y}$ on a problem p. Instead we need to test the hypothesis that the average value of the utility distribution of $\bar{z}$ on problem p is higher than the average value of the utility distribution of $\bar{y}$ on problem

p. There are various known tests relying on samples from each distribution. However, Bonesa does not have such samples, because each parameter vector is only tested once. Therefore, Bonesa uses an adaptation of the Welch's T-test [143] to compare utilities:

$$\tau_{\bar{z},\bar{y},p} = (\hat{u}_{\bar{z},p} - \hat{u}_{\bar{y},p}) / \sqrt{\hat{\sigma}^2_{\bar{z},p}/\hat{\rho}_{\bar{z}} + \hat{\sigma}^2_{\bar{z},p}/\hat{\rho}_{\bar{z}}} \tag{2.3.7}$$

$$\nu_{\bar{z},\bar{y},p} = \frac{\left(\hat{\sigma}^2_{\bar{z},p}/\hat{\rho}_{\bar{z}} + \hat{\sigma}^2_{\bar{y},p}/\hat{\rho}_{\bar{y}}\right)}{\hat{\sigma}^4_{\bar{z},p}/(\hat{\rho}^2_{\bar{z}} \cdot (\hat{\rho}_{\bar{z}} - 1)) + \hat{\sigma}^4_{\bar{y},p}/(\hat{\rho}^2_{\bar{y}} \cdot (\hat{\rho}_{\bar{y}} - 1))} \tag{2.3.8}$$

$$D_{\bar{z},\bar{y},p} = \mathbb{T}\left(-\tau_{\bar{z},\bar{y},p}, \nu_{\bar{z},\bar{y},p}\right) \tag{2.3.9}$$

where $\mathbb{T}$ is the cumulative t-distribution. If the support for one of the two vectors is lower than 10, then this test can lead to unreliable results. Therefore we use the following equations to calculate the D given $\hat{\rho}_x > 10$ and $\hat{\rho}_y \leq 10$:

$$\tau_{\bar{z},\bar{y},p} = (\hat{u}_{\bar{z},p} - \hat{u}_{\bar{y},p}) / \sqrt{\hat{\sigma}^2_{\bar{z},p} \cdot 2} \tag{2.3.10}$$

$$D_{\bar{z},\bar{y},p} = \mathbb{T}\left(-\tau_{\bar{z},\bar{y},p}, 10\right) \tag{2.3.11}$$

If both of the two have a support lower than 10, then there is not enough evidence to decide which dominates the other, and no domination is assumed. In the other cases, $1 - D_{\bar{z},\bar{y},p}$ indicates the level of confidence in the hypothesis that average value of the utility distribution of $\bar{z}$ on problem p, is better or equal than the average value of the utility distribution of $\bar{y}$ on problem p. If this confidence is higher a certain threshold $1 - \epsilon$, we assume that $\bar{z}$ *is better than* $\bar{y}$ on problem p. The definition of dominance can therefore naturally be extended to:

Definition 5 *A parameter vector $\bar{z}$ dominates parameter vector $\bar{y}$ if and only if:*

1. $\exists p \in P \quad : \quad D_{\bar{z},\bar{y},p} \leq \epsilon$, *and*
2. $\forall g \quad (g \neq p) \in P \quad : \quad D_{\bar{y},\bar{z},g} > \epsilon$.

Using this definition of dominance we can define the number of 'slaves', the number of vectors from the archive that are dominated by a vector $\bar{z}$, as:

Definition 6 *The number of slaves $S(\bar{z})$, of any parameter vector $\bar{z}$ is equal to:*

$$PDp(\bar{z}, \bar{y}) = \begin{cases} 1 & \text{if } \bar{z} \text{ dominates } \bar{y} \\ 0 & \text{if } \bar{z} \text{ does not dominate } \bar{y} \end{cases} \tag{2.3.12}$$

$$S(\bar{z}) = \sum_{\bar{y} \in A} PDp(\bar{z}, \bar{y}) \tag{2.3.13}$$

Finally, we define the Pareto Strength $\hat{S}(\bar{z})$ [148] as -1 times the sum of the number slaves over all vectors that dominate $\bar{z}$ ('masters'):

Definition 7 *The Pareto Strength $Q(\bar{z})$, of a newly generated parameter vector $\bar{z}$ is:*

$$PDn(\bar{z}, \bar{y}) = \begin{cases} -1 \cdot S(\bar{y}) & \text{if } \bar{z} \text{ is dominated by } \bar{y} \\ 0 & \text{if } \bar{z} \text{ is not dominated by } \bar{y} \end{cases} \tag{2.3.14}$$

$$Q(\bar{z}) = \sum_{\bar{y} \in A} PDn(\bar{z}, \bar{y}) \tag{2.3.15}$$

This means that candidates which are not dominated by any of the vectors in the archive have a strength of 0, all others have a value lower than zero. The more vectors from the archive dominate the new candidate, the lower the strength. This makes the archive gradually move towards the Pareto-front. The Pareto strength can therefore be used to rank the candidates, much like [122], according to their position on the multi-problem performance landscape.

Based on this quality indicator, we can select the w vectors, from the set of generated vector, with the highest Pareto strength. In case of a tie, the candidate with the lowest support ($\hat{\rho}$) is preferred. This ensures a sampling bias towards areas with only a few vectors (low support), which in its turn, raises the support of the vectors in this area and therefore refines the model. Finally, the set of the w best vectors is tested on each of the problems and added to the archive. If the maximum number of tests is not yet reached, a new iteration is started. Observe that by increasing k or lowering w Bonesa can be made more explorative or exploitative.

Algorithm 1 shows how these components come together.

2.3.4 Validation experiments

For a rigorous evaluation of Bonesa and its derived models, we should apply it to a combination of an evolutionary algorithm and a test-suite is for which the complete per-

Algorithm 1: *The Bonesa algorithm.*

```
1  for i ← 1 to M do                                          // Initialisation
2      archive[i] ← CreateRandomParameterVectors();
3      archive[i].RawQualities[] ← RunAndEvaluate(archive[i], problems);
4  i ← M ;
5  while maximum budget not reached do        // Intertwined Searching and
   Learning
6      i ← i + 1;
7      archive.PredictedQualities[] ← KernelFilterMean(archive, problems);
8      archive.PredictedVariances[] ← KernelFilterVariance(archive, problems);
9      for j ← 1 to K do                                      // Generate Candidates
10         candidates[j] ← DrawFromArchiveDensity(archive) ;
11         candidates[j].PredictedQualities[] ← KernelFilterMean(candidates[j],
           problems);
12         candidates[j].PredictedVariances[] ← KernelFilterVariance(candidates[j],
           problems);
13         candidates[j].PredictedParetoRank ← CalculateParetoRank(candidates[j],
           archive);
14     Sort(candidates, candidates.PredictedParetoRank);
15     archive[i] ← candidates[1] ;
16     archive[i].RawQualities[] ← RunAndEvaluate(archive[i], problems);
17 for j ← 1 to i do                                       // Create termination set
18     archive[j].ParetoRank ← CalculateParetoRank(archive[j], archive) ;
19     if archive[j].ParetoRank == 0 then
20         terminationset ← { terminationset, archive[j] };
```

formance landscape is known. However, to our knowledge there is no such combination in evolutionary computing. So to be able to compare the models derived by Bonesa with the true utility landscape, we have created artificial performance landscapes in which we can control all aspects. We decided to take the mean best fitness performance measure (and not the EA speed, or whatever else) to be reflected by the artificial utility. Furthermore, for the sake of visualizing the results, we have chosen to limit the test-suite to only two artificial utility landscapes, based on two parameters, leading to three-dimensional robustness and four-dimensional utility landscapes. The performance $H^p(\bar{x})$ of a parameter vector $\bar{x}$ on an artificial utility-landscape p is defined as:

$$H^p(\bar{x}) = \sqrt{((\bar{x}_1 - B_1^p)/C_1^p)^2 + ((\bar{x}_1 - B_2^p)/C_2^p)^2}$$

$$\varepsilon(\bar{x}) = 0.05 \cdot log(4 + \bar{x}_1) \cdot (2 - \bar{x}_2)$$
$$U^p(\bar{x}) = e^{-1 \cdot Q^p(\bar{x})} + N(0, \varepsilon(\bar{x}))$$

The vectors B^p and C^p can be used to control the landscape. B^p indicates for which parameter-values the utility (mean best fitness) is optimal. C^p indicates how much the utility drops when moving away from the optimal vector. Finally, ε controls the amount of noise that we add to the outcome of each 'run'. This is to include the "cloud of repeated runs" (cf. Figure 10) in our artificial test data. In our two landscapes, we have chosen to make ε dependent on the parameter values of the 'run'. High values for parameter 2 yield stable results, high values for parameter 1 yield very noisy results. The values for B and C used in this experiment, are given in Table 6. The setup of Bonesa is given in Table 7.

Bonesa is incorporated in a Java application that specifically designed with user-friendliness in mind. The full sourcecode of both the algorithm and application is available from `http://tuning.sourceforge.net` and described in Appendix B.

2.3.4.1 Results

The main objective of Bonesa is to provide a cost-efficient method to accurately model the grand utility landscape. Since this landscape is a high dimensional surface, the accuracy is illustrated in Figure 16 using several slices. Each subfigure shows the known and predicted utility (EA performance) for a certain parameter value.

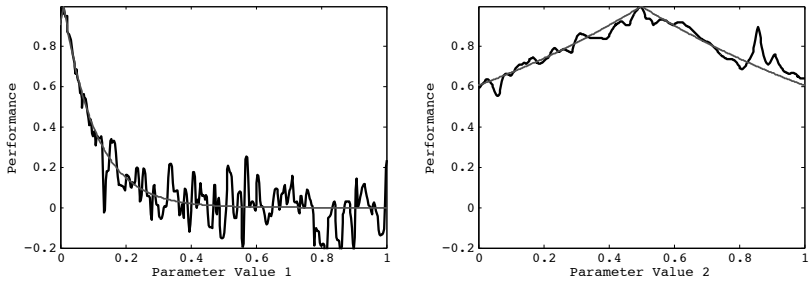
It is clear that the accuracy greatly differs between different areas of the grand performance landscape. The higher the performance, the better the accuracy as result

Table 6: *Setup of the artificial utility-landscapes*

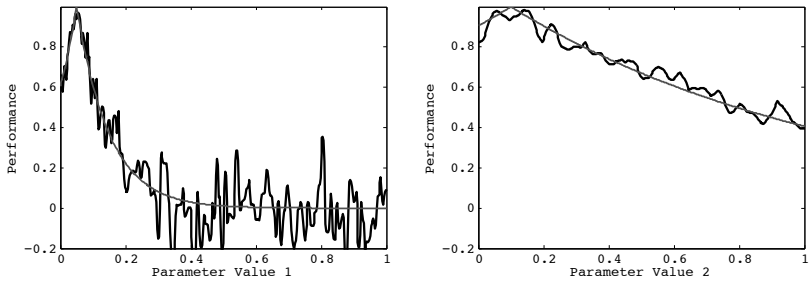
Parameter	Landscape 1	Landscape 2
B_1	0.01	0.05
B_2	0.50	0.10
C_1	0.10	0.10
C_2	1.00	1.00

Table 7: *Setup of Bonesa*

Parameter	Value
K	1000
k	1000
w	10
Dominance ϵ	0.05
Maximum number of tests	4000



(a) Parameter 1, Problem 1 (b) Parameter 2, Problem 1



(c) Parameter 1, Problem 2 (d) Parameter 2, Problem 2

Figure 16: *Real utility values (smooth curves) vs. utility values predicted by the Bonesa-made models (rugged curves).*

of the focus of Bonesa on 'promising points'. Figure c) shows an almost perfect fit, when predicting the performance for the 10% best parameter-values. Furthermore, it is clear that the tuneability of a parameter also influences the accuracy of the model. The predictions for parameter 1 are much more accurate near the optimum, than the predictions for parameter 2. So, the more important a correct parameter-value is, the better the accuracy of the prediction. Furthermore, since the grand performance landscape contains a "cloud" around the average performance, we also look at accuracy in predicting the variance in performance. These results are shown in Figure 17 for parameter one and problem one. All other graphs are very similar to this one, therefore omitted here. Figure 16 and 17 demonstrate that Bonesa is able to accurately model the grand performance landscape, especially in the 'interesting' areas.

To assess the efficiency of Bonesa, we inspect the space of all possible performance values for problem 1 and problem 2, a 2D square with values ranging from 0 to 1 on both axes, cf. Figure 18. On this square, the Pareto front is indicated by the grey line representing the optimal performance based on a certain combination of weights for both problems. The grayscale values of the rest of the plot show the computing time spent in a given area (black is much time, white is little time). The data exhibited by Figure 18 shows that Bonesa is able to identify and explore the interesting areas very

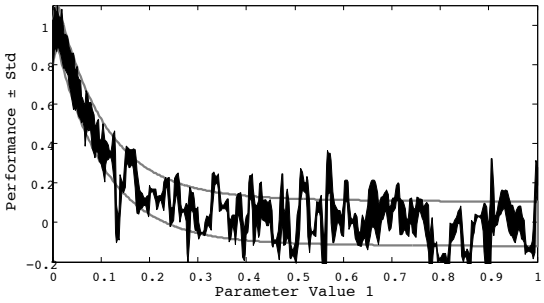


Figure 17: *Real noise shown by the real performance ± standard deviation (two smooth curves) vs. noise predicted by the Bonesa (rugged curves).*

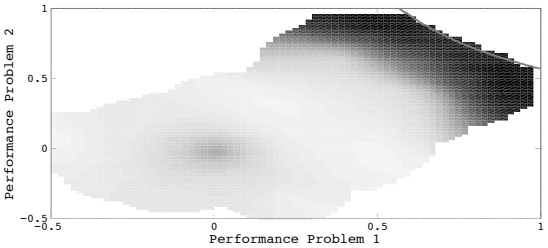


Figure 18: *Illustration of the search effort in relation to the predicted utilities*

effectively. 80% of the computing-time is spent in the area close to the Pareto front.

SECTION **2.4**

Comparing Tuning Algorithms

2.4.1 Introduction

Tuning algorithms can be compared on all different kinds of aspects, for example how they do tuning, how well do they do it, how efficient they are etc. One straightforward approach is to compare them based on our previous taxonomies $NI/I-SS/MS$ & $A/B/C$ & $MR/SA/SC/MB$ that classify how they approach the tuning problem. Table 8 shows this comparison for $NI/I-SS/MS$ & $A/B/C$. Most currently used methods are multi-stage methods, which makes sense since one if often only interested in a certain area of the parameter-space. For $A/B/C$ it is somewhat different, namely all modern (multi-stage) approaches optimize on A and some optimize on B too. Which type of approach (from $MR/SA/SC/MB$) does not really seem to influence this. Except for multi-stage sampling methods, all categories have at least one method that optimizes on both. But, that is a somewhat misleading conclusion, since an approach as Iterative F-RACE could also have been classified as a multi-stage sampling technique with B optimization. Only due to bonds with its ancestor (F-RACE) it has been classified differently.

Just as I/F-RACE originates from F-RACE, each tuner has its own history and origin. Some were designed by researchers from the field of system selection [24], others are influenced by design of experiments [10, 88], or are from the field of evolutionary computation itself [56]. However, eventually most of them are, in essence, a potpourri of the different components of the four basic approaches from $MR/SA/SC/MB$.

Figure 19 and Figure 20 show the general outline of the current state-of-the-art parameter tuning approaches and for historical commendation also the meta-EA of Greffenstette. It is based on the same presentation as previously used in Figure 12.

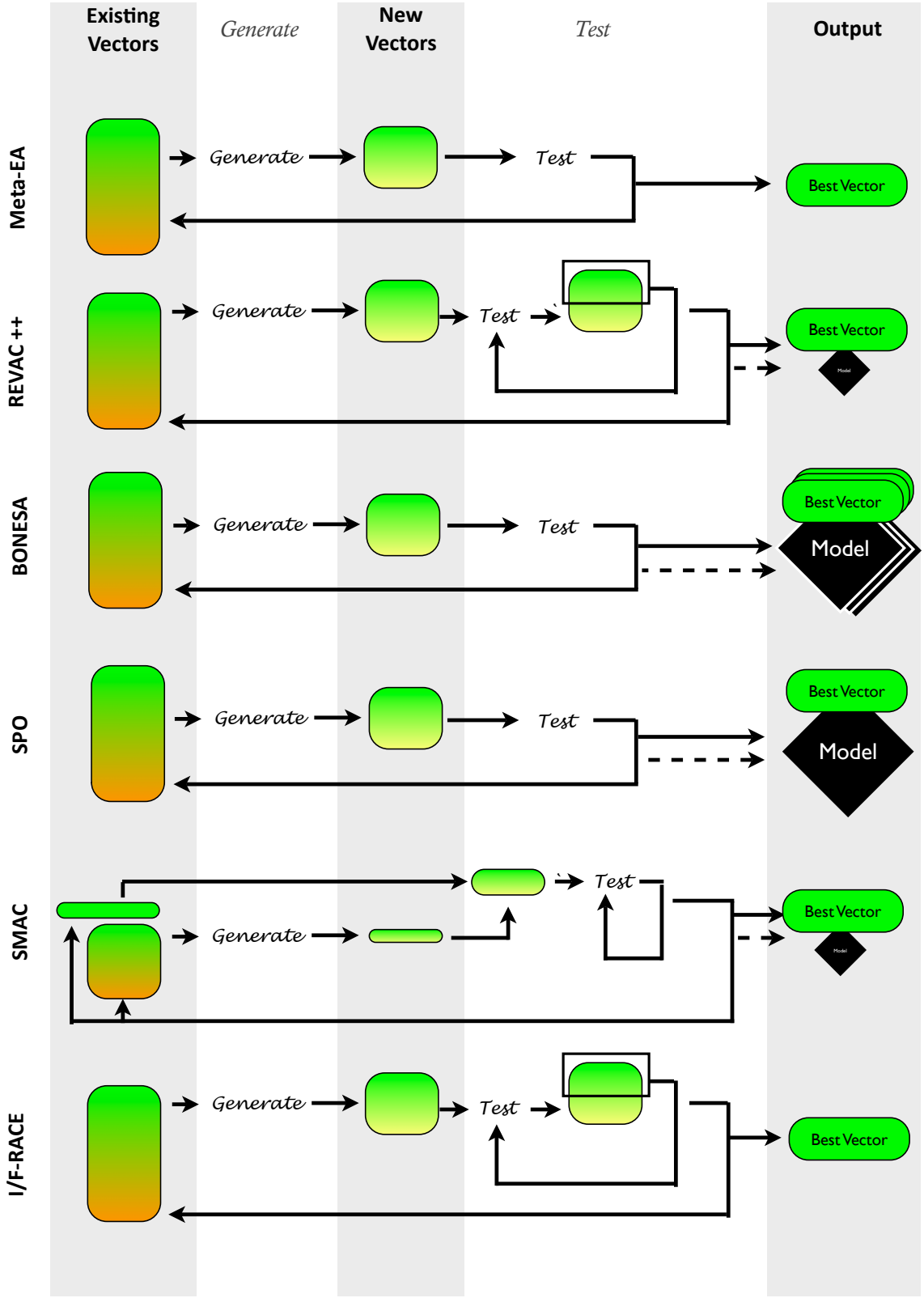


Figure 19: *The tuners loop*

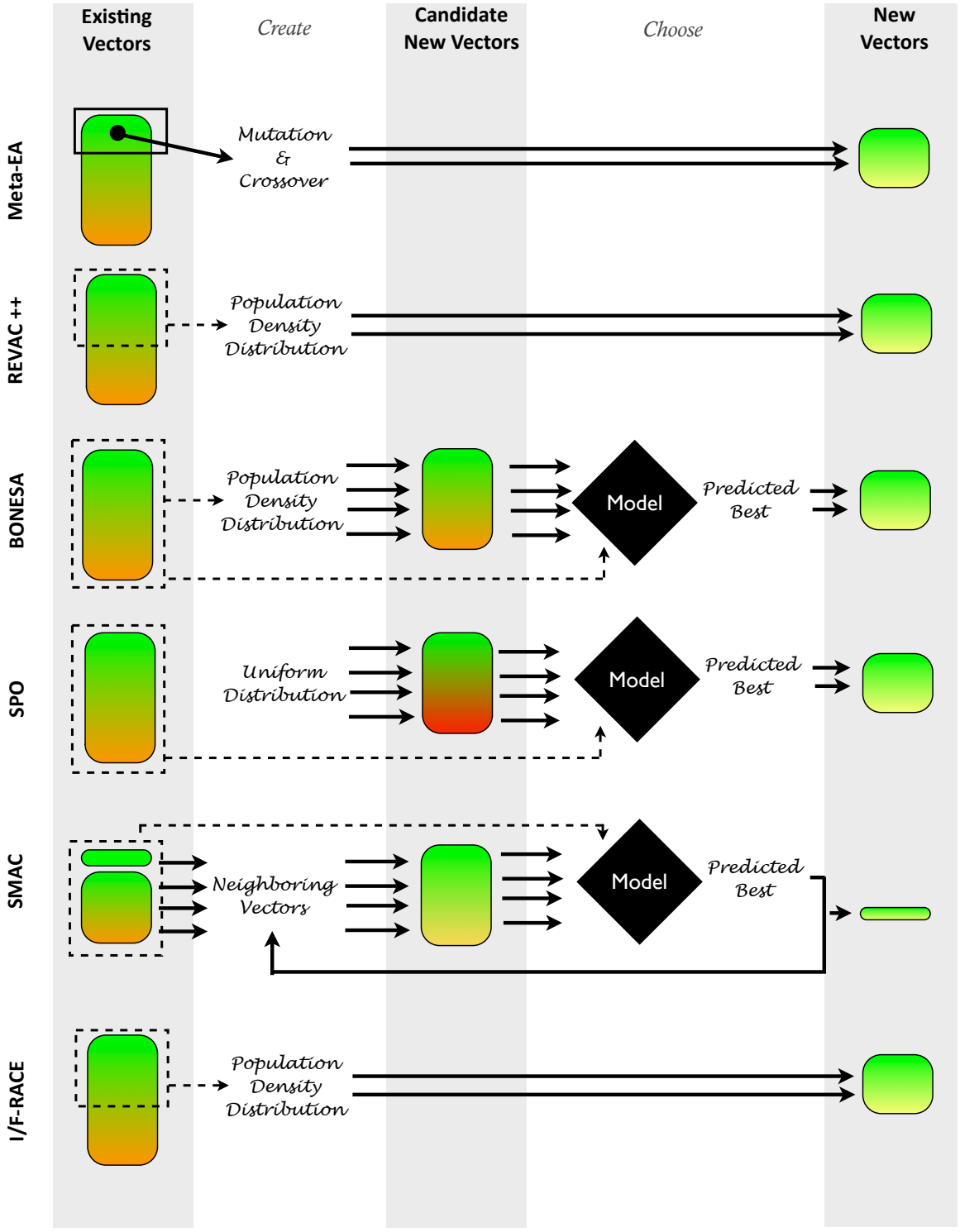


Figure 20: *The tuners generate stage*

Table 8: *Tuning methods feature comparison as described in taxonomy* $NI/I - SS/MS$ *&* $A/B/C$

Described in Section	Method	Taxonomy $NI/I - SS/MS$	Taxonomy $A/B/C$
2.1.3.1	Latin-Square [102]	Single Stage	A
	Taguchi Orthogonal Arrays [138]	Single Stage	A
2.1.3.1.1	CALIBRA [1]	Multi Stage	A
	Empirical Modelling of Genetic Algorithms [102]	Two Stage	A
2.1.3.2	Sequential Experiment Designs [118]	Single Stage	A
	François-Lavergne [51]	Single Stage	A
	Logistic Regression [115]	Single Stage	A
	ANOVA [123]	Single Stage	A
	Design of Experiments with Regression Tree [16]	Single Stage	A
2.1.3.2.1	Coy's Procedure [30]	Multi Stage	A
	BONESA [132]	Multi Stage	$A\&B$
	Sequential Parameter Optimization (SPO) [10]	Multi Stage	A
	SPO + OCBA [88]	Multi Stage	A
2.1.3.3	Interactive Analysis [124]	Single Stage	B
	Ranking and Selection [120]	Single Stage	B
	Multiple Comparison Procedures [69]	Single Stage	B
	Sequential indifference-zone selection [86]	Single Stage	B
	Racing [92]	Single Stage	B
	F-RACE [20]	Single Stage	B
2.1.3.3.1	Iterative F-RACE [8]	Multi Stage	$A\&B$
2.1.3.4	Meta-Plan [97]	Multi Stage	A
	Meta-Algorithm [56]	Multi Stage	A
	Meta-GA [52]	Multi Stage	A
	Meta-ES [6]	Multi Stage	A
	Meta-CMA-ES [127]	Multi Stage	A
	OPSO [96]	Multi Stage	A
2.1.3.4.1	FocusedILS [76]	Multi Stage	$A\&B$
	FocusedILS + Adaptive Capping [78]	Multi Stage	$A\&B\&C$
	Meta-GA + Racing [147]	Multi Stage	$A\&B$
	REVAC [103]	Multi Stage	A
	REVAC ++ [127]	Multi Stage	$A\&B$
2.1.3.4.2	Local Unimodal Sampling [109]	Multi Stage	A
	SMAC [75]	Multi Stage	$A\&B$
	SMAC+AC [77]	Multi Stage	$A\&B\&C$
	M-FETA [131]	Multi Stage	$A\&B$
	Performance Fronts [37]	Multi Stage	A

When comparing them to Figure 12 and Figure 13 their similarities are immediately clear. REVAC++ is essentially as combination of the meta-EA, a sampling approach, and screening techniques and is, on this higher level presentation, identical to I/F-RACE. SPO combines a meta-RSM with a model based approach, and BONESA extents this with REVAC components. Finally, SMAC shows much similarities to REVAC++ and I/F-RACE; however it is completely different with respect to generating new vectors. There it looks more like SPO and BONESA, but has an inner local-search loop.

Since each of them consists of these kind of 'building blocks', comparing them boils

down to evaluating what the effect is of the different components, on the performance of the algorithm and the information that can be gathered.

Measuring the information is rather straightforward, it is either absent, complete, or anything in between depending on the choice of components and their corresponding characteristics. However, from experience we know that it is hard to draw general conclusions about the performance of an algorithm based on its characteristics, and we do not expect tuners to be different. Hence, stating general conclusions is just as impossible as with comparing any set of 'ordinary' algorithms. On the other hand, it is still interesting to investigate how well they perform on a certain test set, even without any guarantees about the scope. Therefore, Section 2.4.3 shows such a comparison.

2.4.2 Information Comparison

Our comparison is summarized in Table 9 listing the parameter tuning approaches in this survey, showing to what extent they are able to:

1. Tune an EA for one or multiple objectives/problems at the same time
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.

Algorithms are rated varying from ++to - -, and are assigned based on the theoretical constraints imposed by techniques that are used. For example, algorithms that are only optimizing on B are generally of limited use to identify the best parameter-vector, as they depend on an initial static set of vectors. Although they often supply very accurate information about the whole parameter space, this mainly includes low-quality areas that might not be interesting. As another example, algorithms for optimizing A are score negative points in generating models and assessing robustness, because they, on their turn, are more aimed at finding the best parameter vector. This in contrast to model based methods (also when combined with optimizing on A) that are specifically suited for this. In case an algorithm optimizes both on A and B, then the grade is given based on the main focus of the approach.

Table 9: *Tuning methods feature comparison for scientific testing of robustness*

		Objectives	Seeds		Parameters		Problems	
Described in Section	**Method**	**Single/Multi**	**Success**	**Stability**	**Tolerance**	**Tuneability**	**Applicability**	**Fallibility**
2.1.3.1	Latin-Square [102]	multi	- -	□	- -	- -	- -	- -
	Taguchi Orthogonal Arrays [138]	multi	- -	□	- -	- -	- -	- -
2.1.3.1.1	CALIBRA [1]	single	□	□	+	+	- -	- -
	Empirical Modelling of Genetic Algorithms [102]	single	□	-	+	+	- -	- -
2.1.3.2	Sequential Experiment Designs [118]	multi	+	-	□	□	□	□
	François-Lavergne [51]	multi	+	-	□	□	□	□
	Logistic Regression [115]	multi	+	-	□	□	□	□
	ANOVA [123]	multi	+	-	□	□	□	□
	Design of Experiments with Regression Tree [16]	multi	+	-	□	□	□	□
2.1.3.2.1	Coy's Procedure [30]	single	□	-	□	□	□	□
	BONESA [132]	multi	++	+	□	□	++	+
	Sequential Parameter Optimization (SPO) [10]	single	+	-	++	++	+	+
	SPO + OCBA [88]	single	+	-	++	++	+	+
2.1.3.3	Interactive Analysis [124]	single	□	□	□	+	- -	- -
	Ranking and Selection [120]	single	□	□	□	+	- -	- -
	Multiple Comparison Procedures [69]	single	□	□	□	+	- -	- -
	Sequential indifference-zone selection [86]	single	□	□	□	+	- -	- -
	Racing [92]	single	□	□	□	+	- -	- -
	F-RACE [20]	single	□	□	□	+	- -	- -
2.1.3.3.1	Iterative F-RACE [8]	single	□	□	++	+	- -	- -
2.1.3.4	Meta-Plan [97]	single	□	-	- -	- -	- -	- -
	Meta-Algorithm [56]	single	□	-	- -	- -	- -	- -
	Meta-GA [52]	single	□	-	- -	- -	- -	- -
	Meta-ES [6]	single	□	-	- -	- -	- -	- -
	Meta-CMA-ES [127]	single	□	-	- -	- -	- -	- -
	OPSO [96]	single	□	-	- -	- -	- -	- -
2.1.3.4.1	Meta-GA + Racing [147]	single	-	-	- -	- -	- -	- -
	FocusedILS [76]	single	-	-	- -	- -	- -	- -
	FocusedILS + Adaptive Capping [78]	single	- -	-	- -	- -	-	-
	REVAC [103]	single	□	-	++	+	- -	- -
	REVAC ++ [127]	single	□	□	++	+	□	□
2.1.3.4.2	Local Unimodal Sampling [109]	single	□	-	- -	- -	□	□
	SMAC [75]	single	□	-	- -	- -	□	- -to ++
	SMAC+AC [77]	single	- -	-	- -	- -	□	- -to ++
	M-FETA [131]	multi	+	+	□	□	++	++
	Performance Fronts [37]	multi	+	+	-	-	+	+

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

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. For example,

the influence of A reduction techniques in methods lowers the information about the tuneability and fallibility, since this requires better information about the 'worst-case' performance. Especially a meta-RSM, but also a technique such as SMAC, coverge very fast towards the best area's, and therefore provide less information about the rest of the search space. Hence, scoring relatively low on information, but obviously they are expected to score high on performance. On the other hand, a complete absence of A reduction techniques also lowers the information, since they provide less information about the good areas.

The multi-objective methods of course score high on robustness in the problems space, since they are specifically build for it. The score of SMAC on this, heavily depends on the amount of characterizing features that are known for the set of problems. If a lot of them are present, and their influence on the performance is high, using SMAC can have huge benefits both on information and performance.

Multi-objective methods also score high on robustness regarding success, since they can optimize on both performance, and the variance of performance. This allows the user to decide each time the algorithm is run, how those two should be balanced.

These rating are only based on theoretical grounds, and approaches that share the same rating, can perform very differently on these aspects. The rating should only be used as an indication of its specialties, rather than a judgment of its absolute quality. However, we still can conclude that there is no such thing as the best tuner, even without taking the performance into account. Namely, tuners do not only balance between performance and information, but also between the different types of information. Depending on the needs of the researcher, one could be more interesting than the other. In general, the modern tuners discussed earlier are nicely balanced, but are still leaning towards a certain direction.

2.4.3 Performance Comparison

If one is out for 'the best tuner', comparing parameter tuning methods based on performance is an obvious approach to choose it. However, anytime when selecting a 'best' from a set of alternatives, the question rises: "How to define the best". Finding the best evolutionary algorithm for solving real-valued problems is already hard. It mainly depends on the choice of user preferences (Section 1.2.1): Which functions does it need to solve, how to measure the performance, are we interested in robust performance, etc. It is even more difficult when comparing parameter tuners. On top of these choices, one also needs to define the algorithm that is tuned, and how to

measure to performance of the tuner itself. Furthermore, the information that can be gained using a specific tuner can vary widely, and therefore different user preferences will lead to a different choice for 'the best tuner'.

Nevertheless, comparing tuners on a specific set of preferences can lead to useful insights, as will be shown in this section. Without any general claims, we show on a specific set of functions, that all tuners are able to fine-tune a state-of-the-art evolutionary algorithm. To show the usefulness of each of the tuners from a performance point of view, we try to answer the following questions:

1. Which of the tuners performs best when comparing them on the performance on each function?
2. Can we use the multi-objective approach of Bonesa to identify generalists and class-specialists that outperform the recommended parameter-values? And what can we learn from them?

2.4.3.1 Experimental Setup

In this set of experiments we will, due to computational constraints, restrict the set of tuners in the comparison to five state-of-the-art tuners. These are REVAC++, SPO, Bonesa, SMAC and I/F-RACE. To make the task for the tuners extra challenging, we deliberately require the tuners to tune an EA that is hard to improve. Hence, we have selected the $(1, \lambda^s_m)$-CMA-ES [5]. The $(1, \lambda^s_m)$-CMA-ES is the general case of the $(1, 2^s_m)$-CMA-ES and $(1, 4^s_m)$-CMA-ES, that were two top performing algorithms from the BBOB'10 [110] competition. The $(1, \lambda^s_m)$-CMA-ES has six different parameters (Table 10), of which five are real valued, and one requires integers.

Table 10: *The parameters of the* $(1, \lambda^s_m)$*-CMA-ES*

Parameter	Description	Range	Default
λ	number of children	[2 10]	4
σ	the initial coordinate wise standard deviations for the search	[1 10]	2
THF	stop if the historical fitness changes are smaller than this	[0 0.1]	$1e-12$
TF	stop if the fitness change is smaller than this	[0 0.1]	$1e-12$
CMACS	cumulation constant for step-size	[0 1]	0.5
CMACCOV	learning rate for rank-one update	[0 1]	0.1121

Furthermore, to be able to compare the tuned parameter values with parameter values set by an expert (to which the tuners did not have access), we have also selected

the BBOB'10 test-suite for the application layer. Since the two $(1, \lambda ms)$-CMA-ES were submitted to this renowned competition, it can be expected that these are the best possible parameter-values as determined by an expert. The 25 functions are supposed to cover a whole range of different problems without favoring certain types of algorithms. Testing on the whole test-suite of 25 functions is infeasible due to the computational costs required, therefore we have selected one function from each of the five categories (Table 11), and used only its 2 dimensional instance. We do not expect this to influence to outcomes of the tuner comparison much, however obviously does influence the outcomes of the individual tuner sessions. The rest of the competition is kept the same as specified: each algorithm (instance) is graded based on the number of evaluations needed to reach a certain fitness threshold, with a maximum allowed number of evaluations of $20,000$.

Table 11: *BBOB'10 Test-suite: selected problems*

	Class	**Function**
f_1	Separable	Sphere
f_6	Low or Moderate condition	Attractive sector function
f_{10}	High Condition	Ellipsoid with monotone x-transformation
f_{15}	Multi-Modal	Rastrigin with both x-transformations
f_{20}	Multi-Modal with Weak Global structure	Schwefel $x \cdot sin(x)$ with tridiagonal transformation

With the choice of user-preferences (test-suite and performance measures), and the choice of algorithm fixed, the only choices left are related to the tuning itself. We allowed each tuner to do $10,000$ tests, such that they are only evaluated based on their quality of finding good parameter-vectors within a reasonable time-frame (a couple of days of computational time), rather than finding them fast. Furthermore, since only Bonesa is specifically built for multi-problem optimization, all tuners (including Bonesa) were allowed to define a specific parameter vector for each problem, instead of having a single parameter vector to solve them all. Bonesa has also been run in multi-problem mode, in which it identified problem-specific parameter-vectors (specialists), a single robust parameter setting (generalist) and per-class parameter vectors (class-specialists) [1]. In terms of the BBOB'10 test-suite, this means that all tuners have a *crafting effort*[60] of 0.7, since each parameter vector is specific to each problem. However, Bonesa is also run with a crafting effort of 0, namely only a single parameter value is used, or any number in between, specific to the number of selected class-specialists. The crafting effort (CrE) is defined as:

[1] Mind that these class-specialist are of different classes, than distinction in the BBOB'10 testsuite. In fact, they do not even have to obey this distinction, since classes can be defined based upon which parameter-values can be used to solve them, rather than what the characteristics of the problems are

$$CrE = -\sum_{1}^{K} \frac{n_k}{k} log(\frac{n_k}{k})$$

where $n = \sum_1^K n_k$ is the number of functions in the testbed and n_k is the number of functions where the parameter setting with index k was used, for $k = 1, \ldots, K$.

Since most of the tuners do not use the same number of tests per parameter-vector (and sometimes only estimate the quality), a validated performance is calculated based on the mean performance over 100 runs with that specific parameter-vector. This is then repeated 10 times, such that for each tuner, on each problem, 10 parameter-vectors are found that are tested 100 times.[2]

Tables 12, 13, 14, 15 and 16 describe the setup of the tuners, which are the default parameter values. BONESA and SPO are included twice. As mentioned before, BONESA has been ran in single-problem and multi-problem mode, and SPO has been run with and without OCBA. The source code for each of the tuners was provided by the original authors [132, 10, 8, 127, 75]. Due to computational reasons some of them are slightly altered to allow parallel computing.

The sourcecode for the $(1, \lambda_m^s)$-CMA-ES and BBOB'10 testsuite are both in Matlab and is the original code used in the competition, as published by Auger et al. [5].

Each of the seven tuners has been ran 10 times, on five different function, therefore a total of $10 \cdot 7 \cdot 5 \cdot 10.000$ runs of the $(1, \lambda_m^s)$-CMA-ES are carried out using seven 2.4GHz CentOS octo-core machines with 10Gbit/s connections, resulting in up to 50 parallel threads. The whole experiment took more than a month to finish, excluding the time needed to setup and configure the tuners.

2.4.3.2 Results

Since most of the tuners are only suited for single-problem tuning, the main comparison on performance has to be based on the so called 'specialists'; those parameter vectors that are specifically designed to solve only a single problem. This indicates how effectively a tuner is able to search the space of all possible parameter-vectors for that single one with the best performance.

[2]The rules of this competition are equal to those in the case study (Section 3.2), and 2009 [127] comparison. That comparison featured three tuning approaches, that were at that time the state-of-the-art. Two of these methods are still in this comparison, namely REVAC++ and SPO. Furthermore, the test-suite is changed. At that time, the CEC 2005 was the most widely used test-suite for numerical optimization, but is now surpassed by the BBOB'10 test-suite.

Table 12:
Bonesa Setup

Parameter	Value
# Problems	1 and 5
Mean support	50
Minimal support	10
Expected Distance	50
ϵ	0.05
k/w ratio	10000

Table 13:
REVAC++ Setup

Parameter	Value
Population size	100
Selected Points	50
Smoothing	5
Minimum # evaluations	2
Maximum # evaluations	5
Multiplication Factor	1.5
Multiplication Threshold	100

Table 14:
SMAC Setup

Parameter	Value
x_0	$[0]^6$
max N	5
$nInit$	1000

Table 15:
SPOT Setup

Parameter	Value
Model	Kriging
# Initial design points	60
# Samples per point	5 or OCBA
# Candidates	10

Table 16:
I/F-Race Setup

Parameter	Value
μ	5
T^{first}	5
T^{each}	1
Statistical test	F-test

2.4.3.2.1 Specialists

Figures 21 to 25 show the results of the parameter tuning sessions. The quality of REVAC++, single-problem Bonesa, I/F-Race, SPOT and SMAC is determined by doing 100 validation runs using the parameter vector that was returned as 'the best'. For the multi-problem version of Bonesa, first the specialists are extracted from the Pareto front for each individual problem, and their performances are then validated using 100 validation runs. Hence, a specific parameter-vector (specialist) is used for each problem, but they originate from a single tuning run. In Section 2.4.3.2.2 we discuss how the multi-problem Bonesa can also be used to identify a generalist parameter vector, that can be used for all problems.

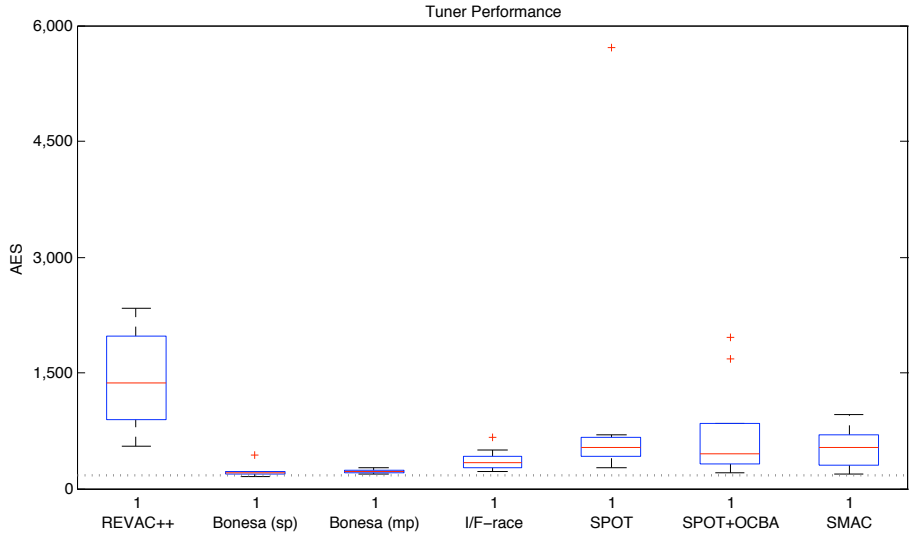


Figure 21: *Performance comparison based on the average number of evaluations to success on f_1, of the 10 specialists as delivered by the 7 tuners. The dotted line shows the performance using the recommended parameters.*

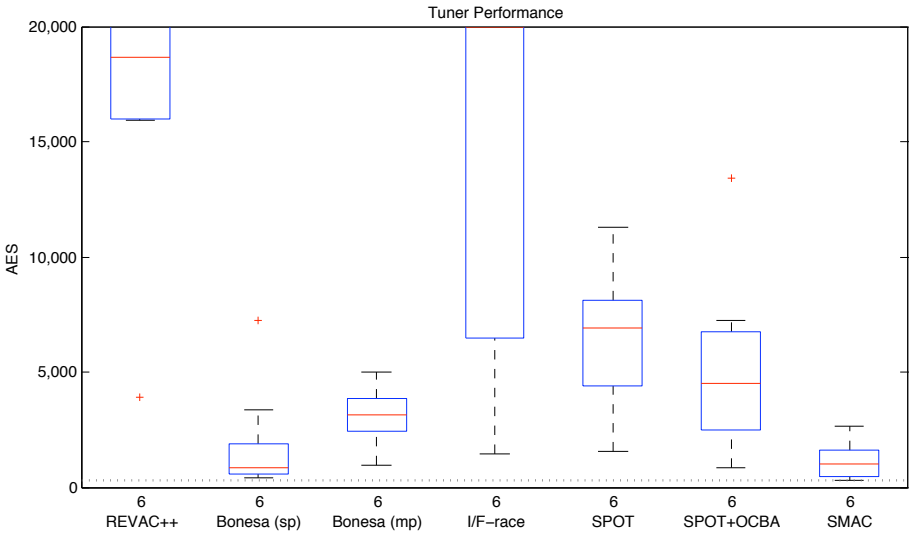


Figure 22: *Performance comparison based on the average number of evaluations to success on f_6, of the 10 specialists as delivered by the 7 tuners. The dotted line shows the performance using the recommended parameters.*

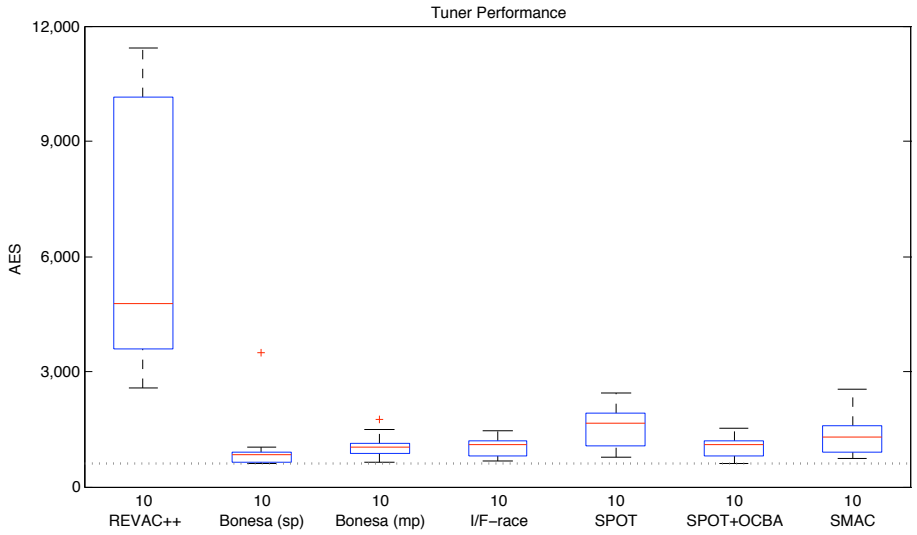


Figure 23: *Performance comparison based on the average number of evaluations to success on f_{10}, of the 10 specialists as delivered by the 7 tuners. The dotted line shows the performance using the recommended parameters.*

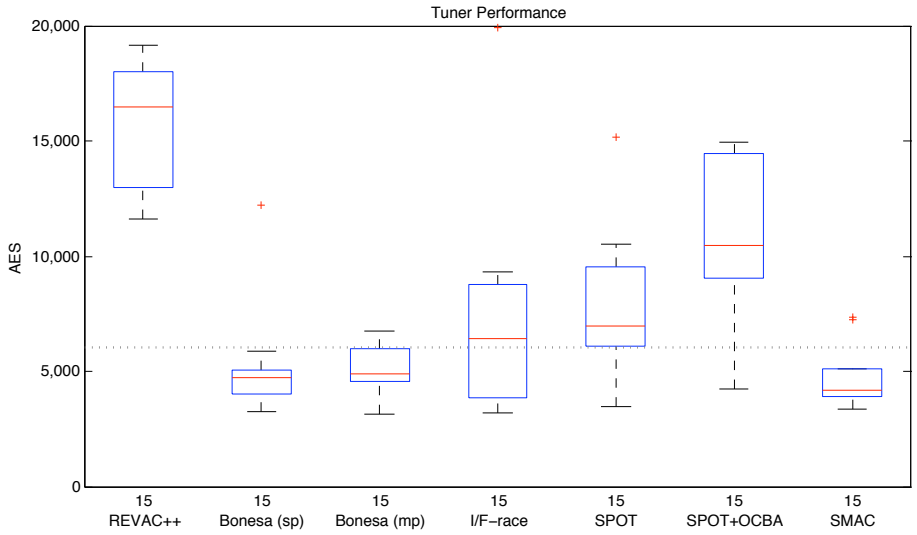


Figure 24: *Performance comparison based on the average number of evaluations to success on f_{15}, of the 10 specialists as delivered by the 7 tuners. The dotted line shows the performance using the recommended parameters.*

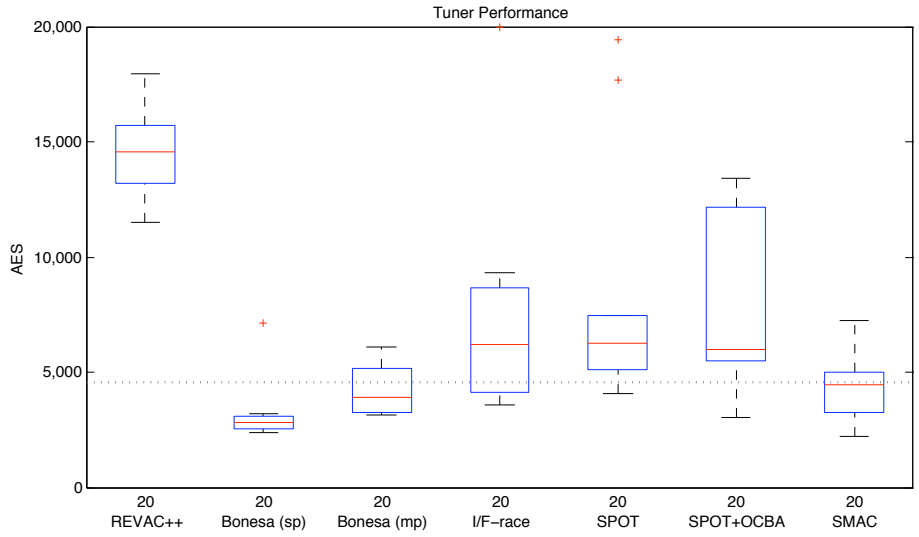


Figure 25: *Performance comparison based on the average number of evaluations to success on f_{20}, of the 10 specialists as delivered by the 7 tuners. The dotted line shows the performance using the recommended parameters.*

Remarkably, there are two tuners that significantly outperform all others, on all five problems (based on a t-test with $\alpha = 0.05$). Both single-problem Bonesa and SMAC perform very well, both on performance, and stability. In general, they have the best performance, and are able to reach this level of performance in most of the runs. Third are the Bonesa specialists identified in a multi-problem run. Even though the tuning task is much more complicated, namely the parameter Pareto front needs to be found, it still manages to identify very good specialists. I/F-Race is also very competitive, however it is much less stable than SMAC and Bonesa, and even completely fails on f_6 (Figure 22).

Both versions of SPOT perform quite reasonably, and it is interesting to note that OCBA sometimes helps (Figures 22 and 23) finding the good parameter values, sometimes hinders the search (Figure 24), and sometimes leads to either very good or very bad performance (Figures 21 and 25). This confirms the results the authors found on their artificial testbed [14], in which they concluded that OCBA helped in 3 out of 5 problems, which is close to the ratio in our experiments.

It is also interesting that the recommended parameters, as used for the BBOB'10 testsuite, perform very well. On f_1 and f_6, none of the tuners managed to find a parameter vector that performs significantly better. However, on f_{15} and f_{20}, all tuners (except for REVAC++) identified parameter vectors that perform significantly better,

at least in one out of 10 tuning runs. The single-problem Bonesa even terminated with better parameter values in 9 out of 10 runs. The difference between f_1 and f_6 on the one hand, and f_{15} and f_{20} on the other, is the 'location' of the best parameter values, and more specifically, the parameter values for the restart parameters THF and TF. Both f_1 and f_6 are unimodal functions, therefore restarting is unlikely to be beneficial. However, they only way to prevent restarts is to set both THF and TF to values smaller than 10^{-6}, which is only $\frac{1}{10^8}\%$ of the search space and very close to the border of the parameter-space. This probably caused that the tuners did not sample this area (enough), resulting into a relatively low performance on the easy functions compared to the recommended parameter vector. Since f_{15} and f_{20} are (highly) multimodal they can benefit from restarts. Although THF and TF are then still very important, their optimal values are no longer at the edges of the search space, and might therefore be easier to find. f_{10} is somewhere in between the unimodal and multi-modal functions, since it has a very clear general structure, and local optima with only very small bases of attraction therefore a very strict restarting approach might lead to the best results.

2.4.3.2.2 Generalists

Although the recommended parameter values were very efficient for solving f_1, f_6 and even f_{10}, the performance on f_{15} and f_{20} left room for improvement by a generalist. Table 18 shows that Bonesa was indeed, in half of the runs, able to identify a parameter vector that is a better generalist. In run 5 (for which the parameter values are shown in Table 17), without losing much performance on f_1, f_6 and f_{10}. It managed to solve f_{15} and f_{20} in only half of the number of evaluations. Also on run 6, 8 and 10 the recommended parameter vector is significantly outperformed, while run 1, 4 and 9 are not significantly better or worse. f_6 proved to be the main cause for differences in performance, since the best AES and the worst AES differ by a factor 10.

Table 17: *The best and recommended parameters of the* $(1, \lambda_m^s)$*-CMA-ES*

Parameter	Description	Range	Recommended	Best Found
λ	number of children	[2 10]	4	9
σ	the initial coordinate wise standard deviations for the search	[1 10]	2	8.99
THF	stop if the historical fitness changes are smaller than this	[0 0.1]	$1e-12$	$1.2e-6$
TF	stop if the fitness change is smaller than this	[0 0.1]	$1e-12$	$9.2e-6$
CMACS	cumulation constant for step-size	[0 1]	0.5	0.78
CMACCOV	learning rate for rank-one update	[0 1]	0.1121	0.96

However, tuning on the total AES of the five functions, is not something that specifically requires a multi-problem approach such as is implemented in Bonesa. Although problems can rise when tuning on such an aggregation (see Section 3.1), in

principle any parameter tuner can be used for this. However, in Section 1.2.2.1 we already mentioned that it would be interesting to know how many vectors are needed to solve all the problems from the test-suite. If we can discover that each problem can be solved by either using parameter vector $\bar{x}_1$ or $\bar{x}_2$, in theory, much better performance could be reached, with only twice the effort. If algorithm-designers would be able to provide such a "shortlist" of parameters vectors, algorithm-users would be able to get much better results with only little effort when compared to only using the single recommended parameter vector.

Table 18: *Multi-problem results for a single generalist vector, the average number of evaluations to solution and the 95% confidence interval are given*

Run	f_1	f_6	f_{10}	f_{15}	f_{20}	**total**
recommended	165(± 6.1)	326(± 12.8)	622(± 23.1)	6018(± 1031.7)	4583(± 727.0)	11713 (± 1262)
1	195(± 15.7)	3380(± 589.3)	814(± 107.1)	4182(± 734.5)	3657(± 641.3)	12228(± 1145)
2	267(± 34.2)	3716(± 647.9)	851(± 137.3)	5100(± 982.9)	3241(± 701.3)	13175(± 1378)
3	272(± 36.5)	2278(± 376.6)	1216(± 206.6)	6938(± 1151.7)	4173(± 754.8)	14876(± 1443)
4	216(± 21.1)	2848(± 424.0)	737(± 81.8)	3821(± 598.6)	3148(± 519.8)	10770(± 903)
5	177(± 9.1)	359(± 36.6)	607(± 36.4)	3013(± 570.8)	2312(± 432.8)	6468(± 718)
6	241(± 27.5)	2286(± 427.5)	733(± 83.4)	3924(± 717.8)	3381(± 540.2)	10564(± 999)
7	292(± 34.3)	3767(± 736.1)	1021(± 124.0)	5165(± 924.1)	3605(± 732.8)	13851(± 1396)
8	174(± 16.2)	579(± 80.0)	630(± 62.5)	3622(± 648.6)	2776(± 551.5)	7782(± 858)
9	209(± 17.4)	2647(± 458.4)	677(± 84.8)	4023(± 776.2)	3234(± 627.0)	10790(± 1102)
10	205(± 17.2)	878(± 153.2)	658(± 57.5)	3983(± 749.7)	2936(± 544.8)	8660(± 941)

Therefore, in each multi-problem tuning run we let Bonesa select two parameter vectors to solve the whole test-suite, rather than only one. Table 19 and Table 20 show these results. Table 19 shows the mean performances and their 95% confidence intervals for each of the two parameter vectors (a and b) per tuning run as selected by Bonesa. Table 20 is an aggregation of this, showing the combined strength of the two class-specialists. It is immediately clear, that the added value of this approach looks rather limited. In most cases, the pure 'generalist' is also selected as one of the two class-specialists and outperforms the specialist on all problems. However, run 6 shows the added value of this approach. The first class-specialist solves problem f_1 and f_6, not remarkably the two unimodal functions for which we concluded earlier that they require different parameter values, while the second one is used to solve the multimodal ones. The total is therefore significantly lower, than by using only a single vector. Also on run 1, the performance is significantly increased by adding a specialist for solving f_6.

Again we noticed the effect of THF and TF on the performance. The recommended parameter values (using $1e-12$ for both parameters), significantly outperformed our class-specialists on f_1 and f_6. Again, the problem of reaching the edges of the

Table 19: *Multi-problem results of the two class-specialist vectors in each run, the average number of evaluations to solution and the 95% confidence interval are given*

Run	f_1	f_6	f_{10}	f_{15}	f_{20}	**total**
recommended	165(± 6.1)	326(± 12.8)	622(± 23.1)	6018(± 1031.7)	4583(± 727.0)	11713 (± 1262)
1a	195(± 15.7)	3380(± 589.3)	814(± 107.1)	4182(± 734.5)	3657(± 641.3)	12228(± 1145)
1b	278(± 35.4)	2415(± 458.9)	927(± 104.5)	5602(± 1124.1)	4139(± 695.5)	13361(± 1403)
2a	217(± 25.1)	4667(± 780.7)	875(± 100.3)	5733(± 1064.1)	3905(± 772.9)	15397(± 1533)
2b	265(± 35.5)	5023(± 886.8)	693(± 59.2)	6951(± 1221.5)	3968(± 725.2)	16900(± 1676)
3a	315(± 48.0)	7161(± 1148.9)	1327(± 195.0)	7583(± 1207.6)	5220(± 1055.0)	21606(± 1983)
3b	252(± 27.8)	941(± 159.1)	1340(± 193.0)	8114(± 1010.1)	5766(± 989.6)	16413(± 1436)
4a	279(± 35.0)	4137(± 720.2)	1037(± 136.8)	5906(± 949.7)	4324(± 868.8)	15683(± 1482)
4b	229(± 25.2)	3166(± 653.0)	1048(± 139.2)	6305(± 956.8)	4731(± 831.1)	15479(± 1433)
5a	177(± 9.1)	359(± 36.6)	608(± 36.4)	3013(± 570.8)	2312(± 432.8)	6468(± 718)
5b	222(± 21.2)	2665(± 514.8)	903(± 122.3)	5210(± 925.3)	3479(± 665.7)	12479(± 1257)
6a	241(± 27.5)	2286(± 427.5)	733(± 83.4)	3924(± 717.8)	3381(± 540.0)	10564(± 999)
6b	177(± 15.5)	1327(± 231.0)	814(± 99.2)	4327(± 804.3)	3569(± 597.1)	10214(± 1033)
7a	292(± 34.3)	3767(± 736.1)	1021(± 124.0)	5165(± 924.1)	3605(± 732.8)	13851(± 1396)
7b	256(± 37.3)	3863(± 737.7)	1120(± 133.7)	6936(± 1048.4)	4087(± 755.7)	16262(± 1495)
8a	174(± 16.2)	579(± 80.0)	630(± 62.5)	3622(± 648.6)	2776(± 551.5)	7783(± 858)
8b	252(± 28.8)	3118(± 547.7)	741(± 74.0)	4123(± 698.1)	3561(± 621.8)	11795(± 1086)
9a	209(± 17.4)	2647(± 458.4)	677(± 84.8)	4023(± 776.2)	3234(± 627.0)	10790(± 1101)
9b	226(± 28.3)	3517(± 653.2)	845(± 98.5)	5219(± 741.9)	3560(± 642.7)	13367(± 1184)
10a	205(± 17.2)	878(± 153.2)	658(± 57.5)	3984(± 749.7)	2936(± 544.8)	8660(± 941)
10b	232(± 34.6)	4588(± 819.6)	1026(± 125.5)	5180(± 943.6)	4000(± 769.7)	15026(± 1474)

search space, made the search for class-specialists nearly impossible. However, the recommended parameter values indicate that such class-specialist are likely to exist. To verify this, we used the parameter values found in run 5, that showed excellent performance on f_{10}, f_{15} and f_{20}, and replaced their THF and TF values with the recommended values (Table 22) used during the competition. Table 21 shows the result of these control runs.

It clearly indicates that we can distinguish two main types of problems, namely unimodal(f_1 and f_6) and multimodal (f_{15} and f_{20}). Although this is not a very novel insight, it shows the strength of taking a more scientific approach. Rather than just selecting the best parameter values, we try to get to know our algorithm. Here, for solving the unimodal problems, THF and TF need to be very small, which effectively means that restarts are turned off. This obviously makes much sense, because restarting is only meant to get out of a local optima if the search get stuck. In case of a unimodal problem, the only optimum is the global one. This confirms the existence of class-specialists, although Bonesa did not discover them directly, due to the tiny fraction of the search space with a high quality. However, because this is quite a general rule, it can be directly applied. Either one knows if it is dealing with a unimodal or multimodal problem, and selects the corresponding vector, or one just performs 2 separate runs to determine which one is more suited. In terms of the BBOB'10 competition, this means

Table 20: *Aggregated multi-problem results using two class-specialist vectors, the average number of evaluations to solution and the 95% confidence interval are given*

Run	f_1	f_6	f_{10}	f_{15}	f_{20}	**total**
recommended	165(± 6.1)	326(± 12.8)	622(± 23.1)	6018(± 1031.7)	4583(± 727.0)	11713 (± 1262)
1	195(± 15.7)	2415(± 459)	814(± 107.1)	4182(± 734.5)	3657(± 641.3)	11264(± 1083)
2	217(± 25.1)	4666(± 780.7)	875(± 100.3)	5733(± 1064.1)	3905(± 772.9)	15397(± 1533)
3	252(± 27.8)	941(± 159.1)	1327(± 195.0)	7583(± 1207.6)	5220(± 1055.0)	15323(± 1623)
4	229(± 25.2)	3166(± 653.0)	1037(± 136.8)	5906(± 949.7)	4324(± 868.8)	14662(± 1450)
5	177(± 9.1)	359(± 36.6)	607(± 36.4)	3013(± 570.8)	2312(± 432.8)	6468(± 718)
6	177(± 15.5)	1327(± 231.0)	733(± 83.4)	3924(± 717.8)	3381(± 540.2)	9542(± 931)
7	256(± 37.3)	3767(± 736.1)	1021(± 124.0)	5165(± 924.1)	3605(± 732.8)	13814(± 1396)
8	174(± 16.2)	579(± 80.0)	630(± 62.5)	3622(± 648.6)	2776(± 551.5)	7782(± 858)
9	209(± 17.4)	2647(± 458.4)	677(± 84.8)	4023(± 776.2)	3234(± 627.0)	10790(± 1102)
10	205(± 17.2)	878(± 153.2)	658(± 57.5)	3983(± 749.7)	2936(± 544.8)	8660(± 941)

that we were able to beat the $(1, \lambda_m^s)$-CMA-ES by itself on each problem, except for f_1, using a crafting effort of 2.

Table 21: *The performance of the two class-specialists of the* $(1, \lambda_m^s)$*-CMA-ES, the average number of evaluations to solution and the 95% confidence interval are given*

Vector	f_1	f_6	f_{10}	f_{15}	f_{20}	**total**
recommended	165(± 6.1)	326(± 12.8)	622(± 23.1)	6018(± 1031.7)	4583(± 727.0)	11713(± 1262)
Specialist 1	177(± 9.1)	359(± 36.6)	607(± 36.4)	**3013(± 570.8)**	**2312(± 432.8)**	6468(± 718)
Specialist 2	**166(± 4.6)**	**304(± 16.7)**	576(± 30.4)	3888(± 703.7)	2769(± 459.8)	7704(± 841)

Table 22: *The class-specialists of the* $(1, \lambda_m^s)$*-CMA-ES*

Parameter	**Description**	**Sp. 1**	**Sp. 2**
λ	number of children	9	9
σ	the initial coordinate wise standard deviations for the search	8.99	8.99
THF	stop if the historical fitness changes are smaller than this	$1.2e-6$	$1e-12$
TF	stop if the fitness change is smaller than this	$9.2e-6$	$1e-12$
CMACS	cumulation constant for step-size	0.78	0.78
CMACCOV	learning rate for rank-one update	0.96	0.96

2.4.3.3 Discussion

Although the comparison on performance as discussed in the previous section has only a limited scope, the results clearly indicate that Bonesa and SMAC are very well capable of tuning an algorithm 'to the max'. That does not necessarily mean that those are the

best tuners (see Table 23), only that on this specific test-suite, and the $(1, \lambda_m^s)$-CMA-ES as algorithm-to-be-tuned, and these specific 'rules of the game', those two found the best performing parameter vectors. Other aspects such as the information that can be gained from the tuning session, were disregarded in this performance comparison, since these mainly depend on the preferences of the user.

One of the aspects that proved to be the most influential on performance, was the ability to identify the tiny region of the search space with very small values for THF and TF. Values in that specific area were shown to be superior to all others. Remarkably Bonesa and SMAC used a different approach for sampling the search space, but both managed to reach this area where others failed. The $1 + 1$ like approach of SMAC, made sure that the search was pulled towards this area very rapidly. In Bonesa on the other hand, the search was pushed away from the large values for THF and TF, allowing for more time to search the other areas.

Although these were the two top performing tuners, in general we can conclude that most of the tuners discussed here found parameter values that were highly competitive to the recommended ones, especially on the 'hard' problems f_{15} and f_{20}. However, it is also clear that even though the current state-of-the-art tuners have specialized methods to reduce the number of parameter vectors that need to be tested, sometimes the area containing the best parameter values is only a very small needle, in a very big noisy haystack. And given these circumstances, all tuners did a very good job, and showed that you do not need to be an expert in the field to find expert(or better)-quality parameter values since a tuner can do this for you.

The second question that we stated in the introduction of the comparison was, if we could use the multi-problem Bonesa for effectively identifying generalists and class-specialists. With respect to "effectively", we can be very clear. Even if we were interested in a specialist, the degradation in performance using a multi-problem approach is rather limited (confirming the findings in [58]). However, the main strength of Bonesa is obviously its ability to identify generalists. The generalists that Bonesa managed to identify showed to be of a superb quality. compared to the recommended parameter-values, it solved all problems in only half of the number of evaluations. This shows the nice prospects of such algorithms, even without taking into account the extra information that can be gained with such an approach. In the end, many users are after generalists, or class-specialists, rather than specifically problem-tailored parameter values.

Furthermore, we validated the presence of so called 'class-specialists', parameter vectors that show excellent behavior on a specific subset of the problems, and together cover all problems. Again the key factor here was the value of THF and TF. We identified two different classes of problems, namely unimodal and multimodal. This

distinction is rather straightforward, and the rules of the BBOB'10 competition would have allowed for the use of different parameter vectors. Hence, by doing just a simple multi-problem BONESA run, the algorithm would have performed much better in the competition. In fact, it would have been the only algorithm (instance) that reached the top 5 on each of the five functions.

Table 23: *A digest of Table 9 showing the tuning capabilities of the state-of-the art tuners for scientific and competitive testing.*

	Performance	**Type**	**Information about robustness**					
			Seeds		**Parameters**		**Problems**	
Method		**Single/Multi**	**Success**	**Stability**	**Tolerance**	**Tuneability**	**Applicability**	**Fallibility**
BONESA (single problem) [132]	++	single	+	+	□	□	+	+
BONESA (multi problem) [132]	+	multi	++	+	□	□	++	+
Sequential Parameter Optimization (SPO) [10]	□	single	+	-	++	++	+	+
SPO + OCBA [88]	□	single	+	-	++	++	+	+
Iterative F-RACE [8]	□	single	□	□	++	+	- -	- -
REVAC ++ [127]	-	single	□	□	++	+	□	□
SMAC [75]	++	single	□	-	- -	- -	- -to ++	- -to ++

++ excellent quality + good quality □ reasonable quality - poor quality - - very poor quality

III

CASE STUDIES

"This emphasis on scientific testing requires a new set of norms for research. It asks that experimental results be evaluated on the bases of whether they contribute to our understanding, rather than whether they show that the author's algorithm can win a race with the state of the art"

– J.N. Hooker

Case Studies

ABSTRACT

This chapter contains the results of three case studies that are conducted, each of which had a different goal. The first case is used to illustrate how parameter tuning can be used in practice for benchmarking and competitive testing of evolutionary algorithms. It shows that tuners can improve the 'world champion' EA in just a couple of days. The second case illustrates how parameters tuners can be used for a fair comparison of evolutionary algorithms on a certain testbed. Finally, the last case is use to illustrate the use of parameter tuners for scientific testing. Using Bonesa, we gained tremendous insight into a specific algorithm for controlling robots. Insight, which can help to formulate new research questions and identify possibilities for improvement of the algorithm, hence we concluded that such a scientific testing approach is much more valuable than merely identifying the best possible setting.

SECTION 3.1

Competitive Testing for Benchmarking

3.1.1 Introduction

Using algorithmic parameter tuners for competitive testing of EAs offers immediate benefits, since an EA instance can be obtained with improved EA performance. Here the gains can be substantial, while the costs are low. In particular, the tuned EA can greatly outperform the EA based on usual parameter values, while the costs of a tuning session are by all means acceptable, typically in the range of hours. This makes algorithmic parameter tuners interesting for practitioners as well as EC scientists engaged in a performance-based competition (implicitly over a sequence of publications, or explicitly within a programming contest).

The main objective of this case study is to demonstrate this benefit. To this end, we carry out an experimental comparison by the usual EC template: "Our EA beats your EA on an interesting set of test functions", where the only difference between "our EA" and "your EA" is that "our EA" is simply "your EA" with tuned parameter values. To make the demonstration convincing we use an EA that has proved to be very good, hence hard to improve. To find such an EA we turn to the CEC-2005 contest on real valued function optimization, take the overall winner (G-CMA-ES) and try to improve its performance over the whole test suite by tuning it with REVAC. [1]

[1] The rules of this competition are equal to those in the 2009 [127] comparison. That comparison featured three tuning approaches, that were at that time the state-of-the-art. Two of these methods are still in this comparison, namely REVAC++ and SPO. Furthermore, the test-suite is changed. At that time, the CEC 2005 was the most widely used test-suite for numerical optimization, but is now surpassed by the BBOB'10 test-suite.

3.1.2 System Description

In this section, we give a general description of the components that instantiate the three layers as introduced in Chapter 1.1.

3.1.2.1 Application Layer: CEC-2005 Test-suite

We have chosen the 25 benchmark functions provided by Suganthan et al. [137] for the CEC 2005 Special Session on Real-Parameter Optimization. The 25 functions are supposed to cover a whole range of different problems without favoring certain types of algorithms. Functions F_1 to F_5 of the benchmark are unimodal (U) and F_6 to F_{12} are basic multimodal. Functions F_{13} and F_{14} are expanded multimodal and F_{15} to F_{25} are hybrid multimodal test functions that are constructed by combining multiple standard test functions. In order to prevent exploitation of problem characteristics as symmetry, all problems are shifted and many of them are rotated. F_4 and F_{17} are distorted by the addition of white noise.

The CEC-2005 test-suite specifies that algorithms are allowed 10^6 fitness evaluations per problem. Furthermore, for each function a success-threshold is defined. If the algorithm terminates with a best found fitness value below this threshold, then it is regarded as 'successful'. For comparison purposes, we have scaled F_6 to F_{16} and F_{17} to F_{25} in such a way that all functions have a success-threshold of 10^{-6}.

3.1.2.2 Algorithm Layer: G-CMA-ES

As explained in the introduction, we deliberately use an EA that is hard to improve, thus making the task to our tuner more challenging. Our choice is the overall winner of the CEC-2005 contest, the so-called G-CMA-ES, a variant of the the CMA-ES from Hansen [4]. The *covariance matrix adaptation evolution strategy* (CMA-ES) [62, 63, 64] is an ES that adapts the full covariance matrix of a normal search (mutation) distribution. Compared to many other EAs, an important property of the CMA-ES is its invariance against linear transformations of the search space. We sum up the general principle of the algorithm in the following and refer to [61] for the details.

Each generation λ children are sampled independently according to a multi-variate normal distribution. The μ best offspring are recombined into the new mean value $\vec{x}$, using a weighted recombination scheme that inherits most from the best individual and takes a gradually decreasing amount of information from the worse of the μ best

offspring. Only initial values for the object variables $\vec{x}^{(0)}$ and step sizes $\sigma^{(0)}$ have to be set depending on the problem to be solved.

The default population size prescribed for the CMA-ES grows logarithmically, the corresponding formula reads $\lambda = 4 + a \log_2(D)$, where D denotes the dimension of the search space. On multi-modal functions, the optimal population size λ can be considerably greater than the default population size [61].

For the restart strategy, CMA-ES is stopped whenever one stopping criterion as described below is met, and an independent restart is launched with the population size increased by a factor of $d = 2$. Hansen and Auger [4] report that for the increasing factor, values between 1.5 and 5 could be reasonable. To decide when to restart, the following (default stopping) criteria do apply.

i) Stop if the range of the best objective function values of the last generations is zero, or the range of these function values and all function values of the recent generations is below a certain threshold (e).

ii) Stop if the standard deviation of the normal distribution is smaller than a pre-specified tolerance in all coordinates and the evolution path is smaller than this tolerance in all components.

iii) Stop if adding a 0.1-standard deviation vector in a principal axis direction of the covariance matrix does not change the mean value.

iv) Stop if adding 0.2-standard deviation in each coordinate does not change the mean value.

v) Stop if the condition number of the covariance matrix exceeds a certain threshold.

The distributions of the starting points $\vec{x}^{(0)}$ and the initial step-sizes $\sigma^{(0)}$ are derived from the problem dependent upper and lower bounds of the initial search region (L_u and L_b).

Unlike most algorithms, the parameters of the CMA-ES and their default values are not described as fixed values, but as a function of other parameters and/or problem characteristics in [61]. This may suggest fewer parameters, however, most of these functions still contain "magic constants" like the 2 in $\mu = \frac{\lambda}{2}$. Therefore, we decided to tune these "magic constants" instead the standard parameters like λ and μ. In this way, we can use the knowledge of the authors about parameter interactions, without restricting ourselves to a fixed setup. The parameter-functions and the constants tuned are summarized in Tables 24 and 25. The ranges are chosen in such a way that we can be reasonably sure that the good parameter values are included. This setup requires $k = 5$ parameters to be tuned.

Table 24: *Parameters, defining formulas, and magic constants of the G-CMA-ES*

Parameter	**Symb.**	**Defining formula**	**Magic const.**
Offspring size	λ	$\lambda = 4 + a \cdot log_2(n)$	a
Population size	μ	$\mu = \frac{\lambda}{b}$	b
Step size	σ	$\sigma = c \cdot (L_u - L_b)$	c
Population multiplication	d		
Stop Crit. theshold	e		

Table 25: *Setup of the G-CMA-ES. This table summarizes the "magic" constants introduced in Table 24.*

To be tuned	**Value-range**	**Recommended Value**
a	$[1, 10]$	3
b	$[1, 5]$	2
c	$[0.02, 10]$	0.5
d	$[1, 4]$	2
e	$[0, 0.001]$	10^{-12}
Termination		10.000 evals

3.1.3 Experimental Setup

In this study we use REVAC to find the best possible set of parameter-values for the G-CMA-ES. In the most basic approach, we define the performance of a given EA instance $EA(\bar{p})$ by the fitness of the best candidate solution at termination. Consequently, if the best candidate solution found in a given run of $EA(\bar{p})$ is $\bar{s}^*$, then the utility of $\bar{p}$ is $f(\bar{s}^*)$. However, in this study, it should not be calculated on a single function, but on the whole CEC-2005 test-suite.

This optimization contest crisply specifies the metrics used to compare the competing evolutionary algorithms. The two principal components of these metrics are the test suite and the EA performance measure. Furthermore, there are rules on technical details, for instance about the required precision when optimizing the given objective functions, the number of fitness evaluations EAs are allowed to do, or the number of independent runs with an EA to calculate its performance. In general, any experimental comparison between EAs is based on some metric X. Then it is a natural idea to use this metric X as utility function within the tuner to find an EA that scores well on

X. In principle, this guarantees that (the parameters of) the EAs are optimized for the right objectives. Our present study follows this rationale, but makes a number of practical compromises that amount to using slightly modified version X' of the final comparison metric as utility function during tuning. The modifications are such that X' is in essence the same as X, but it can be calculated much faster.

As for the test suite, the CEC-2005 rules specify that a contestant EA is to be tested on each of the 25 problems (equally weighted), and is allowed to use $D \cdot 10^4$ fitness evaluations per run per problem, where D is the dimensionality of the given problem. In order to reduce the time used for tuning, we decided that REVAC calculates utility values based on $D \cdot 10^3$ fitness evaluations only. This reduces the duration of the tuning sessions at the cost of the quality of information used to guide the tuning algorithm. Although it is likely that the best possible parameter vector with $D \cdot 10^3$ evaluations differs from the best with $D \cdot 10^4$ evaluations, our results indicate that the best parameter vector found also performs well using $D \cdot 10^4$ evaluations without the corresponding tuning costs.

As for the EA performance measures, the official CEC-2005 list contains three of them,

- MBF,
- SR,
- Rank,

and the rules prescribe that the performance of each EA is to be calculated based on 25 independent runs. In this study, all three performance measures are used to form utility functions and we tune the G-CMA-ES for each of these independently. This implies that we perform three complete tuning sessions such that REVAC is using either of these performance measures to calculate the utility of parameter vectors. For each of these utility functions it holds that $u(\vec{p})$ of a parameter vector $\vec{p}$ is calculated by running the G-CMA-ES 5 times independently on the whole test suite with the parameter values in $\vec{p}$. This represents a second modification of the final comparison metrics meant to deliver computationally cheaper utility functions. Once again, we deliberately trade quality of information for execution speed, and the results show that this is not harmful.

To cope with the two notions of algorithm quality (parameter vector quality) we introduce the term *predicted utility* and *verified utility*. Predicted utility stands for the one used during the tuning session, i.e., during a REVAC run. In general, this is the

Table 26: *REVAC Parameters*

Population Size	80
Best Size	40
Smoothing coefficient	10
Repetitions per vector	5
Maximum number of vectors tested	5000

X' in the first paragraph of this section; here this is based on $D \cdot 10^3$ fitness evaluations and 5 repetitions. By verified utility we mean the one used for reporting the final outcomes. In general, this is the X in the paragraph above, and it corresponds to the ultimate comparison metric behind the given experimental study. Here we use the official CEC-2005 definitions, based on $D \cdot 10^4$ fitness evaluations and 25 repetitions.

To complete this section EA performance measures need to be specified. MBF and SR are commonly used ones, therefore we omit their definitions. Rank, however, is not that well known. Conceptually, Rank is a lexicographic ordering based on SR (primary measure) and MBF (secondary measure) such that the best parameter vector gets rank 1. An important difference between MBF and SR on the one hand, and Rank on the other hand is that SR and MBF can be calculated for any parameter vector v in isolation, while Rank is a relative measure showing how good v is within a set of vectors V. Technically, given a set V of parameter vectors, the rank $R(v)$ of any $v \in V$ is calculated in two different ways, depending on its SR value:

- if $SR(v) > 0$, then $R(v)$ equals its SR-based rank in $\{u \in V : SR(u) > 0\}$
- if $SR(v) = 0$, then $R(v)$ equals its MBF-based rank in $\{u \in V : SR(u) = 0\}$ plus $|\{u \in V : SR(u) > 0\}|$

This definition allows us to calculate rank R for any set of parameter vectors. During a tuning session with REVAC, the set V is changing over time and at any time t it is the set of all parameter vectors that have been generated and tested till that time.

With the application layer, algorithm layer, and the definition of utility, we complete the instantiation the three layers, by stating the REVAC setup (which are the default settings) in Table 26.

3.1.4 Results

The results from the experiments can be investigated from two different viewpoints, namely the performance of the best parameter vectors (one for each utility function),

and the values of the best parameter vectors themselves (again, one for each utility function).

3.1.4.1 Performance

In this section, we present the outcomes of our experiments by showing the verified utilities of the three best parameter vectors (one for each utility function) in Table 27,28 and 29, together with the results obtained by using the recommended parameter values in the G-CMA-ES. From these data it is immediately clear that the performance measure used for tuning highly influences the utilities. Comparing the outcomes, we find that for each of our three measures, the best parameter vector is the one that is tuned on that specific measure. While this is not surprising per se, it is worth mentioning as it confirms that REVAC is capable of tuning an EA *for a specific user preference*, e.g., MBF, SR, or Rank.

It also shows that the recommended parameters that are used in the G-CMA-ES are very good for solving the 'easy' functions, but on 'hard' functions they perform far worse.

From Table 27 we can observe how the choice for Mean Best Fitness as performance measure has influenced the results. The best found vector does not perform well on the 'easy' functions, hardly reaching values below 10^{-6}. However, on the 'hard' functions, especially F_{15} it outperforms all other vectors. As predicted in section 1.2.2.1, during tuning it was mainly aimed at improving the MBF on functions with high objective values, rather than generally improving the MBF. This effect is clearly visible in Table 28. Furthermore, on hardly any of the problems, the best found fitness dropped below the success-threshold. The opposite is true for the parameter vector found by using success-rate as performance measure. On most of the easy functions, it outperformed the other vectors, and was often able to get the perfect score of 25 successes out of 25 runs. However, on the 'hard' functions it performed much worse. Table 29 shows this 'all or nothing' behavior in which it either is the best, or the worst of the three tuned vectors. The recommended parameters perform reasonably well on the success-rate, which was one of the main criteria in the evaluation of the CEC-2005 competition.

The parameter vector found by tuning on the rank shows the most constant behavior. It performs well on the easy functions, and not very bad on the hard functions. What is also clear from Table 29 is that the small increase in MBF on the hard functions was enough to compensate for the loss in performance on the 'easy' functions. This shows the effect of using the ranks, rather than the raw performance value.

Table 27: *Results using Mean Best Fitness as performance measure*

Parameter Vector	**AVG**	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}
Recommended Parameters	7.7e-3	4.5e-15	2.2e-15	0.0e-15	4.5e-15	2.4e-11	4.7e-5	4.7e-7	2.0e-3	9.7e-4	7.9e-4	1.3e-4	8.6e-2
Best found using mean best fitness	2.0e-3	5.6e-7	6.7e-7	5.0e-7	7.9e-7	1.1e-5	4.9e-4	8.2e-5	2.0e-3	1.7e-4	2.0e-4	9.9e-5	4.6e-6
Best found using success-rate	3.0e-3	4.7e-9	5.1e-9	6.2e-9	6.8e-9	2.3e-7	5.8e-8	1.1e-7	2.0e-3	1.1e-4	1.3e-4	6.1e-7	4.0e-5
Best found using rank	2.5e-3	1.1e-7	1.2e-7	1.0e-7	2.0e-7	9.3e-7	1.3e-6	2.6e-7	2.0e-3	1.3e-4	1.2e-4	6.4e-6	2.6e-7
Parameter Vector	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}	F_{21}	F_{22}	F_{23}	F_{24}	F_{25}
Default Parameters	8.9e-5	3.4e-4	3.4e-2	1.1e-2	1.0e-3	8.4e-3	7.6e-3	8.2e-3	9.0e-3	7.7e-3	8.3e-3	2.3e-3	4.0e-3
Best found using mean best fitness	6.3e-5	3.3e-4	5.5e-3	6.5e-3	1.0e-3	3.0e-3	3.0e-3	3.0e-3	3.0e-3	7.3e-3	8.7e-3	2.0e-3	4.0e-3
Best found using success-rate	6.0e-5	1.0e-4	2.8e-2	7.8e-3	9.7e-4	3.2e-3	3.2e-3	3.8e-3	4.9e-3	7.2e-3	8.0e-3	2.0e-3	4.0e-3
Best found using rank	6.0e-5	1.5e-4	1.5e-2	7.4e-3	1.0e-3	3.0e-3	3.0e-3	3.0e-3	5.0e-3	7.2e-3	8.1e-3	2.0e-3	4.0e-3

Table 28: *Results using Success-rate as performance measure (only solved functions are shown)*

Parameter Vector	**Problems Solved**	**SR**	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_9	F_{10}	F_{11}	F_{12}	F_{16}
Recommended Parameters	9	29.9 %	25	25	25	25	25	22	24	0	0	7	9	0
Best found by mean best fitness	5	1.4 %	25	22	25	19	0	0	0	0	0	0	2	0
Best found by success-rate	11	37.7 %	25	25	25	25	25	25	25	8	4	25	24	0
Best found by rank	11	29.9 %	25	25	25	25	16	11	25	4	4	0	25	2

Keeping the rules of the CEC-2005 contest in mind, we take the Rank-optimal parameter vector as our final recommendation, and in Table 30 we compare our best EA with the overall winner of that contest.

3.1.4.2 Best Parameter Values

The results in the previous section clearly show that the performance measure used within the tuner highly influences the results on the 25 test problems. The vector tuned on MBF is focused on a few 'hard' problems, while the one tuned on success-rate is mainly focused on the easy problems. The question rises which parameters are responsible for these different behaviors. In Table 31 we display the three best parameter vectors (one for each performance measure) with their verified utility.

The difference between the recommended parameters and the tuned values is clear. The e-value for the three tuned vectors values are up to 10^5 times as big, resulting in much more restarts without losing to much precision. However, to indicate the differ-

Table 29: *Results using the rank as performance measure*

Parameter Vector	**AVG**	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}
Recommended Parameters	2.80	1	1	1	1	1	2	3	1	4	4	2	3
Best found using mean best fitness	2.44	1	4	1	4	4	4	4	1	3	3	4	4
Best found using success-rate	1.60	1	1	1	1	1	1	1	1	1	1	1	2
Best found using rank	1.48	1	1	1	1	3	3	1	1	2	1	3	1
Parameter Vector	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}	F_{21}	F_{22}	F_{23}	F_{24}	F_{25}
Recommended Parameters	4	4	4	4	2	4	4	4	4	4	3	4	1
Best found using mean best fitness	3	3	1	2	2	1	1	1	1	3	4	1	1
Best found using success-rate	1	1	3	3	1	3	3	3	3	1	1	1	1
Best found using rank	1	2	2	1	2	1	1	1	2	1	2	1	1

Table 30: *Comparing the original and the REVAC-tuned versions of the G-CMA-ES. (For MBF and Rank, lower is better.)*

	Performance over the whole test suite by		
EA	**Mean Best Fitness**	**Success-rate**	**Rank**
REVAC-tuned ES	2.5e-3	29.9%	1.48
CEC-2005 winner	7.7e-3	29.9%	2.80

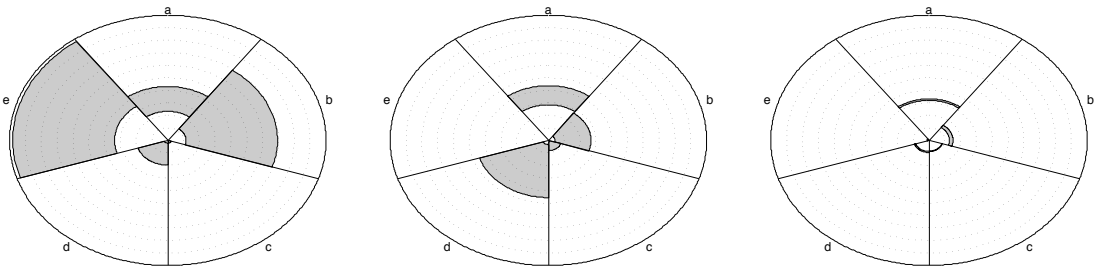
Table 31: *Best Parameters Found by REVAC*

	Default	*Performance measure used for tuning*		
	value	**Mean Best Fitness**	**Success-rate**	**Rank**
a	3	4.82	4.93	3.61
b	2	2.34	1.31	1.14
c	0.5	0.14	0.55	0.81
d	2	1.13	2.37	1.20
e	10^{-12}	$4.0 \cdot 10^{-4}$	$1.8 \cdot 10^{-6}$	$2.8 \cdot 10^{-5}$

ences between the three tuned vectors, Figure 26 is much more informative. Figure 26 shows the top 1% of all generated vectors in each run, based on the corresponding utility function. The gray area shows the .05 and .95 quantile of these best performing parameter values. From these figures, it is immediately clear that e and c highly influence the MBF. As could be expected, e needs to be large for an optimal performance based on MBF. This ensures that on 'hard' problems as many repetitions as possible are executed. Secondly, these Figure 26 and Table 31 also show the importance of the stepsize c, namely only very small values lead to the best MBF performance.

Although hard to see, c and e are also the most influential parameters when tuning on success-rate and rank. The parameters that are tuned for success-rate are mainly focused at solving unimodal problems. Therefore, the values for stepsize c and stop-condition e are quite low in order to gradually climb towards to the success-threshold without restarts. However, compared to the recommended settings, still many more restarts are executed. For rank based performance, both e and c are somewhat larger. This ensures a good fitness on 'hard' problems, due to a larger number op repetitions and a broader search.

Based on these results, we can conclude that there is no such thing as "robust parameter values", because the good parameter values depend strongly on the performance measure, even for generalist EAs optimized for a large collection of problems.



(a) Tuned on Mean Best Fitness (b) Tuned on Success Rate (c) Tuned on Rank

Figure 26: *The good parameter ranges when using the three different performance measures. The parameter ranges from Table 25 are scaled to* $[0, r]$

3.1.5 Discussion

Perhaps the most catchy aspect of this study is the evidence that, using REVAC, we could improve the 'world champion' EA in just a few days, spent on tuning its parameters. While this can be seen as a nice result itself, it is the far reaching implications of this case study that form the main message. Namely, this exercise demonstrates the ease of improving EA performance by using an automated tuning method. In other words, we have shown that the costs of tuning with our technology are by all means acceptable. Our case study also gives a hint about the possible gains. Considering that the 'world champion' EA must have been carefully designed and optimized approximating its best possible performance, the room for possible further improvements could not be very large. Yet, we succeeded in improving it with ease. For 'normal' EAs that are not pushed to their limits yet, the margins for possible improvements are expectedly much much bigger, and all our experience with REVAC indicates that it is possible to realize these improvements. To formulate it simply, automated tuners are able to find a more competitive setup (possibly for for many EAs and other meta-heuristics), at acceptable costs.

Our results also indicate that one has to be careful with claims about robust parameters. In particular, we found that using different EA performance measures for tuning can lead to different optimal parameter vectors. In other words, REVAC has disclosed the existence of 'specialized generalists', that is, EAs that are generalists in the sense of performing well on a large set of test problems (rather than on one problem only), but are specialists in the sense that they perform well only along one performance measure and not along another one. This shows that the very notion of robust parameters is questionable. To get more insights into these kind of dynamics, competitive testing is not sufficient, and more scientific testing is required.

Finally, the question arises what the outcome of the CEC-2005 competition would have been, if all of the participants had tuned their algorithm. Definitely, some algorithms would have benefited more from tuning than others, since it is not likely that all of them are equally tuneable. The results, hence the final rankings, would have been most likely different, but without further research it remains an open question whether we crowned the wrong king.

SECTION 3.2

Competitive Testing for Comparison

3.2.1 Introduction

In Section 1.3.2 we introduced Figure 27 as an illustration of the contribution of parameter tuning to competitive testing. We argued that in order to properly compare two algorithms, both have to be tuned-to-the-max. In the previous section, we concluded the same: in order to crown the right king, all algorithms should be tuned equally. Therefore, in this case study we will illustrate this statement by comparing two state-of-the-art algorithms that were designed for the same purpose, namely winning the CEC 2005 numerical optimization contest from the previous section.

Obviously, we do not want to redo the entire competition with tuned contestants. Thus, to keep the computational costs of the experiments within reasonable limits, we restrict ourselves to one objective function used in the contest and we only tune two EAs: the overall best (G-CMA-ES) and the best one on the chosen objective function (SaDe) as reported in [53]. Furthermore, we use REVAC++ rather than standard REVAC to reduce the computation costs even further. So we will use 1 objective function and 2 EAs to be tuned by REVAC++.

The first one was declared the winner: the G-CMA-ES. And after winning this competition, it was used in hundreds [65] of different problems and applications, because it was "the best". The second algorithm we evaluate, SaDE, is far less popular, just as happens to most 2nd places.

Since the G-CMA-ES is already described in the previous section, we will only introduce SaDE and the objective function to be optimized (application layer), together

with other details of our experiments in the next section.

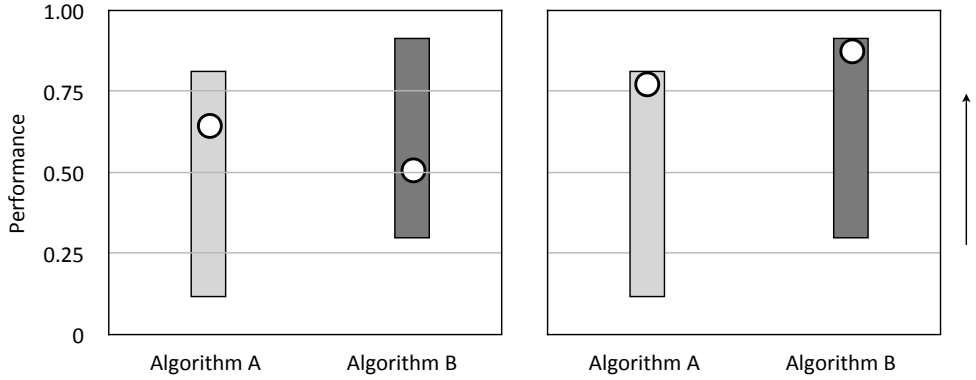


Figure 27: *The effect of parameter tuning on competitive testing. Left: the traditional situation, where the reported EA performance is an "accidental" point on the scale ranging between worst and best performance (as determined by the used parameter values). Right: the improved situation, where the reported EA performance is a near-optimal point on this scale, belonging to the tuned instance. This indicates the full potential of the given EA, i.e., how good it can be with using the right parameter values.*

3.2.2 System Description

In this section, we give a general description of the components that instantiate the three layers as introduced in Section 1.1.

3.2.2.1 Application Layer: Details of the objective function

As reasonably difficult test case, we selected the shifted expanded Griewank plus Rosenbrock function (F_{13}) defined by

$$\begin{aligned} F8(x) &= \sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod_{i=1}^{D} cos(\frac{x_i}{\sqrt{i}}) + 1 \\ F2(x) &= \sum_{i=1}^{D-1} (100 \cdot (x_i^2 - x_{i+1})^2 + (x_i - 1)^2) \\ F_{13}(x) &= F8(F2(z_1, z_2)) + F8(F2(z_2, z_3)) + \cdots \\ &\quad + F8(F2(z_{D-1}, z_D)) + F8(F2(z_D, z_1)) \\ z &= x - o + 1 \end{aligned}$$

and the shifted global optimum (o) as specified in [137]. In all our experiments the number of dimensions is $D = 10$. Note that this function is very highly multimodal.

3.2.2.2 Application Layer: The SaDE Algorithm

SaDE from Qin and Suganthan [114] was ranked second in the CEC 2005 contest on the set of "solved multimodal problems", therefore we have selected it to challenge the G-CMA-ES in this study. SaDE is a self-adaptive variant of the DE algorithm which was proposed by Storn and Price [136]. It is described in [114] as the combination of two DE learning strategies. These two learning strategies are chosen to be applied to individuals in the current population with probability proportional to their previous success rates to generate potentially good new solutions. Two out of three critical parameters associated with the original DE algorithm, namely, CR and F, are adaptively changed instead of taking fixed values to deal with different classes of problems. Another critical parameter of DE, the *population size* NP remains a user-specified variable to tackle problems with different complexity.

The two learning strategies chosen in [114] are:

1. Randomly select candidate solutions according to the update rule

$$V_{i,G} = X_{r_1,G} + F \cdot (X_{r_2,G} - X_{r_3,G})$$

It is referred to as "rand/1/bin."

2. Take the best solutions into account. The corresponding update rule reads

$$V_{i,G} = X_{i,G} + F \cdot (X_{best,g} - X_{i,g}) + F \cdot (X_{r_1,G} - X_{r_2,G})$$

It is referred to as "current to best/2/bin."

Assumed that the probability of applying strategy rand/1/bin to each individual in the current population is equal to p_1, then the probability of applying the other strategy is $p_2 = 1 - p_1$. The initial probabilities are set to 0.5. After evaluation of all newly generated trial vectors, the number of trial vectors successfully entering the next generation while generated by the strategy rand/1/bin and the strategy current to best/2/bin are recorded as ns_1 and ns_2 respectively. The number of trial vectors discarded while generated by the strategy rand/1/bin and the strategy current to best/2/bin are recorded as nf_1 and nf_2. Those two numbers are accumulated within a specified number of generations (a value of 50 was chosen in [114]), called the *learning period*. This value will be referred to as LearnGen in our experiments (see Tab. 32 and 36). Then, the probability of p_1 is updated as

$$p_1 = \frac{ns_1 \cdot (ns_2 + nf_2)}{ns_2 \cdot (ns_1 + nf_1) + ns_1 \cdot (ns_2 + nf_2)},$$

and p_2 is chosen as $1 - p_1$.

[114] used different random values from the interval (0, 2] for F. They are determined as realizations of a normally distributed random variable with mean 0.5 and standard deviation 0.3. These values will be referred to as Fmean and Fstd in our experiments (see Tab. 32 and 36).

The crossover rate (CR) plays an essential role in the original DE algorithm [113], because it defines the level of variation. Therefore, in [114] the authors accumulated the previous learning experience within a certain generation interval so as to dynamically adapt the value of CR to a suitable range. They assumed CR normally distributed as a random variable with mean CRm and standard deviation 0.1. Initially, CRm was set at 0.5 and different CR values conforming this normal distribution are generated for each individual in the current population. These CR values for all individuals remain for several generations (5 in [114]) and then a new set of CR values is generated under the same normal distribution. During every generation, the CR values associated with trial vectors successfully entering the next generation are recorded. After a specified number of generations (25 in [114]), CR has been changed for several times (e.g., 25/5=5 times) under the same normal distribution with center CRm and standard deviation 0.1. The mean of normal distribution of CR is recalculated according to all the recorded CR values corresponding to successful trial vectors during this period. With this new

Table 32: *The parameters of SaDE*

Parameter	**Symbol**	**Value range**
Population size	NP	$[2, 200]$
LearnGen		$[2, 200]$
Fmean		$[0.1, 2]$
Fstd		$[0.01, 1]$
CRMstd		$[0.01, 1]$
Termination	10.000 evals	

normal distribution's mean and the standard deviation 0.1, the SaDE procedure is repeated. As a result, the proper CR value range for the current problem can be learned.

The standard deviation used to determine a realization for the parameter CRm will be referred to as CRMstd in our experiments (see Table 36).

Parameters used in the SaDE algorithm are summarized in Table 32.

3.2.2.3 Experimental Setup

In the present study we define the performance of a given EA instance $EA(\bar{p})$ by the fitness of the best candidate solution at termination. Therefore, if the best candidate solution found in a given run of $EA(\bar{p})$ is $\bar{s}^*$, then the utility of $\bar{p}$ is $f(\bar{s}^*)$. At this point it is important to recall that by the stochastic nature of EAs two independent runs of a given instance $EA(\bar{p})$ can yield different results, hence different measurement values for establishing the utility of $\bar{p}$. Consequently, a number of tests, i.e., runs of $EA(\bar{p})$, must be performed for a good estimation of utility. This is, indeed, what REVAC++ does: it performs r repetitions of each run (r is dynamic, depending on racing and sharpening), thus obtaining r measurements $u_1(\bar{p}), \ldots, u_r(\bar{p})$, where $u_i(\bar{p}) = f(\bar{s_i}^*)$ and $\bar{s_i}^*$ is the best solution vector found in run i of $EA(\bar{p})$. Therefore the *predicted utility* of $\bar{p}$ based on these r tests is just the mean performance of these runs:

$$u(\bar{p}) = \frac{1}{r} \cdot \sum u_i(\bar{p}).$$

A complete tuner run therefore involves generating and testing up to A parameter vectors, with up to B tests (i.e., EA runs) per parameter vector and C fitness evaluations per test (i.e., per EA run), resulting in a total budget of $A \cdot B \cdot C$ fitness evaluations.

Different parameter vectors can have predicted utility values based on different

Table 33: *Setup of REVAC++*

Parameter	Symbol	Value
Population Size	m	100
Best Size	n	50
Smoothing coefficient	h	5
Initial no of tests/vector for racing	r_{min}	2
Initial no of tests/vector for sharpening	B_0	5
Length of stagnation period for sharpening	L	200 tests

numbers of tests. This holds for parameter vectors within one tuner run, as well as for best-of-run vectors over a number of independent tuner runs with the same EA. To eliminate such differences and to improve the reliability of final utility values, we perform a validation step with the best-of-run parameter vector, $\bar{p}^*$, after each completed REVAC++ run. To this end, we execute 100 extra runs with $EA(\bar{p}^*)$ and calculate the *validated utility* $v(\bar{p}^*)$ of $\bar{p}^*$ by taking the mean of the resulting additional measurements:

$$v(\bar{p}^*) = \frac{1}{100} \cdot \sum u_j(\bar{p}^*).$$

Then the outcome of one tuner run is the best-of-run parameter vector, $\bar{p}^*$ and its validated utility. Note that the notions of predicted and validated utility are formally identical, they are both means of utility values over a number of tests. Thus, the distinction we make here is not conceptual, but practical, it helps to be accurate about the experimental results we will present later on: they all concern validated utilities. The rationale behind this is that the predicted utility should be based on a few runs, to make tuning fast, while the validated utility should be based on many runs to make the value reliable.

Finally, because of the stochastic nature of the tuner it is advisable to perform more than one tuner run, especially if the tuner is aimed at fast convergence towards a local optima on the utility landscape (such as REVAC++ is doing). Therefore we have conducted 10 independent repetitions of REVAC++ runs, yielding 10 (possibly different) best-of-run parameter vectors, $\bar{p}^*_1, \ldots, \bar{p}^*_{10}$ that we all validate by 100 extra tests as described in the previous paragraph.

The results presented in the next section are obtained with allowing a maximum of 1000 tests for each tuner.

Table 34: *Mean best fitness (to be minimized) of the G-CMA-ES and SaDE, with the original and the tuned parameters. Standard deviations shown within brackets.*

	G-CMA-ES	**SaDE**
CEC2005 parameters	0.971 [0.318]	1.432 [0.254]
REVAC++ tuned parameters	**0.854 [0.239]**	1.105 [0.981]
	0.975 [0.239]	0.851 [0.235]
	1.001 [0.342]	1.105 [0.981]
	0.936 [0.244]	**0.711 [0.212]**
	0.952 [0.246]	0.894 [0.228]
	0.933 [0.282]	0.969 [0.209]
	1.003 [0.340]	1.539 [2.122]
	0.887 [0.283]	1.016 [0.217]
	1.052 [0.363]	0.723 [0.238]
	0.868 [0.275]	0.781 [0.329]

3.2.2.4 Results

Looking at the results from Table 34, we can observe that both EAs, the G-CMA-ES as well as the SaDE, were much improved by tuning their parameters. The differences between the CEC2005 benchmark performance and the tuned performance are significant (tested through unpaired t-tests with $\alpha = 0.05$), which confirms our message: automated parameter tuning can greatly improve EA performance—even if the benchmark to be improved is an EA instance that has been carefully designed by a human expert.

However, tuning in our case has been problem specific and the obtained parameter vectors will not improve algorithm behavior on *all* possible problems. The CEC 2005 competition allowed only for one parameter vector per algorithm, being utilized for all 25 benchmark problems. So two effects merge here: Automated tuning versus human expert setup, and problem specific parameter values versus generally good (default) parameter values. Still, the observed performance differences are remarkable, especially if we take into account that the G-CMA-ES with the default settings is used afterwards to solve completely different problems, that are most likely not represented in the CEC-2005 test-suite.

Another interesting observation is that the algorithmically tuned instance of SaDE outperforms the algorithmically tuned instance of the G-CMA-ES on the given objective function. Somewhat exaggerating, in fact, taking the liberty to generalize this without decent experimental support on the other objective functions, this means that SaDE

Table 35: *Parameter values for the G-CMA-ES, the one with the best performance is in bold*

	λ	μ	σ	Multipl.	Stop Cr.
CEC2005 parameters	10	5	5	2	1e-12
Found by REVAC++	**31**	**8**	**3.53**	**1.5021**	**0.2665**
	29	9	2.71	3.8192	0.9090
	14	3	2.36	3.4282	0.7437
	17	14	3.79	2.5113	0.7088
	37	13	3.60	2.1250	0.5831
	10	2	3.55	2.5678	0.4616
	11	2	3.68	1.1295	0.9310
	25	5	2.78	2.5404	0.7437
	36	14	3.79	3.9063	0.9229
	19	18	4.28	2.4276	0.0010

could have won the CEC 2005 competition, if it had been tuned properly!

It is also interesting to look at the best parameter values the tuners find for the EAs to be tuned. The values for the parameters of the G-CMA-ES are given in Table 35. This table shows that the expert values, especially the stop-criteria, greatly differ from the ones optimized algorithmically. Considering the obtained parameter values with respect to the allowed parameter ranges, we may however assess that the parameter vectors are roughly similar. One may conjecture that the G-CMA-ES parameters are in some sense redundant for this problem: Increasing selection pressure (λ and μ) or increasing the initial stepsizes (σ) seems to lead to the same performance, as shown by the first and the last row. Nevertheless, these are only hints to a possible parameter relationship and could be analyzed further in a more scientific testing approach.

The values for the parameters of SaDE are given in Table 36. In this case, the difference in performance between the expert (default) value and the tuned values is even bigger than for the G-CMA-ES. The large spread in performance over the multiple REVAC runs shows that tuning SaDE is much harder since REVAC often terminates with rather bad performances. However, on the other hand, the benefits of tuning are also much higher with respect to the default values.

Table 36: *Parameter values for SaDE, the one with the best performance is in bold*

	NP	LearnGen	Fmean	Fstd	CRMstd
CEC2005 parameters	50	25	0.198	0.3	0.1
Found by REVAC++	7	57	0.348	0.879	0.484
	9	33	0.712	0.631	0.482
	6	53	0.460	0.699	0.265
	8	**25**	**0.466**	**0.729**	**0.549**
	10	15	0.611	0.532	0.446
	11	60	0.296	0.679	0.562
	5	57	0.538	0.887	0.742
	13	70	0.413	0.956	0.677
	7	24	0.552	0.869	0.492
	8	26	0.416	0.905	0.468

3.2.3 Discussion

Considering the user effort, we can be short: the overhead implied by the use of tuners can be very limited. That is, given an objective function and an EA for solving it, the extra work consists of interfacing the (existing) tuner code with the EA, which does not exceed an hour or two at most. After this, the tuner needs to run unsupervised for a couple of days in order to optimize the parameter settings. For competitive tuning, postprocessing the results of the tuner is not necessary, since we were only interested in a good parameter vectors for the algorithms participating in the 'competition'. These can than be used for fair competitive testing of the algorithms tuned to-the-max.

SECTION 3.3

Scientific Testing

3.3.1 Introduction

For many years, so called *horse-race-papers* [82] have been the dominant type of paper in the field of evolutionary computing. Authors showed how their algorithm outperformed some other algorithm on some test-function or test-suite. These papers shed light on the question 'whether a certain algorithm instance can outperform another algorithm instance on a certain problem'. It does not, however, indicate when or why this is the case, nor does it give an indication of an algorithm's performance on other problems. Such research is therefore only of real interest to people that need to deal with the same specific problem, and need to choose between the same specific algorithm instances.

[68] noted that the fact that an algorithm performs well on a certain problem is no guarantee that it also works on a different one. Even worse, the fact that a certain set of parameter values works well on a certain problem gives no indication of its performance on any other function. Especially since common parameter settings – for instance, a population size of 100 – tend to be much less robust than assumed. In practice, such parameter values are mostly selected by conventions (mutation rate should be low), ad hoc choices (let's use uniform crossover), and experimental comparisons on a limited scale (testing combinations of three different crossover rates and three different mutation rates) rather than based on solid foundations. Until recently, there were not many workable alternatives for such manual tuning.

However, developments over the last couple of years have resulted in a number of automated tuning methods and corresponding software packages that enable evolutionary computation practitioners to perform a more detailed analysis of the algorithm, and the appropriate parameter values without much effort.

It is easy to see that a detailed understanding of algorithm behavior has a great practical relevance. Understanding the effects of problem characteristics and algorithm characteristics on algorithm behavior allows users to make well-informed design decisions regarding the (evolutionary) algorithm they want to use. In general, the two main factors that are the most interesting for such an analysis are the ones that determine the quality of an algorithm: *performance* and *robustness*.

3.3.2 The $(\mu + 1)$ on-line Evolutionary Algorithm

The objective of the study described in this section is to demonstrate a scientific testing approach by analyzing a relatively new algorithm in a real application (robotics), namely $(\mu + 1)$ ON-LINE. The $(\mu + 1)$ ON-LINE algorithm as described by [57] and [26] was devised specifically to provide autonomous adaptation of robot controllers through on-line, on-board evolution. As the notation indicates, $(\mu + 1)$ ON-LINE generates $\lambda = 1$ child per cycle, which then may replace the worst in the population if it achieves higher fitness. The $(\mu + 1)$ ON-LINE algorithm sports a number of specific features to handle the idiosyncrasies of on-line evolution of robot controllers. The more important of these are: re-evaluation and racing.

In on-line evolution, fitness must be evaluated *in vivo*: the quality of any given controller can only be determined by actually assigning that controller some runtime on a robot and see how well the robot performs its tasks. However, different controllers will be evaluated under different circumstances: any controller's evaluation will start wherever and in whatever state the previous evaluation left the robot. To level the playing field among the genomes that encode the controllers, $(\mu + 1)$ ON-LINE re-evaluates them in the population with a given probability ρ. This means that at every evolutionary cycle one of two things happens: either a new individual is generated and evaluated (with probability $1 - \rho$), or an existing individual is re-evaluated (with probability ρ).

An often heard criticism of stochastic search methods in general and evolutionary algorithms in particular is that they are computationally inefficient because they spend so much time evaluating poor candidate solutions. This is especially painful in the context of on-line evolution, because the actual performance of the evolving system (the robot's controller, in this case) drops when these sub-optimal solutions are evaluated. To minimize the amount of time spent trying less than promising individuals, [59] suggested adding *racing* to $(\mu+1)$ ON-LINE. This entails that during a new individual's evaluation, intermediate results are compared to the fitnesses of individuals already in the population to estimate the likelihood that the new individual is good enough to

beat the worst individual in the population and replace it. If it is fairly certain that this new individual is going to turn out worse than the worst in the current population, further evaluation is most likely a waste of time with an elitist replacement scheme such as $(\mu + 1)$ ON-LINE uses. What exactly 'fairly certain' means is determined by two parameters α and β based on a modified version of the Hoeffding inequality [70].

3.3.3 Experimental Set-up

We use Bonesa to optimize the following $(\mu + 1)$ ON-LINE parameters:

α The significance level of the comparison for racing. α can range from 0 to 1, with 0 disabling racing altogether.

β The second parameter that regulates the strictness of the racing comparisons. This can range from 0.5 to 2.0.

σ The choice of σ adaptation scheme. In $(\mu+1)$ ON-LINE, new individuals are mutated using a Gaussian mutation with step-size σ. We consider a number of ways to update σ: two 'standard' self-adaptive schemes, where either a single σ for the whole genome or multiple σs, one for each real-valued gene in the genome, evolve as additional part of the genome as described by [43, pp. 75–77]. The third scheme updates σ using the derandomised approach proposed by the authors of [107], who specifically deem this method useful with small populations. In the result plots, these schemes are identified as 'ss', 'ms' and 'dr', respectively.

τ The length of an episode during which a controller is given the run of the robot to evaluate it. To (re-)evaluate an individual, the genome is expressed in the controller (in this case, the weights of the artificial neural net are set to those specified in the genome), which then determines the robot's actions over τ time-steps, unless evaluation is aborted by the racing procedure. τ can range from 100 to 1200.

P_c The crossover rate. The likelihood of performing recombination of two parents when producing a new individual. Ranges from 0 to 1.

Bonesa has been run once in both single-problem and once in multi-problem mode, in order to compare both approaches, and to take full advantage of their strengths. In single-problem mode, Bonesa should be able to identify 'specialists', parameter values that only work on one particular problem, more effectively since no time is spend on

exploring the good parameter values for the other problems. In principle, this would mean that the performance of the specialist found by single-problem tuning is better than the performance of the specialist found by multi-problem tuning.

In multi-problem mode on the other hand, Bonesa terminates with a whole set of parameter-vectors. An experimenter can choose which of those suits his needs best. For example, certain parameter vectors could be well-suited for solving a subset of problems, rather than only a single problem. Most of these "bronze bullets", would not have been found with a single-problem approach, since they are always outperformed by a true specialist on the problem. But "bronze bullets" are often much more interesting than true specialist because they are more robust. Furthermore, in multi-problem mode, the range of tested parameter vectors is most likely wider, and therefore provides more information about the different behaviors of the algorithm.

3.3.4 Four Tasks

To understand how $(\mu + 1)$ ON-LINE behaves across a range of problems, we test it on four tasks, all commonly found in evolutionary robotics literature: phototaxis, fast forward (or obstacle avoidance), collective patrolling and locomotion.

In all cases, a single trial runs for 10,000 seconds of simulated time; we use time rather than number of evaluations or generations because we are interested in the performance of the robots in real time, regardless of how that is achieved by the evolutionary algorithm. Also note, that the settings for τ, α and β influence the number of evaluations performed per timespan: lower values of τ obviously increase the number of evaluations per timespan, just as increasing the strictness of racing comparisons (by increasing α and β).

3.3.4.1 Phototaxis

Seeking out or tracking a light source is a very straightforward task that has been addressed by many researchers in evolutionary robotics. In our comparison, we use the simplest version of phototaxis: robots only have to move towards a stationary light source and then remain as close to it as possible. The fitness function is simply the sum of the intensities of received light over time.

The arena is an empty (apart from the ten robots) square with a light source in the middle: we ignore collisions between robots in these experiments, so the robots' distance sensors can be ignored.

3.3.4.2 Fast Forward

Moving in a straight line as fast as possible while avoiding obstacles – also known as obstacle avoidance – is maybe the most commonly tackled task in evolutionary robotics research. In a confined environment with obstacles this task implies a trade-off between avoiding obstacles and maintaining speed and forward movement. The fitness function we use is based on one described in [106]; it favours robots that are fast and go straight ahead.

Although fast forward is considered a trivial task, we added some extra difficulty by using a complicated maze-like arena with tight corners and narrow corridors that fit only a single robot and sometimes lead to dead ends. This arena structure, combined with the fact that multiple robots will be simultaneously deployed, makes the task considerably harder than most commonly seen instances. This additional complexity is confirmed by results of baseline trials in this arena that [84] reported: the baseline Braitenberg controllers invariably get stuck after a while.

3.3.4.3 Collective Patrolling

An obvious real-world task for a group of autonomous mobile robots is that of a distributed sensory network, where the robots have to patrol an area as a group and detect events that occur periodically. It differs from the previous tasks since it requires some level of co-ordination: the success of the group depends not only on the efficient movement of the individual robots but also on the spread of the group across the arena to maximize the probability of detecting events.

Somehow, robots need to liaise so as not to patrol the same areas. To this end, they are equipped with a pheromone system: robots continuously drop pheromones (this is a fixed behaviour and not controlled by the evolved controller) while sensors detect the local pheromone levels. These pheromone are also used to determine the fitness, namely the fitness is equal to the sum of the pheromones levels at each grid-point with some discount if an obstacle is close. This collective patrolling task is described by [93] where controllers evolve off-line, although in that work the approach to events is more complicated and the robots use other sensory inputs.

3.3.4.4 Locomotion

The locomotion task differs from the preceding three: it concerns individual robots that are physically linked together to form an *organism* that to all intents and purposes

looks like a single entity. In fact, ten Roombots robots form a four-legged organism as shown in Fig. 28 that need to coordinate their actions. Although each controls its own oscillator, they have a common task of moving as fast as possible. The fitness function

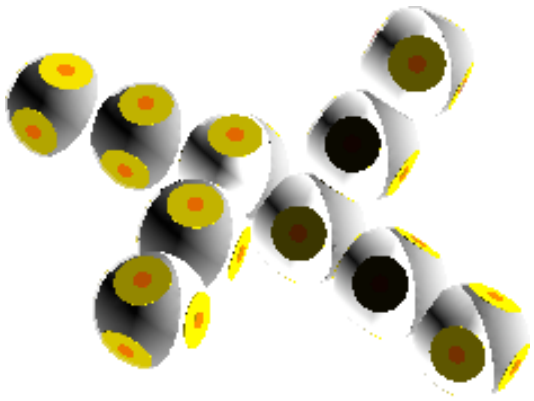


Figure 28: *Organism for the locomotion task. Each of the five consituent modules consists of two spheres. Each module has three degrees of freedom: it can rotate the two (dark and light) halves of each sphere and it can rotate the connection between the spheres.*

is therefore very straightforward: it simply sums the euclidean distance traveled per time-step.

This experiment was first described by [29], who used a non-evolutionary stochastic approximation method, SPSA, to implement on-line adaptation. We refer to that publication for a more detailed description of the robots and the coupled oscillator controller.

3.3.5 Results

The objective of this case study is not to prove that tuning leads to good results – that has been amply illustrated by the previous case studies– but to show the merits of scientific testing. Nevertheless, we do, of course, look at the performance of the tuned instances of $(\mu + 1)$ ON-LINE to compare the results of tuning for a specific task with those of the multi-problem approach.

From Fig. 29, we can see that for three of the four tasks – fast forward, patrolling and locomotion – multi-problem tuning carries no performance penalty. In fact, the locomotion performance of the multi-problem runs is often better than that of tuning specifically for locomotion. For phototaxis, however, the performance is slightly worse

when tuning for all these four tasks at once – this seems to indicate that phototaxis requires parameter settings that conflict with the other tasks, something we will see more proof of later. We also see that parameter vectors that work well for the fast forward task also work well on locomotion and/or patrolling, but that the converse is not necessarily true: the highest performance for patrolling is achieved by parameter vectors with indifferent performance on the fast forward task.

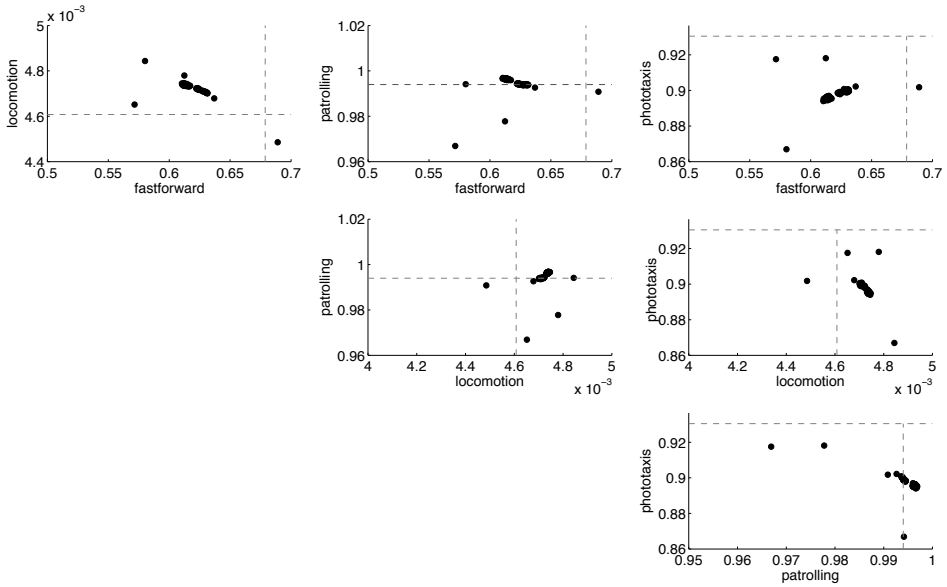


Figure 29: *2-dimensional projections of the Pareto-front. Each plot shows the performance of all non-dominated parameter vectors from the multi-objective run on two tasks. The horizontal and vertical dotted lines indicate the maximum performance reached when tuning for that specific task only. In all cases: higher values are better.*

An important insight from tuning exercises like these is what parameters matter most for algorithm performance; for an analysis of the parameter tolerance and tuneability (Chapter 3.3), we turn to Table 37. That shows the variance of the best performance for each problem over the range of the parameters. For each of the three σ-strategies, for example, the best performance achieved with that specific strategy is noted. Table 37 lists the variance of these performances for each specific problem, divided by the difference between the best and worst achieved performance on that problem. Numerical parameters binned into 100 intervals prior to this calculation. Thus, large values indicate a parameter that causes large a fluctuation in performance – that has a large impact. Using the terminology introduced by [42], such a measure indicates both the tunability and the tolerance of a parameter.

It is clear that τ is the most sensitive parameter, since for all of the problems it has the highest or second-highest variance. This indicates that it needs to be set very carefully, as it has the biggest influence on the performance. ρ comes in second place, and P_c on the third place. The σ-adaptation strategy, interestingly, has the least effect on performance, except in the case of patrolling.

Table 37: *The variance of the best performance across possible values for each of the parameters. The parameter with the highest variance for each problem is shown in bold, the next highest is underlined. The higher, the more sensitive the algorithm is to the value of that specific parameter.*

task	τ	σ	ρ	P_c	α	β
fastforward	**0.0262110**	0.0000177	0.0168660	<u>0.0253330</u>	0.0002089	0.0001982
phototaxis	<u>0.0000622</u>	0.0000010	**0.0002482**	0.0001162	0.0000287	0.0000371
patrolling	**0.0029074**	0.0000082	<u>0.0027625</u>	0.0000057	0.0000076	0.0000088
locomotion	**0.0000463**	0.0000009	<u>0.0000069</u>	0.0000025	0.0000025	0.0000029

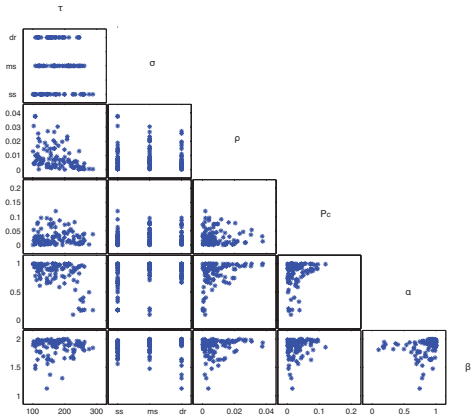


Figure 30: *Non-dominated parameter vectors in fast-forward. A cell shows two entries of each non-dominated parameter vector, for instance τ vs. ρ in the first column, third row. Each dot indicates a combination of values that occurs in the non-dominated set.*

Figure 30 to 34 shows the make-up of non-dominated parameter vectors in pairs of parameter values. For single-task experiments (plots 30 to 33), non-domination is defined as not performing statistically worse (using a Welch-T test with $\alpha = 5\%$) than the best parameter vector found. For the multi-problem results in Fig. 34, Pareto dominance is used, in which "better" and "worse" is replaced with "significantly better" and "significantly worse".

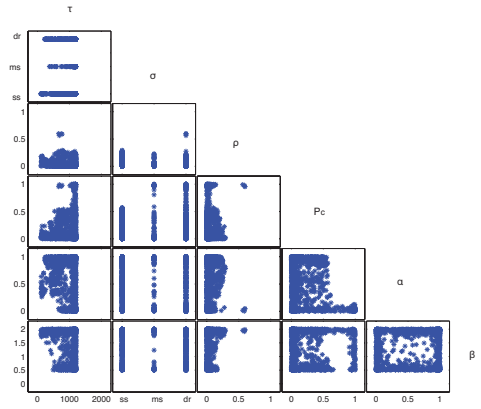


Figure 31: *Non-dominated parameter vectors in phototaxis. A cell shows two entries of each non-dominated parameter vector, for instance τ vs. ρ in the first column, third row. Each dot indicates a combination of values that occurs in the non-dominated set.*

Each dot represents a pair of values in a non-dominated vector; the top-left plot in each matrix, for instance, shows combinations of σ and τ; the dots' vertical position denotes the σ control scheme (single-step self-adaptive (*ss*), multi-step self-adaptive (*ms*) or the derandomised approach (*dr*)), their horizontal position shows the τ value.

As an example, consider Fig. 33. In the cell that shows combinations of τ and σ, we see that $\sigma = 0$ is much more prevalent than other values and that the non-dominated vectors combine that value with low values of τ. This means that, although it hardly influences the best possible performance level that can be reached (Table 37), it heavily influences the sensitivity of the other parameters.

The plot for P_c vs α shows a rectangular region that contains the majority of dots; this indicates that no interaction between these two parameters was found and they can therefore be set to optimal values independently. This contrasts with a cell like that of ρ and α in Fig. 30, which shows interaction: there, if ρ is low, α may have any value and vice versa.

The results from the fast forward task in Fig. 30 show interaction between ρ and the racing parameters α and β: in those vectors where racing is less strict (low α and β), re-evaluation is all but turned off (although ρ is never very high for this task). There are similar interactions between racing and crossover with low values of α and β occurring only when P_c is almost zero. For this task, the strategy for updating mutation step-size seems immaterial. Note, that the values for τ are very low: even the highest values are less than 300, while the full range extends to 1200. Within this relatively small range of τ values, there is an interesting pattern of interaction with ρ: the triangular shape of the region containing the value pairs seems to indicate a

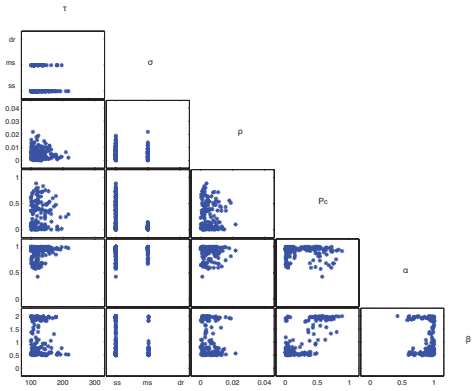


Figure 32: *Non-dominated parameter vectors in patrolling. A cell shows two entries of each non-dominated parameter vector, for instance τ vs. ρ in the first column, third row. Each dot indicates a combination of values that occurs in the non-dominated set.*

minimum number of new evaluations: if τ is lightly higher, ρ is less as if to compensate and vice versa.

Phototaxis (Fig. 31) is by far the easiest task that the robots have to learn as borne out by the number of different non-dominated vectors: many parameter settings lead to near-optimal behaviour. It is the only task that seems to require – or should we say allow – lengthy evaluations (note, that the scales differ from problem to problem). Again, there is evidence of interaction between crossover and racing: racing may be almost off (low α), but only when P_c is high.

The patrolling task (Fig. 32) is the first to show a clear preference when it comes to σ-adaptation schemes: the derandomised approach does not occur in the non-dominated set. In the cases where multi-step adaptation is selected, the crossover rate is very low. The results for this task show an interaction between τ and ρ similar to that for fast forward: if one is high (within the small range that occurs in the non-dominated vectors), the other is low.

The locomotion task is the hardest of the four tasks and it is hard to distinguish any interactions from Fig. 33. It is clear, though, that τ and ρ are mostly low, that racing is quite strict and that the σ-adatation scheme is single-step self-adaptation.

Figure 34 shows the non-dominated parameter vectors when optimising settings for all four tasks at the same time. Here, we see a strong preference for strict racing, but only if P_c is low. Although there are far fewer points with higher crossover probabilities, there seems to be a trend to lower values for α and β as P_c increases. All three σ-adaptation schemes occur, but the derandomised scheme seems to require very specific

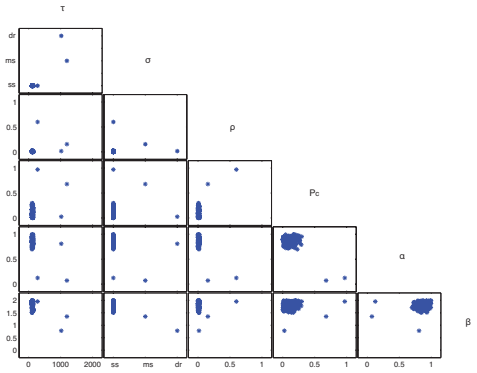


Figure 33: *Non-dominated parameter vectors in locomotion. A cell shows two entries of each non-dominated parameter vector, for instance τ vs. ρ in the first column, third row. Each dot indicates a combination of values that occurs in the non-dominated set.*

settings for the remaining parameters: the dots for *dr* are very closely clustered around specific values for the other parameters.

3.3.6 Discussion

For all tasks except the easy phototaxis task, the best parameter settings tend towards low values for the re-evaluation rate ρ and evaluation length τ and towards settings that imply strict comparisons for racing (high α and β). These settings all indicate that it is preferable to try many points in the search space and not dwell too much on the effects of noisy or unfair evaluations.

From all this, one might conclude that re-evaluation is not a worthwhile feature of $(\mu + 1)$ ON-LINE. This seems at odds with the findings presented by [26], where re-evaluation was shown to be beneficial in e-pucks evolving controllers for the fast-forward task. However, the tests there were conducted with $\mu = 1$, so the population could not as easily recover from genomes that were unluckily evaluated as very poor. Also, our experiments did not consider dynamic environments where higher population diversity could be essential - which in turn might imply a need for higher re-evaluation rates.

It is possible that errornous evaluations pose less of a problem in our experiments because the influence of an unrealistically high evaluation is limited: an individual that actually performs quite poorly may become part of the population, but enough properly good individuals remain to make sure that sufficient good offspring is generated.

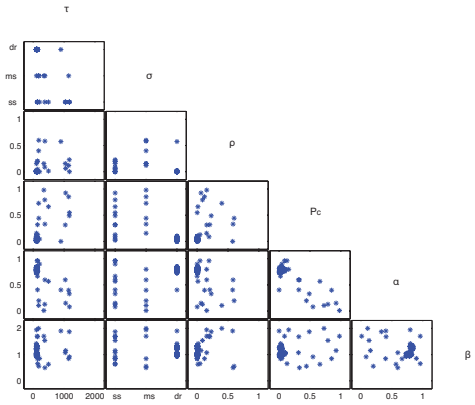


Figure 34: *Non-dominated parameter vectors in multi-problem. A cell shows two entries of each non-dominated parameter vector, for instance τ vs. ρ in the first column, third row. Each dot indicates a combination of values that occurs in the non-dominated set.*

Furthermore, if it is selected as parent, its offspring will be cut short by racing (hence the high values for α and β) and so not even have lasting impact on the robot performance. This may also explain why P_c is set to low values: crossover increases the likelihood of such a bad individual contaminating the offspring of good individuals. Moreover, crossover acts as a macro-mutation [83], so low P_c values lead to a lower population diversity as illustrated in Fig. 35. This increases the likelihood that when a good individual is replaced unfairly with a bad one, a copy or at least a very similar individual remains to continue the blood-line.

We have seen that the phototaxis task requires parameter settings that conflict with those for the other tasks. We can only hypothesize what makes phototaxis so different – possibly the fact that it is so easy to solve. Be that as it may, we can certainly conclude that there is no 'silver bullet' parameter vector that yields optimal $(\mu + 1)$ ON-LINE performance for all tasks. Let us restate the vision that underlies the research into on-line, on-board evolutionary algorithms such as $(\mu + 1)$ ON-LINE: it is the vision of robots that autonomously adapt to any circumstances, particularly ones that we cannot foresee. This goal strongly suggests that there is a need for robust control schemes that allow $(\mu + 1)$ ON-LINE or similar algorithms to change their parameters on the fly and so ensure universal adaptability. However, up to then tuning can be helpful in identifying which problems can be solved. Apparently fast-forward, patrolling and locomotion have something in common, since they can be 'solved' with the same parameter values. Hence, tuning can tell us something about the similarity of problems. This can possibly be used to evaluate the suitability of a certain parameter vector for a certain problem beforehand.

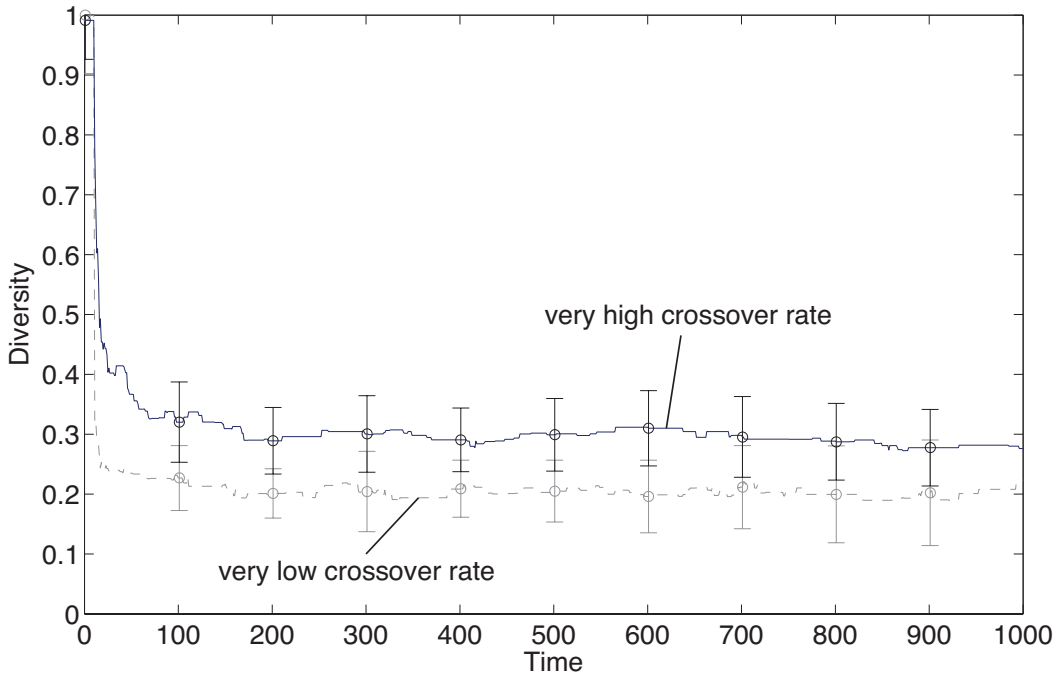


Figure 35: *Average population diversity over time for 10 runs of the fast forward task with low and with high P_c. Although the difference is not spectacular, a high crossover rate seems to increase diversity, suggesting macro-mutation effects.*

Maybe the most important conclusion from this research is a methodological one: analyzing the make-up of the non-dominated parameter vectors as provided by Bonesa yields tremendous insight into the tuned algorithm. This insight can help to formulate new research questions and identify possibilities for improvement of the algorithm. These results of tuning are much more valuable than merely finding the best possible setting for some parameter on a particular problem or even on a range of problems.

IV

THE FUTURE OF PARAMETER TUNING

"It asks scholarly journals to publish studies of algorithms that are miserably failures when their failure enlightens us"

– *J.N. Hooker*

The Future of Parameter Tuning

ABSTRACT

In the this chapter, we discuss possible future use of parameter tuning methods in different areas. We start by illustrating how it can be used for numerical optimization of noisy fitness landscapes. Results on the very challenging and time-expensive Tetris-problem show that on this specific problem, Bonesa can be used to heavily cut down computational time without much loss of performance. This indicates its practical use in this area. Hereafter, we discuss how the strengths of both parameter tuning and parameter control can be combined in order to create high performing and problem-tailored algorithms. Furthermore, we make a first attempt in actually defining such a hybrid, and show that, in contrast to previous work, this basic problem-tailored control mechanism can outperform tuned (but static) parameter values.

SECTION 4.1

The Future of Parameter Tuning

4.1.1 Introduction

Parameter tuning in EC has been a largely ignored issue between the start of evolutionary computing and the mid 00s. However, over the last couple of years, there are promising developments. Algorithms and software have been developed that made automated parameter tuning possible, efficient and accessible. Due to the exponentially growing computer power, and easy access to grids, tuning experiments now take a few hours to a week, something that Greffestette could only have dreamed about when he wrote his first meta-evolutionary algorithm [56].

That does not mean that we have now reached the end of tuner development; people are still inventing new and even more efficient tuners. However, the future of parameter tuning might be much broader than just tuning the parameters. For example, SPO has already been applied as problem solver in machine engineering, aerospace industry, elevator group control, graph drawing, algorithmic chemistry, technical thermodynamics, agri-environmental policy-switchings, vehicle routing, and bioinformatics [13]. So rather than just optimizing the parameters, tuners can be much more widely used to solve real-world problems.

Many real-world applications are, in contrast to most scientific applications of evolutionary algorithms, non-deterministic. Noise is omnipresent, and sometimes quite large. Conditions much like the ones faced with parameter tuning. In the next section we therefore investigate the applicability of tuners for these kind of problems, where noisy is large and function evaluations are very expensive.

But, that is not the only line of future research that can sprout from parameter tuning. Ever since the introduction of the distinction, numerous evolutionary computing

scientists and practitioners posed the question: "Which approach is better, parameter tuning or parameter control?". An obvious answer would be that both have their own strength and weaknesses, so general claims about the superiority of one of the two would be wrong. However, the best static parameter values will never be better than the best scheme that changes them on the fly. Since, in principle, keeping parameter values static is just a special case of parameter control in which the parameter values do not change.

One can therefore ask the question: why would there be a future for parameter tuning? The reason for this is simple, just as there is no such thing as an evolutionary algorithm without parameters, control is not a way of getting rid of parameters. Of course, parameters can be hidden, or can have less influence on the performance, still they are there, and also these parameters need to be set and will influence the performance. Still, some problems require specific settings, and an all-purpose problem solver is just as an illusions as it is now. Furthermore, for most parameters, we have no idea about how to control them. We might have a gut-feeling, but the whole field of parameter tuning has already shown how bad our gut-feelings sometimes are.

In Section 4.3.4 we therefore introduce a generic control framework in which parameter control and parameter tuning work together in order to find very specialized control-mechanisms for repetitive problems. Their combined strength shows that parameter tuning is here to stay.

SECTION 4.2

Parameter Tuning and Noisy Function Optimization

4.2.1 Introduction to Noisy Function Optimization

Noisy function optimization is a subset of problems that is often solved using meta-heursitics such as evolutionary algorithms. Mathematically, a noisy fitness function can be described as follows:

$$F(X) \quad = \quad \int_{-\infty}^{\infty} (f(X) + z) \cdot p(z) \, \mathrm{d}z$$

With $f(X)$ as the noiseless part of the fitness function, z the additive noise, with the probability distribution $p(z)$ with an mean value of 0. Optimizing this function adheres to optimizing $f(X)$ with the constraint that only $f(X) + z$ can be measured.

Many 'real world' problems are inherently noisy since they depend on approximations, inaccurate measurements, simulations or the fitness is the result of a stochastic process. The latter is exactly the same problem as faced with parameter tuning. In essence, the parameter tuning problem is just a subset of all noisy function problems, with specific properties:

- Objective function evaluations are computational expensive
- Noise is often very big

- The fitness-landscape (i.e., utility landscape) may have certain properties, such as relative smoothness and relatively few local optima that are caused by the nature of the problem

Handling noise in fitness evaluations is often an important issue in evolutionary robotics [117] and evolutionary process optimization [48]. The application of EAs in noisy environments has therefore been the focus of many research papers, of which a very extensive overview can be found in [81]. The most common method of dealing with noise, is to enhance a state-of-the art solver for non-noisy problems with specific noise-operators. Those can be divided into three different approaches[81], namely: explicit averaging, implicit averaging and modifying selection.

All previously mentioned parameter tuning methods use in some sense, the explicit averaging approach of repeating measurements, and averaging them to approximate the performance $u(\bar{p})$ of a certain parameter-vector $\bar{p}$. Implicit averaging methods rely on the fact that evolutionary algorithms repeatedly sample promising areas of the search space. As a consequence, by simply using a large population, a generational approach, and proportional selection, the effect of noise can be canceled, as proved for infinitely large populations in [100]. The approach is used by BONESA is therefore similar to this, since it also relies on the strong causality principle to seed a certain promising area of the search space into the population. Modifying selection is an approach used by most racing methods, since it adheres to selecting based on confidence intervals, bounds or hypothesis testing rather than comparing the vanilla averages.

This shows obvious interconnections between parameter tuning methods and noisy fitness optimization. Therefore, the progress that is made the last couple of years on parameter tuning methods might be useful in this field, and vice-versa. To evaluate the effectiveness and merits of both sides, we have selected a very challenging problem from the field of noisy function optimization to be solved, namely evolving a Tetris player.

4.2.2 Evolving a Tetris Player

Tetris is a very popular game that, although it has very simple rules, requires a lot of practice in order to master the game. Designing artificial players has been a great challenge, mainly since the number of possibles states of the game approaches 10^{60}, hence approximation strategies has been developed to lower the complexity. Most current artificial Tetris players are therefore based on an evaluation function that assigns a numerical value to each possible move of the piece. Given this function, the

current state of the board and the current piece, the artificial player will choose the decision that leads to a new state with the highest value. Hence, designing a Tetris player adheres to designing an appropriate evaluation function.

In [22] an evaluation function has been proposed that performs as well as the best existing players, and was based on eight different features, that are in essence measures of a certain state, for example the size of the highest peak, and the number of holes. Each of these eight features is multiplied with a weight-value and summed. The higher this sum, the better this state is scored. In order to find the optimal weights of the evaluation function, they used the current state-of-the-art solver for numerical optimization, the previously mentioned CMA-ES.

4.2.3 Experiment

In the first set of experiments CMA-ES was ran with 20 restarts, and 1500 different weight-vectors per start, and each vector was tested 100 times, to establish its 'true' performance. They concluded that such an approach was not viable, since it took "several months", to finish the experiments. However, they did conclude that "for the feature set presented here, we have probably obtained the best player and further improvements will not be significant", indicating an upper bound of the performance around $36.3 \cdot 10^6$.

Therefore, in a follow-up experiment, they enhanced CMA-ES with three racing approaches, namely racing with Hoeffding bounds, Bernstein bounds, and racing with Tetris bounds [23] in order to determine if it was possible to reach the same level of quality with less computation effort. In [23], they reported a significant improvement in the number of evaluations needed, however also concluded that the choice of bound was highly influencing the effectiveness of the racing approach. The performance of the best found weight-vector was not affected.

Table 38 shows the scores as reported in [23] without racing, and while using the different bounds for racing.

It is immediately clear that the Bernstein bound had no effect on the number of evaluations, while Tetris bounds, and Hoeffding bounds heavily reduced the computational time needed.

The same set of experiments has been executed using Bonesa to find the optimal weight-vector. Table 39 shows the parameter settings of Bonesa that were used, which are also the default values. Bonesa was restricted to use only 5,000 evaluations.

Table 38: *The mean scores over 100 runs, and the number of evaluations using a CMA-ES with and without racing on the Tetris problem*

Strategy	Score	Confidence Interval	Evaluations
No racing	$36.3 \cdot 10^6$	±10%	15,000,000
Bernstein Bounds	$36.3 \cdot 10^6$	±10%	15,000,000
Tetris Bounds	$31.5 \cdot 10^6$	±10%	10,000,000
Hoeffding Bounds	$33.2 \cdot 10^6$	±10%	4,050,000
BONESA	$34.9 \cdot 10^6$	±10%	5,000

Table 39: *Bonesa Setup*

Parameter	Value
Mean Support	50
Minimal support	10
Expected Distance	50
ϵ	0.05
k/w ratio	10000

The added value of the more elaborate way of dealing with noise by Bonesa is very clear. The number of evaluations needed to reach an acceptable performance is much lower, even when we take into account that most probably just a single repeat would have been enough to reach a reasonable performance using the restart CMA-ES+racing. In only 5,000 fitness evaluations, Bonesa managed to reach the same level of performance as the enhanced CMA-ES reached after several million evaluations.

4.2.4 Discussion

In this section we have demonstrated the use of Bonesa, and parameter tuners in general, for noisy function optimization. Although this is only a single example, we believe that tuning algorithms can have a big contribution to the field of noisy function optimization, especially when computational power is the limiting factor, rather than optimization power. The approach of enhancing a good optimizer with special features to deal with the noise might not be the best choice is such circumstances.

Not only did Bonesa reached the same level of performance with only a fraction of the computation power, more importantly, it has been run with the default settings, in contrast to the CMA-ES. There, the choice of bound heavily influenced the results.

Using Bernstein bounds for example, had no effect at all. From the fact that BONESA worked right out of the box, we can conclude that approaching a noisy fitness optimization problem like this, as a parameter tuning problem can lead to great benefits. Especially if the standard deviation of the fitness has the same order of magnitude as the fitness itself, like here, racing techniques are likely to spend to much budget on increasing confidence. Two weight-vectors that are similar are likely to perform similar too. This means that a racing approach will use the total budget in order to determine which one is best. Since the CMA-ES, uses relatively small step-sizes, the chance of having such a race between two very similar vectors is quite hight.

The field of parameter tuning has matured the last couple of years, which resulted in a large set of sophisticated methods of dealing with this kind of noise, and also reducing the computational expenses. For noisy fitness optimization problems that share the same characteristics, such an approach might therefore lead to better results than approaching it as a numerical optimization problem (with noise).

In the future, we expect more studies along these ideas. On important aspect here, is to be able to identify beforehand, when to approach a problem as a parameter tuning problem, and when to approach it as a numerical optimization problem. The noisiness and computational costs are certainly two main factors, but might not be the only ones. Nevertheless, the fact that most 'real world' problems are inherently noisy and often require much computational power makes parameter tuning methods, like Bonesa, a very interesting alternative for usage in industrial and commercial applications.

SECTION 4.3

Static vs Adaptive Parameters

4.3.1 Introduction

Ever since the introduction of the distinction, numerous evolutionary computing scientists and practitioners posed the question: "Which approach is better, parameter tuning or parameter control?" Quite remarkably, there are hardly any studies trying to answer this straightforward question.[1] In this section we address this very question and try to find an answer based on experiments.

Parameter tuning and parameter control can be compared based on theoretical arguments. In principle, parameter control has the advantage that it is capable to use adequate parameter values in different stages of the search. This capability is really 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 of parameters might be optimal at different stages of the evolutionary process. For instance, large mutation steps can be good in the early generations helping the exploration of the search space and small mutation steps might be needed in the late generations to help fine-tuning the near-optimal chromosomes. This implies that the use of static parameters is inherently inferior to changing parameter values on-the-fly if both are done optimally. Furthermore, theoretically speaking, parameter control subsumes parameter tuning, in the sense that using constant parameter values can be seen as a special case of parameter control, where the control policy is simply `keep-this-value`.

The conceptual distinction between tuning and control can be lifted if we consider the control mechanism as an integral part of the EA. In this case the EA and its

[1] We will discuss existing work in Section 4.3.2.

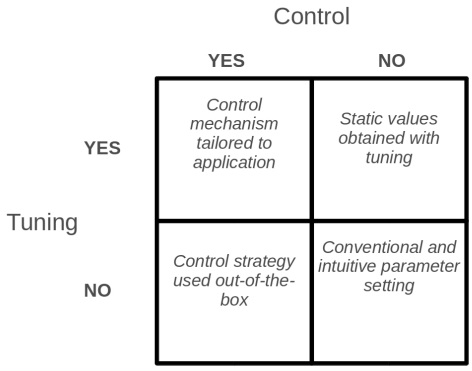


Figure 36: *Tuning vs control*

parameter control mechanism (that may be absent) are considered as one entity. Furthermore, this composed entity may or may not be tuned before being applied to a new problem. The resulting matrix of four options is shown in Figure 36.

The combinations in the top row have the advantage of enhanced performance at the cost of the tuning effort [105]. 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. Such an approach combines on-line parameter adjustment (control) and off-line configuration (tuning). The evolutionary algorithm incorporates a parameter control mechanism (seen as a black box for the moment) but this controller itself has certain parameters that can be configured for a problem through an off-line tuning process. This approach can be seen as an alternative to tuning static parameter values. Instead of spending time to search for good (static) parameter values, the same effort is spent to make a good calibration of a mechanism that performs on-line control of these parameters. Such a method has both the advantage of enhanced performance thanks to varying parameter values, and the possibility to be tuned to a specific problem (and is therefore most appropriate for repetitive problems). On the other hand, this mechanism has the same disadvantage as tuning static parameters, i.e., the need for computational time dedicated to problem tailoring.

In order to achieve this substitution of calibrated control for static tuning, a suitable control mechanism is required. This controller has to be capable to handle any parameter and to be tailored to specific problems. The objective of this section 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 conducted experiments using our controller to control three parameters of a simple evolution strategy: the population size, the generations gap and the mutation step size.

4.3.2 Related work

Parameter control is an increasingly popular topic in the field of evolutionary algorithms [90]. The outline of the most commonly used methods is quite similar: one of the parameter values is altered based on some specific evidence. Most often these methods are designed for specific parameters. The most popular parameter-specific control methods focuses on mutation probability [49], mutation step size [116] and operator selection [135], but methods also exist for the selection pressure [140], the population size [134], the fitness function [91] and the encoding [125].

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 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 σ was introduced. Self-adaptive control is not based on such deterministic rules to change parameter values. Instead, it encodes σ 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 'relearned', the information gained about the appropriate parameter values for different situations, cannot be reused in another run or another EA. In [85] we show that this cost of learning can lead to a decreased performance when dealing with a limited amount of time. A control mechanism using an artificial neural network with specifically learned weights managed to outperform a self-adapting *sigma* on most of the problems, clearly indicating the usefulness of re-usable control schemes in case of repetitive problems.

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 [3]. Another approach was to approximate a good population size on-line: the parameter-less GA [67] automatically performs an on-line search for a good population size by creating and running populations of progressively larger size. Direct control of the population size parameter has also been studied deterministically [87], adaptively [39] and self-adaptively [44].

Some generic control methods for numeric parameters also exist. In [145] an adaptive mechanism is proposed that works in alternating epochs, first evaluating parameter values in a limited set and then applying them probabilistically. At 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 [89] 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 [36] and [95]: 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 [95] 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 [36], 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 do compare control strategies with tuned parameter values [108, 109, 50, 111] however these typically arrive at a conclusion opposite to that of [89, 36, 95].

Extensive literature reviews on parameter control can be found in [40] and [33].

4.3.3 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 to be controlled(i.e., *what* is to be controlled).

2. A set of observables that will be the input to the control mechanism (i.e. what *evidence* is used).

3. An 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.

4.3.3.1 EA State

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

$$S_{EA} = \{G, \bar{p}, \mathcal{F}\} \tag{4.3.1}$$

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

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

4.3.3.2 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 [33]). Instead, here we will only distinguish between *quantitative* (e.g., population size, crossover probability) and *qualitative* (e.g., recombination operator) parameters.

4.3.3.3 Observables

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

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

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

Parentheses denote that steps can be bypassed.

i. *Source*: As stated above, the source of all observables is the current state of the EA, i.e., the set of all genomes, the current parameter values and the fitness function.

ii. *Digest*: A function $D(S_{EA}) = v$ that maps an EA state to a value, e.g., best fitness or population diversity.

iii. *Derivative*: Instead of using directly a value v we might be more interested in its speed or acceleration (e.g. to make the observable independent from 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 $F_H(v_1, v_2, ..., v_W)$[2] 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.

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

4.3.3.4 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., qualitative or quantitative) 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:

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

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:

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

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

Methods such as the 1/5th rule are clear examples of static controllers. Since success-rate is a History (iv) of a Digest (ii), it is, together with the current parameter value σ, part of the EA state. Therefore, given a certain EA state, it always produces the same next EA state.

According to this classification, a time-scheduled mechanism is, on the other hand, 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.

4.3.4 Generic Control for Numeric Parameters

In this section we present a control mechanism that belongs to the upper left category of the matrix in Figure 36. This controller can be tailored to specific applications through an off-line tuning process. Subsequently, it is considered suitable mostly for repetitive applications. The controller combines the advantage of varying parameter values, which can lead to increased performance, and the capability to be calibrated to specific problems. We propose this controller as a possible substitute to tuning static parameter values: instead of spending a certain amount of time for the search of good static parameter values for a problem, the same effort can be used to calibrate the controller for the same parameter and problem.

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

4.3.4.1 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 $[min, max]$ of values must be specified, which will be the output range of the controller.

4.3.4.2 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.

4.3.4.2.1 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: $f_N = norm(f_B), f_N \in [0, 1]$. This observable is only applicable when the bounds of the fitness function are known. The second fitness-based observable Δf 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:

$$f_N = norm(f_B), f_N \in [0, 1]$$

$$\Delta f = \frac{f_B^W - f_B^{W/2}}{f_B^W - f_B^1}, \Delta f \in [0, 1], W = 100$$

where $f_B = max\mathcal{F}(g), g \in G$. In order to be able to use Δf 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.

4.3.4.2.2 Diversity based

Diversity is observed using the Population Diversity Index (PDI) [133] as the digest function, 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.

4.3.4.2.3 Current parameter values

The current values of the controlled parameters $\vec{p_c}$ are also observed and input to the controller when the Δf 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 4.3.4.1).

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 $\{\Delta f, PDI, \vec{p}\}$.

4.3.4.3 Control Method

As a control method we chose an artificial 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. Therefore, such a controller can be classified as static, since it always produces the same parameter values if given the same EA state. 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 with a scalar as to reach very close to its asymptotes given a domain of $[0, 1]$.

When multiple parameter 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 oversimplifying considering parameter interaction but it also simplifies the controller and significantly decreases the number of weights that need to be calibrated.

4.3.5 Experimental Setup

The experiments presented here are designed as a proof of concept for the viability of a generic, EA-independent and parameter-independent control mechanism that is instantiated through an off-line tuning process and targeted to repetitive applications. This controller is examined as a substitute for using tuned but static parameter values.

The experiments compare the performance of the controller described in the previous section controlling a simple ES and solving a set of test problems with the performance of the same ES solving the same problems but with fixed parameter values. For each test problem, the calibration of the controller and the competing fixed values were found using the same amount of off-line tuning effort.

Six standard 10-dimensional problems in numeric optimization are used [7]: Sphere, Corridor, Rosenbrock, Ackley, Rastrigin and Fletcher & Powell.

4.3.5.1 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 μ, the generation gap $g = \frac{\lambda}{\mu}$ and the mutation step size σ. We run five distinct experiments controlling each parameter alone, controlling μ and g together (considering their obvious interaction) and controlling all parameters together. In the experiments where one of the parameters is not controlled/tuned, that parameter value is always set to a value chosen based on conventional EC knowledge and intuition (see Table 40). These values also serve as initial values when a parameter is controlled.

Table 40: *Default values of parameters when not controlled or tuned.*

μ	10
g	7
σ	*(Sphere)* 0.1, *(Corridor)* 0.1, *(Ackley)* 0.6, *(Rosenbrock)* 0.04, *(Fletcher&Powell)* 0.07, *(Rastrigin)* 0.1

4.3.5.2 Off-line tuning

Both controller calibrations and fixed values are found through an off-line tuning process using Bonesa. For each problem, the four versions of the controller (using the four different observables sets described in 4.3.4.2) 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, μ together with g and all together. The experimental setup is summarized in Table 41.

4.3.6 Results

Results are shown in Tables 42 to 46. 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 (e.g., never reached the optimum) in all cases, so it will be completely disregarded in the subsequent analysis.

Table 41: *Experimental Setup*

EA	$(\mu + \lambda)$-ES with: Gaussian mutation, no recombination and uniform random parent selection, limit of 10000 evaluations, default parameter values shown in Table 40
Parameters	$\mu \in [0, 1]$ $g \in [0, 14]$ $\sigma \in [1, 100]$ (individually, μ with g and all together)
Instantiation	Bonesa with a budget of 3000 tests to calibrate weights $w_i \in [-1, 1]$
Problems	Sphere, Corridor, Ackley, Rosenbrock, Rastrigin, Fletcher& Powell
Controller observables	$\{f_N\}$, $\{f_N, PDI\}$, $\{\Delta f, \vec{p}\}$, $\{\Delta f$, PDI, $\vec{p}\}$

4.3.6.1 Performance

Tables 42, 43 and 45 show that attempting to control solely the population size, the generation gap or their combination was unsuccessful. But, controlling only σ (Table 44) 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 46) 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 μ, g combination.

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

4.3.6.2 Parameter behavior

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

Controlling μ and g: As the performance results show (Tables 42, 43 and 45), there is no improvement when controlling μ and g. In fact, in most cases, the calibration of control creates controllers that maintain constant parameter values. When controlling μ alone, controllers 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, values are kept constant either to its minimum, such that

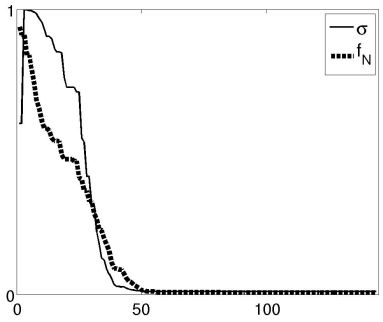


Figure 37: *Example of the behavior of parameter σ over time (Ackley problem with $\{f_N\}$). All values are normalized between 0 and 1.*

each generation has only one offspring (Sphere, Ackley, Rosenbrock) or its maximum (Fletcher & Powell, Rastrigin). Combined control of μ and g shows a different behavior only in the case of the Rastrigin problem, but this does not result in better performance (Table 45). 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 σ: A much more interesting behavior is observed in the values of σ. In most cases, σ 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 σ behavior is shown in Figure 37. Of course, a preferable behavior would be to re-increase σ when a population is stuck in a local optimum. Such a situation can be detected based on the diversity and Δf observables used; however, this desired behavior does not occur in our results. A different control strategy was found for the Rastrigin problem as shown in Figure 38: σ oscillates rapidly while the range of this oscillation seems related to current diversity.

Controlling all combined: Combined control of all parameters produces the best results, with calibrated controllers generally demonstrating a more complex behavior. Values of μ still remain mostly constant, but g generally varies for the Ackley, Rastrigin and Fletcher & Powell problems, but is still kept fixed 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 σ shows a behavior similar to what is described in the previous paragraph. Representative examples of parameters' behavior are illustrated in Figures 39 and 39. In both cases μ is set to a constant value; however, g is controlled according to f_N. In each case, σ shows one of the two behaviors described in the previous paragraph.

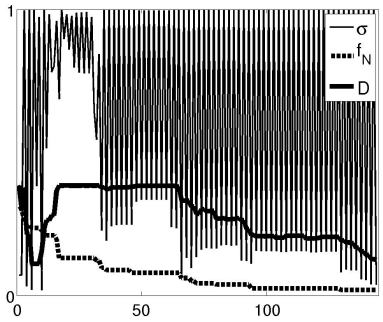
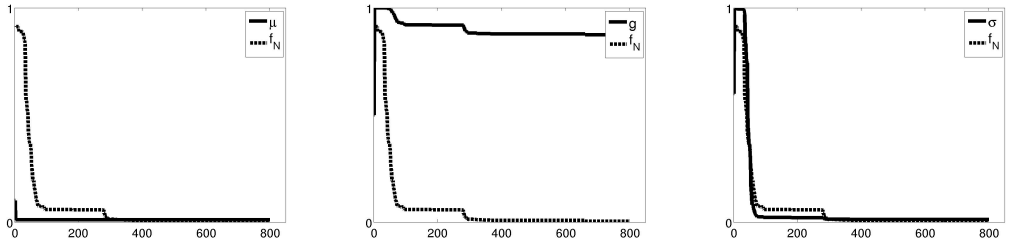
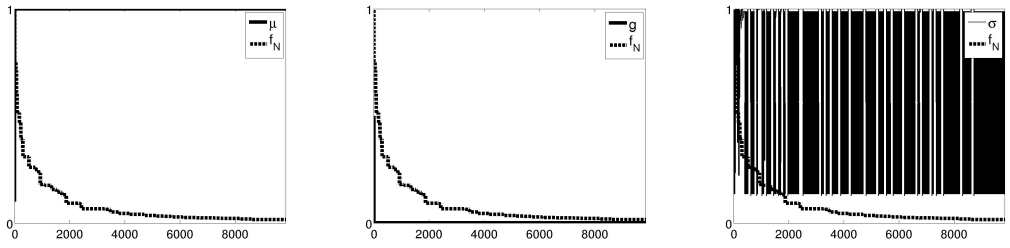


Figure 38: *Example of the behavior of parameter σ over time (Rastrigin problem with $\{\Delta f,\ D,\ \vec{p}\}$). All values are normalized between 0 and 1.*



(a) μ (b) g (c) σ

Figure 39: *Example of parameter behavior over time with combined control (Ackley problem with $\{f_N\}$). All values are normalized between 0 and 1.*



(a) μ (b) g (c) σ

Figure 40: *Example of parameter behavior over time with combined control (Fletcher &Powell problem with $\{\Delta f,\ \vec{p}\}$). All values are normalized between 0 and 1.*

4.3.6.3 Observables

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

Though choosing between f_N or Δf is mostly a matter of feasibility (calculating normalized fitness is not possible if the bounds of the fitness values are not known beforehand), including a diversity observable or not is a more fundamental question. Contrary to our initial expectations, adding diversity to the input does not 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.

4.3.6.4 Discussion on Selection

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

- the failure to calibrate a meaningful controller for μ,
- the absence of control strategies that increase σ late in the run to escape local optima,
- 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 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.

4.3.7 Discussion

Based on our results, the questions stated previously can be easily answered. The main conclusion that can be drawn (although this is only a single example) is that the generic approach taken in this paper is viable and fruitful. In contrast to previous

Table 42: *Performances results for μ. Bold values denote performance not significantly different from static, while underlined values mark a performance significantly better than static (none here).*

	Sph	Crdr	Rsbk	Ack	Rsg	F&P
(f_N,D)	**0.09607**	**9.303**	4.868	**5.75**	36.98	7602
(f_N)	**0.09607**	**9.303**	4.868	**5.75**	36.77	6731
(Δf, D, $\vec{p}$)	**0.09607**	**9.303**	4.868	**5.75**	36.98	6408
(Δf,$\vec{p}$)	**0.09607**	**9.303**	4.868	**5.75**	38.97	5528
Static	**0.09901**	**9.303**	**4.329**	**5.776**	**34.29**	**1335**

Table 43: *Performances results for g. Bold values denote performance not significantly different from static, while underlined values mark a performance significantly better than static.*

	Sph	Crdr	Rsbk	Ack	Rsg	F&P
(f_N,D)	**0.1009**	**9.303**	**5.115**	**5.716**	**55.15**	1.113e+04
(f_N)	**0.1009**	**9.303**	**5.115**	**5.716**	**55.15**	1.065e+04
(Δf, D, $\vec{p}$)	**0.1009**	**9.303**	6.142	**5.752**	**54.4**	1.065e+04
(Δf,$\vec{p}$)	**0.1009**	**9.303**	**5.115**	**5.716**	**55.32**	1.065e+04
Static	**0.1003**	**9.303**	**5.226**	**5.778**	79.35	**5770**

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 δ 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.

Table 44: *Performances results for σ. Bold values denote performance not significantly different from static, while underlined values mark a performance significantly better than static.*

	Sph	Crdr	Rsbk	Ack	Rsg	F&P
(f_N,D)	0.04848	**9.303**	**7.473**	0.26	29.06	6848
(f_N)	**0.01609**	**9.303**	**6.995**	**0.1085**	**23.32**	**5622**
(Δf, D, $\vec{p}$)	0.0528	**9.303**	**8.05**	**0.3153**	27.08	6089
(Δf,$\vec{p}$)	0.0353	**9.303**	**8.111**	0.617	**25.85**	8238
Static	0.05745	**9.303**	**7.066**	3.684	37.4	**4710**

Table 45: *Performances results for μ, g. Bold values denote performance not significantly different from static, while underlined values mark a performance significantly better than static.*

	Sph	Crdr	Rsbk	Ack	Rsg	F&P
(f_N,D)	0.09869	**9.303**	6.89	**5.633**	38.26	5841
(f_N)	**0.09382**	**9.303**	4.789	**5.591**	38.66	6432
(Δf, D, $\vec{p}$)	0.09869	**9.303**	6.89	**5.633**	38.19	4299
(Δf,$\vec{p}$)	**0.09382**	**9.303**	4.789	**5.756**	**36.22**	3465
Static	**0.09475**	**9.303**	**3.834**	**5.575**	**34.08**	**762.3**

Table 46: *Performances results for all. Bold values denote performance not significantly different from static, while underlined values mark a performance significantly better than static.*

	Sph	Crdr	Rsbk	Ack	Rsg	F&P
(f_N,D)	1.102	**9.303**	26.81	3.337	29.28	7824
(f_N)	0.007457	**9.303**	11.54	**0.5488**	**23.99**	5999
(Δf, D, $\vec{p}$)	5.387	**9.303**	82.6	18.5	36.52	2.055e+05
(Δf,$\vec{p}$)	**0.005169**	**9.303**	**7.358**	2.275	30.48	**2028**
Static	0.03565	**9.303**	**7.025**	2.024	35.9	2828

Changing either of these can, in principle, lead to a different conclusion. With respect to the observables chosen, we can conclude that these indeed highly influence the results. Remarkably, adding diversity as an input appears to have hardly any added value for controlling σ.

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

V

CONCLUSIONS

"We have saddled algorithmic researchers with the burden of exhibiting faster and better algorithms in each paper, while expecting them to advance our knowledge of algorithms at the same time. I believe that when researchers are relieved of this dual responsibility and freed to conduct experiments for the sake of science, research and development alike will benefit"

– *J.N. Hooker*

SECTION 5.1

Conclusions

5.1.1 Parameter Tuning and Scientific Testing in Evolutionary Algorithms

As stated in the introduction of the thesis, there are three main subjects treated in this thesis: the distinction between competitive and scientific testing and their relation to user preferences, a survey and comparison of current tuning techniques and the introduction of two new tuners, and finally an outlook into the future of parameter tuning. In this section we reconsider all these matters and conclude the thesis by summarizing the principal message.

Let us begin with recapitulating that parameter tuning in EC has been a largely ignored issue between the mid 80s and mid 00s. 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. Even now, some researchers still believe that using automated approaches to parameter tuning is unnecessary, and just running a few experiments will give you all the insights you need. To this end, our main message is that there are well-working alternatives in the form of existing tuning algorithms for which it has been shown that they can to the tuning job much better than we can, with moderate costs. More importantly, such a new practice will enable better founded experimental comparisons, which we call competitive testing.

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".)

Further to improved competitive testing, the wide adoption of parameter tuners would enable better evolutionary algorithm (and other meta-heuristics) design. As mentioned before, using tuning algorithms one can not 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, which we call scientific testing. 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 (Section 3.3) EA, especially if its parameters are very tolerant (Section 3.3). 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 problem instance every day. As explained in [43, 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. 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 entire evolutionary computing community.

Regarding the current state-of-the-art in tuning algorithms, there is good and bad news. The good news is that most of them are well capable of doing their job. Three case studies showed the added value of using REVAC, REVAC++ and Bonesa for competitive and scientific testing. Each of them, and the other state-of-the-art tuners has its own specialties, some provide better information on tolerance, while others are specialized in evaluating the applicability of an algorithm. Furthermore, although each has its own way of dealing with noise, and costly evaluations, if given enough time, they are capable of finding high performing parameter settings for all problem and algorithm combinations as shown in the performance comparison. We did however, observe a slightly better performance over the whole line by SMAC, and the new tuning algorithm we introduced, Bonesa. Which is remarkable since the latter one was designed with multi-problem tuning in mind.

Bonesa is, in essence, an iterative model based search procedure much like Sequential Parameter Optimization [9], or in general, any search method using surrogate models. Just as SPO, it is based on iteratively build models that are used to pre-assesses the quality of generated parameter vectors rather than testing them directly. Therefore, the improved performance of Bonesa with respect to SPO is most likely caused by the combining it with REVAC, and Gaussian filtering, a new feature in the field of parameter tuners. In contrast to most other approaches, Bonesa does not use reevaluation of parameter vectors nor explicit racing techniques, but uses the strong causality principle, stating that small changes to a parameter vector cause only small changes in its utility. Together with the modeling capabilities, and its ability to optimize on multiple objectives, such as as whole test-suite, makes it a very versatile tool for both scientific and competitive testing.

The bad news for tuners is their current adoption. Currently tuners are mainly used by the research groups that developed them. It might be argued that this situation is just a matter of ignorance by the users of evolutionary algorithms, which a change in attitude could resolve. However, currently most tuners and far from user-friendly, and setting up a tuning experiment might take more time and worries than users are willing to invest. Tuners need to work with a minimum user effort and provide the functionalities to store and present (visualize) all information necessary for a thorough analysis. The acceptance by the community and the obtainable benefits will be realized only if tuning methods are available on-line, are user-friendly, and assist the user in taking a more scientific testing approach.

Next to the use of tuners in competitive and scientific testing, we foresee their application also in other fields of research. Noisy function optimization is a field very close to that of parameter tuning. In essence, parameter tuning is just a subset of it, with specific characteristics such as big noise and expensive evaluations. Preliminary results in Section 4.2 show that parameter tuners can be very competitive to the current state-of-the-art and, on the single problem tested here, can be much more efficient.

The other field of research investigated here is parameter control. Ever since the introduction of the distinction, numerous evolutionary computing scientists and practitioners posed the question: "Which approach is better, parameter tuning or parameter control?" In the fourth chapter of this thesis, we described how parameter tuning can be used to develop problem-tailed control mechanisms for evolutionary algorithms. With this in mind, we can answer this simple question with "both", since their combination delivered a very powerful problem solver that performed better than tuning or control could have done on their own. The ease by which both were integrated in an existing algorithm gives them a very promising future. Parameter values need no longer to be static, but can be adapted to the specific stage of the algorithm and the specific problem at hand. For repetitive problems, this might lead to a new era in evolutionary algorithms.

But maybe the most important outlook is a fundamental one. Scientific testing can help formulate new research questions, identify possibilities for improvement of the algorithm, and increase the usability of it for other users. These results of tuning are much more valuable than merely finding the best possible setting for some parameter on a particular problem or even on a range of problems. This shift from unfair and competitive to scientific testing requires not only a change in methodology, but also in attitude. We need to focus more on getting to know our algorithms, rather than just developing them. Therefore, we end this thesis with the same hope and expectations of the future, that Hooker already had in 1995:

> "*rather than agonize over what are the best parameter settings, one runs controlled experiments in which many different parameter settings are used precisely in order to understand their effect on performance.*"
>
> – J.N. Hooker

APPENDIX A

The Magic BONESA Constant c

In Section 2.3 the scalar c is used for calculating the similarity ω between two parameter vectors. There, c was defined as the value that makes the average value of $\sum \omega$ equal to 50 if m points are random uniformly distributed in a l-dimensional hypercube. Here we will give our approximation of c given a certain size of the archive m, and certain number of parameters to be tuned l.

Since the average value should be equal to 50, this means that the expected value ($E(\omega)$) of the ω of two random points, should be equal to:

$$E(\omega) \quad = (50/m) \tag{A.1}$$

First, recall that we defined the similarity $\omega_{\bar{z},\bar{y}}$ as:

$$\omega_{\bar{z},\bar{y}} \quad = e^{c \cdot x(\bar{z},\bar{y})} \tag{A.2}$$

with $x(\bar{z},\bar{y})$ the normalized [0 1] distance between $\bar{z}$ and $\bar{y}$.

Note that the expected value $E(\omega)$ given a known c, and two random uniformly distributed points in a l-dimensional hypercube, is therefore equal to:

$$E(\omega) \quad = \int_0^1 e^{c \cdot x} \cdot F(x) \;\; dx \tag{A.3}$$

In which $F(x)$ is the probability distribution of observing a normalized distance x between two points in a l-dimensional hypercube. We can roughly approximate this distribution with the incomplete beta-distribution [32]:

$$F_{a,b}(x) \quad = \tfrac{1}{\beta(a,b)} \cdot x^{a-1} \cdot (1-x)^{b-1} \tag{A.4}$$

with the shape parameters a and b equal to:

$$ED \approx \sqrt{(1/6) * l} \cdot \sqrt{(1/3) \cdot (1 + 2 \cdot \sqrt{1 - (3/(5 * l))})}/\sqrt{l} \quad \text{(A.5)}$$

$$a = ED * (((ED * (1 - ED))/(0.0585/l)) - 1) \quad \text{(A.6)}$$

$$b = (1 - ED) * (((ED * (1 - ED))/(0.0585/l)) - 1) \quad \text{(A.7)}$$

In which ED is an approximation [2] of the expected normalized distance between two points in a l-dimensional hypercube $[0\ 1]^l$.

Therefore we need to solve the following equation in order to find c:

$$(50/m) = \int_0^1 e^{c \cdot x} \cdot \tfrac{1}{\beta(a,b)} \cdot x^{a-1} \cdot (1 - x)^{b-1} \ dx \quad \text{(A.8)}$$

which can be rewritten to:

$$(50/m) \cdot \beta(a, b) = \tfrac{\Gamma(a)}{\Gamma(a)} \cdot \tfrac{\Gamma(b)}{\Gamma(b)} \cdot \int_0^1 e^{c \cdot x} \cdot x^{a-1} \cdot (1 - x)^{b-1} \quad \text{(A.9)}$$

The right hand side of Equation A.9 is known as the hypergeometric ${}_1F_1$ regularized function, with parameters a, a+b, and c. And therefore Equation A.9 is equivalent to:

$$(50/m) \cdot \beta(a, b) = \Gamma(a) \cdot \Gamma(b) \cdot {}_1F_1(a, a + b, c)dx \quad \text{(A.10)}$$

Which can be easily solved, since ${}_1F_1$ is implemented in most mathematical packages such as Matlab, R, or Mathematica and can be numerically approximated.

APPENDIX B

BONESA: the Toolbox

Bonesa was created to offer a user-friendly alternative to the current parameter tuning packages. The objective was to allow users to create and start parameter tuning sessions with only a few clicks and without any additional knowledge or programming skills.

However, Bonesa offers more than just a user-friendly interface to find the best performing parameter values. In [71] the term *scientific testing* was introduced, as opposed to *competitive testing*. Where competitive testing refers to simply finding the best parameters for a specific algorithm on a specific problem, scientific tuning refers to the process of understanding the influence of parameters and the interactions between parameter-values and the problem at hand. For this, a specific multi-target parameter tuning algorithm is incorporated that is able to tune on multiple performance measures, such as speed, quality and performance, and on multiple problems at the same time. This can be any set of problems, for example a benchmark test suite.

Figure 41: *BONESA splash screen*

My means of a wizard, Bonesa will guide you through the setup-process, which consists of four stages, namely:

i. **How can your algorithm be started?**
This can either be via the commandline, or directly by calling the run() function of a Java class.

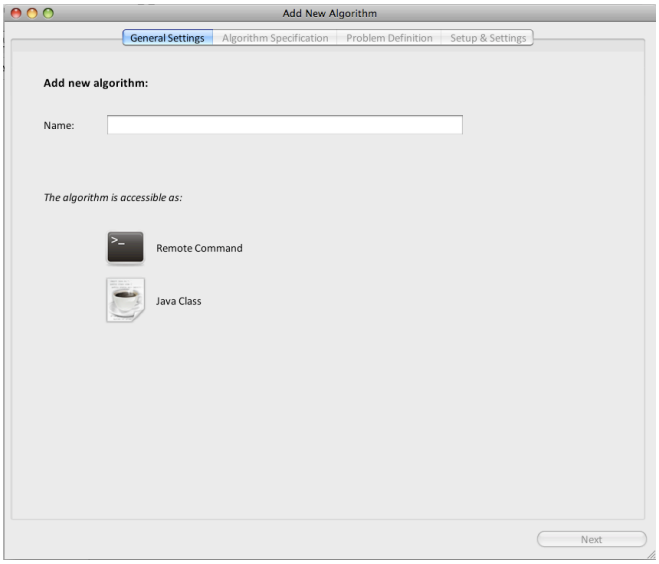


Figure 42: *How can your algorithm be started?*

ii. **Which parameter needs to be set?**
Bonesa can tune any number of parameters, however, these need to be integer or real valued, and bounded to a region of interest

iii. **What need to be optimized?**
For example: the performance on problem A, the performance on problem B, speed, accuracy or all of them at the same time. Since Bonesa is not limited to a single objective, you can tune on any number of targets.

iv. **What is the focus of the session?**
The user can choose any value on the gliding scale between focusing on finding the best parameter values, or exploring the parameter space.

After these are defined, the tuning-procedure can be started. Since parameter-tuning is a highly parallelizable process, Bonesa allows for distributed execution. For this, it uses the IBIS framework[141], which is a highly advanced and efficient Java-based platform for distributed computing. Starting such a distributed session is therefore

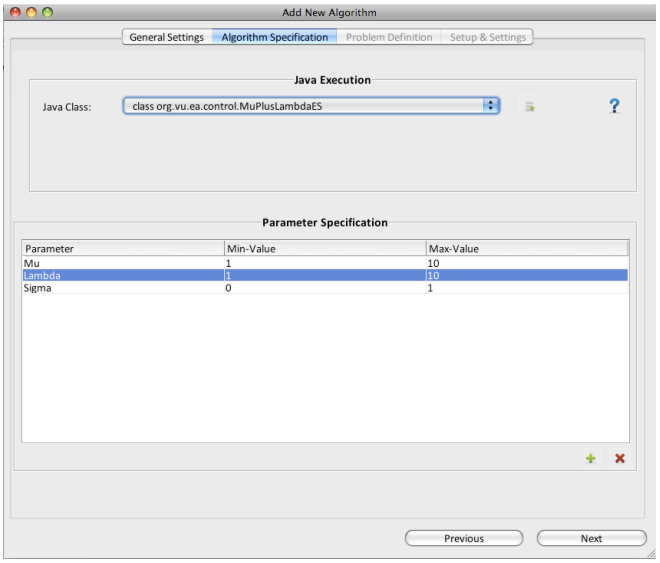


Figure 43: *Which parameter needs to be set?*

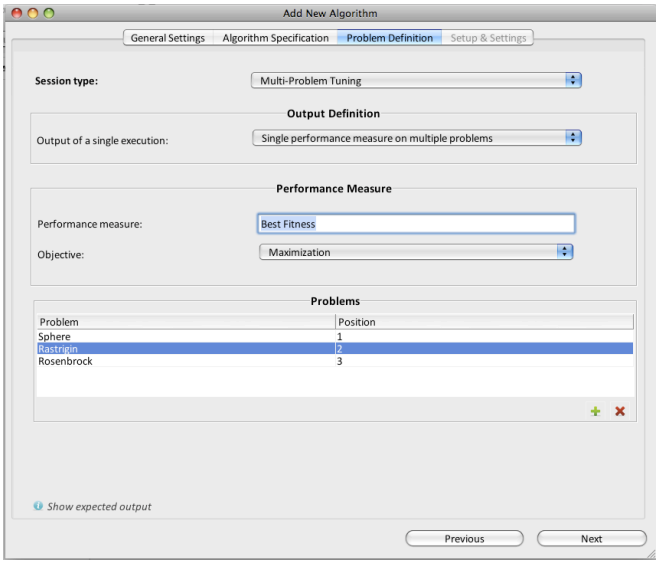


Figure 44: *What need to be optimized?*

just as easy to setup as a single-machine session, since all copies of the same project will automatically join an already running session when executed. So with only a few minutes of work, a whole cluster can be instructed to cooperate in a large-scale parameter tuning session. And, since Bonesa is purely written in Java, it works cross-platform and on any common operating system.

Furthermore, machines –for example a desktop-pc– can also be dedicated to simply monitor the session. This machine will automatically receive the latest information,

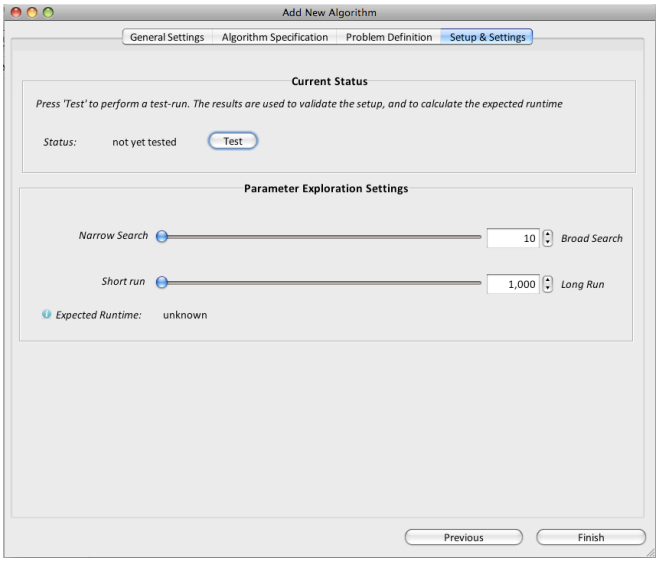


Figure 45: *What is the focus of the session?*

and can display a whole range of visualizations to show the progress. The standard package of Bonesa includes a wide variety of graphs and tables to study the performance landscape of your algorithm. These include robustness and principle component analysis of the algorithm parameters, confidence intervals of the expected performance, and other visualizations to illustrate the influence of parameter values on the performance. However, for those in need of other visualizations, this set is easily extendable by implementing a Java interface and placing it in the plugins-directory, or by including R scripts.

Furthermore, if a tuning session is terminated by the user, stop-condition, or is terminated due to computer-failures, it can be continued at any time without a loss of progress. In case of a distributed experiment, any computer (except the host) can even leave (or join) the session without disrupting the process, making it a fast, robust, easy package for tuning and studying the parameters of any metaheuristic. It has already been successfully applied in a wide range of fields, from robotics to evolutionary algorithms, and from data mining to adaptive cruise control, and has shown to be a very effective tool for both scientific and competitive testing.

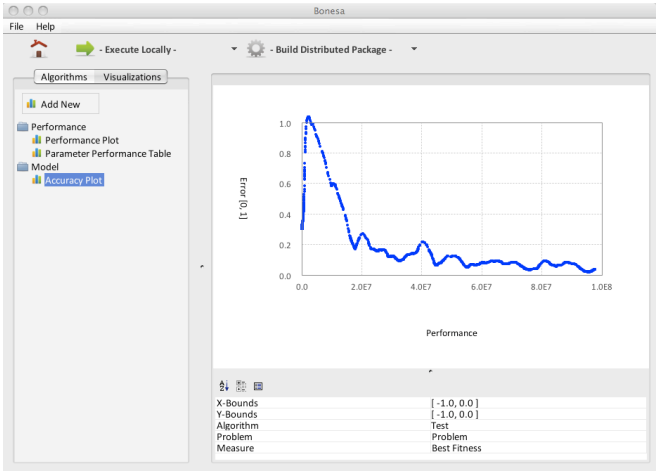


Figure 46: *Example of a visualization in BONESA*

List of Tables

List of Figures

Bibliography

[1] Belarmino Adenso-Diaz and Manuel Laguna. Fine-tuning of algorithms using fractional experimental designs and local search. *Oper. Res.*, 54(1):99–114, 2006.

[2] R. S. Anderssen, R. P. Brent, D. J. Daley, and A. P. Moran. Concerning $\int_0^1 \cdots \int_0^1 ({x_1}^2 + \cdots + {x_k}^2)^{1/2} dx_1 \ldots dx_k$ and a taylor series method. *Journal of Applied Mathematics*, 30:22–30, 1976.

[3] Jaroslaw Arabas, Zbigniew Michalewicz, and Jan J. Mulawka. Gavaps - a genetic algorithm with varying population size. In *International Conference on Evolutionary Computation*, pages 73–78, 1994.

[4] A. Auger and N. Hansen. A restart CMA evolution strategy with increasing population size. In CEC-2005 [27], pages 1769–1776.

[5] Anne Auger, Dimo Brockhoff, and Nikolaus Hansen. Comparing the (1+1)-cma-es with a mirrored (1+2)-cma-es with sequential selection on the noiseless bbob-2010 testbed. In Pelikan and Branke [110], pages 1543–1550.

[6] T. Bäck. Parallel optimization of evolutionary algorithms. In Y Davidor, H.-P Schwefel, and R Männer, editors, *Proceedings of the 3rd Conference on Parallel Problem Solving from Nature*, number 866 in Lecture Notes in Computer Science, pages 418– 427. Springer, Berlin, Heidelberg, New York, 1994.

[7] T Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, Oxford, UK, 1996.

[8] Prasanna Balaprakash, Mauro Birattari, and Thomas Stutzle. Improvement strategies for the F-Race algorithm: Sampling design and iterative refinementace algorithm: Sampling design and iterative refinement. In Thomas Bartz-Beielstein, Maria Blesa Aguilera, Christian Blum, Boris Naujoks, Andrea Roli, Günter Rudolph, and Michael Sampels, editors, *Hybrid Metaheuristics*, volume 4771 of *Lecture Notes in Computer Science*, pages 108–122. Springer Berlin / Heidelberg, 2007.

[9] T. Bartz-Beielstein, C.W.G. Lasarczyk, and M. Preuss. Sequential parameter optimization. In CEC-2005 [27], pages 773–780 Vol.1.

[10] T. Bartz-Beielstein, K.E. Parsopoulos, and M.N. Vrahatis. Analysis of Particle Swarm Optimization Using Computational Statistics. In Chalkis, editor, *Proceedings of the International Conference of Numerical Analysis and Applied Mathematics (ICNAAM 2004)*, pages 34–37. Wiley, 2004.

[11] Thomas Bartz-Beielstein. Experimental Analysis of Evolution Strategies: Overview and Comprehensive Introduction. Technical Report Reihe CI 157/03, SFB 531, Universität Dortmund, Dortmund, Germany, 2003.

[12] Thomas Bartz-Beielstein. *New Experimentalism Applied to Evolutionary Computation*. PhD thesis, Universität Dortmund, 2005.

[13] Thomas Bartz-Beielstein. Sequential Parameter Optimization—An Annotated Bibliography. Technical Report 04/2010, Institute of Computer Science, Faculty of Computer Science and Engineering Science, Cologne University of Applied Sciences, Germany, 2010.

[14] Thomas Bartz-Beielstein, Martina Friese, Martin Zaefferer, Boris Naujoks, Oliver Flasch, Wolfgang Konen, and Patrick Koch. Noisy optimization with sequential parameter optimization and optimal computational budget allocation. In em et al [46], pages 119–120.

[15] Thomas Bartz-Beielstein, Christian Lasarczyk, and Mike Preuss. The Sequential Parameter Optimization Toolbox. In Thomas Bartz-Beielstein, Marco Chiarandini, Luis Paquete, and Mike Preuss, editors, *Empirical Methods for the Analysis of Optimization Algorithms*, pages 337–360. Springer, Berlin, Heidelberg, New York, 2009.

[16] Thomas Bartz-Beielstein and Sandor 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, March 2004.

[17] Thomas Bartz-Beielstein and Mike Preuss. Considerations of Budget Allocation for Sequential Parameter Optimization (SPO). In L. Paquete et al., editors, *Workshop on Empirical Methods for the Analysis of Algorithms, Proceedings*, pages 35–40, Reykjavik, Iceland, 2006. Online Proceedings.

[18] Thomas Bartz-Beielstein and Mike Preuss. Experimental research in evolutionary computation. In Peter A. N. Bosman, Tina Yu, and Anikó Ekárt, editors, *GECCO (Companion)*, pages 3001–3020, New York, NY, USA, 2007. ACM.

[19] R. E. Bechhofer, C. W. Dunnett, D. M. Goldsman, and 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*, B19:971–1006, 1990.

[20] Mauro Birattari. *Tuning Metaheuristics*. Springer, 2005.

[21] Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle. F-Race and iterated F-Race: An overview. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 311–336. Springer, 2010. To appear.

[22] Amine Boumaza. On the evolution of artificial tetris players. In *Proceedings of the 5th international conference on Computational Intelligence and Games*, CIG'09, pages 387–393, Piscataway, NJ, USA, 2009. IEEE Press.

[23] Amine Boumaza. Designing artificial tetris players with evolution strategies and racing. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, GECCO '11, pages 117–118, New York, NY, USA, 2011. ACM.

[24] Jürgen Branke, E. Chick, Stephen, and Christian 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*, pages 708–717. Winter Simulation Conference, 2005.

[25] Jürgen Branke, Christian Schmidt, and Hartmut Schmec. Efficient fitness estimation in noisy environments. In *Proceedings of Genetic and Evolutionary Computation*, pages 243–250, 2001.

[26] Nicolas Bredeche, Evert Haasdijk, and A. E. Eiben. On-line, on-board evolution of robot controllers. In P. Collet *et al*, editor, *Artificial Evolution*, Lecture Notes in Computer Science, pages 110–121. Springer, 2009.

[27] *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, Edinburgh, UK, 2-5 September 2005. IEEE Press.

[28] M. Chiarandini, L. Paquete, M. Preuss, and E. Ridge. Experiments on metaheuristics: methodological overview and open issues. Technical Report DMF-2007-03-003, The Danish Mathematical Society, 2007.

[29] David Johan Christensen, Alexander Spröwitz, and Auke Jan Ijspeert. Distributed online learning of central pattern generators in modular robots. In Stéphane

Doncieux *et al*, editor, *From Animals to Animats 11*, volume 6226 of *Lecture Notes in Computer Science*, pages 402–412. Springer Berlin / Heidelberg, 2010.

[30] Steven P. Coy, Bruce L. Golden, George C. Runger, and Edward A. Wasil. Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics*, 7:77–97, 2001.

[31] A. Czarn, C. MacNish, K. Vijayan, B.A. Turlach, and R. Gupta. Statistical exploratory analysis of genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 8(4):405–421, 2004.

[32] Nagaraja H. N David, H. A. *Order Statistics*. Wiley, New Jersey, 2003.

[33] K. De Jong. Parameter setting in EAs: a 30 year perspective. In Lobo et al. [90], pages 1–18.

[34] Kalyanmoy Deb and Himanshu Gupta. Searching for robust pareto-optimal solutions in multi-objective optimization. *Evolutionary Multi-Criterion Optimization*, pages 150–164, 2005.

[35] K.D. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm : NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, August 2002.

[36] Giacomo di Tollo, Frédéric Lardeux, Jorge Maturana, and Frédéric Saubion. From adaptive to more dynamic control in evolutionary algorithms. In *Proceedings of the 11th European conference on Evolutionary computation in combinatorial optimization*, EvoCOP'11, pages 130–141. Springer-Verlag, 2011.

[37] Johann 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*, pages 2197–2200, New York, NY, USA, 2009. ACM.

[38] A. E. Eiben and M. Jelasity. A critical note on Experimental Research Methodology in Experimental research methodology in EC. In *2002 Congress on Evolutionary Computation (CEC'2002)*, pages 582–587. IEEE Press, Piscataway, NJ, 2002.

[39] A. E. Eiben, E. Marchiori, and V. A. Valko. Evolutionary algorithms with on-the-fly population size adjustment. In Yao *et al* [146], pages 41–50.

[40] A. E. Eiben, Z. Michalewicz, M. Schoenauer, and J. E. Smith. Parameter control in evolutionary algorithms. In Lobo et al. [90], pages 19–46.

[41] A. E. Eiben and S. K. Smit. Evolutionary algorithm parameters and methods to tune them. In E. Monfroy Y. Hamadi and F. Saubion, editors, *Autonomous Search*, chapter to appear. Springer, 2011.

[42] A. E. Eiben and S. K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011.

[43] A. E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer, Berlin Heidelberg, 2003.

[44] A.E. Eiben, M.C. Schut, and A.R. de Wilde. Is self-adaptation of selection pressure and population size possible? - a case study. 4193:900–909, 2006.

[45] M.A. El-Beltagy, P.B. Nair, and A.J. Keane. Metamodeling techniques for evolutionary optimization of computationally expensive problems: Promises and limitations. In W Banzhaf *et al*, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*, pages 196–203. Morgan Kaufmann, San Francisco, 1999.

[46] Natalio Krasnogor em et al, editor. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2011)*, Dublin, Ireland, 12-16 July 2011. ACM.

[47] Hamidreza Eskandari and Christopher D. Geiger. Evolutionary multiobjective optimization in noisy problem environments. *Journal of Heuristics*, 15(6):559–595, 2009.

[48] J.M. Fitzpatrick and J.J. Grefenstette. Genetic algorithms in noisy environments. *Machine Learning*, 3:101–120, 1988.

[49] T. C. Fogarty. Varying the probability of mutation in the genetic algorithm. In *Proceedings of the third international conference on Genetic algorithms*, pages 104–109, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[50] G. Francesca, P. Pellegrini, T. Stützle, and M. Birattari. Off-line and on-line tuning: A study on operator selection for a memetic algorithm applied to the qap. In P. Merz and J.-K. Hao, editors, *Evolutionary Computation in Combinatorial Optimization, 11th European Conference, EvoCOP 2011*, 2011.

[51] O. François and C. Lavergne. Design of Evolutionary Algorithms—A Statistical Perspective. *IEEE Transactions on Evolutionary Computation*, 5(2):129–148, 2001.

[52] B. Freisleben and M. Hartfelder. Optimization of genetic algorithms by genetic algorithms. In R.F. Albrecht, C.R. Reeves, and N.C. Steele, editors, *Artificial Neural Networks and Genetic Algorithms*, pages 392–399. Springer, 1993.

[53] Salvador García, Daniel Molina, Manuel Lozano, and Francisco Herrera. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the cec'2005 special session on real parameter optimization. *J. Heuristics*, 15(6):617–644, 2009.

[54] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1989.

[55] David Goldsman, Barry L. Nelson, and Bruce Schmeiser. Methods for selecting the best system. In *WSC '91: Proceedings of the 23rd conference on Winter simulation*, pages 177–186, Washington, DC, USA, 1991. IEEE Computer Society.

[56] J.J Greffenstette. Optimisation of Control Parameters for Genetic Algorithms. In A.P. Sage, editor, *IEEE Transactions on Systems, Man and Cybernetics*, volume 16, pages 122–128. IEEE Press, Piscataway, NJ, 1986.

[57] Evert Haasdijk, A. E. Eiben, and Giorgos Karafotias. On-line evolution of robot controllers by an encapsulated evolution strategy. In CEC-2010 [79].

[58] Evert Haasdijk, S.K. Smit, and A.E. Eiben. Exploratory analysis of an on-line evolutionary algorithm for robotics. *(under review)*, 2012.

[59] Evert Haasdijk, Atta ul Qayyum Arif, and A. E. Eiben. Racing to improve on-line, on-board evolutionary robotics. In em et al [46], pages 187–194.

[60] N. Hansen, A. Auger, S. Finck, and R. Ros. Real-parameter black-box optimization benchmarking 2010: Experimental setup. Technical Report RR-7215, INRIA, 2010.

[61] N. Hansen and S. Kern. Evaluating the CMA evolution strategy on multimodal test functions. In Yao *et al* [146], pages 282–291.

[62] N. Hansen, S.D. Muller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

[63] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of the 1996 IEEE Conference on Evolutionary Computation*, pages 312–317. IEEE Press, Piscataway, NJ, 1996.

[64] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[65] Nikolaus Hansen. References to CMA-ES applications, 2005.

[66] R. Haralick and L. Shapiro. *Computer and Robot Vision*, volume 1, chapter 7. Addison-Wesley Publishing Company, 1992.

[67] G. R. Harik and F. G. Lobo. A parameter-less genetic algorithm. In *In W. Banzhaf et al., editors, Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99, San Francisco, CA. Morgan Kaufmann*, pages 258–265, 1999.

[68] W.E. Hart and R.K. Belew. Optimizing an arbitrary function is hard for the genetic algorithm. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of theFourthInternational Conference on Genetic Algorithms(ICGA '91)*, pages 190–195, San Mateo CA, 13–16 1991. Morgan Kaufmann Publishers, Inc.

[69] Y. Hochberg and A. C. Tamhane. *Multiple comparison procedures*. John Wiley & Sons, Inc., New York, NY, USA, 1987.

[70] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[71] J.N. Hooker. Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1:33–42, 1995.

[72] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann, 2004.

[73] E. Hughes. Evolutionary multi-objective ranking with uncertainty and noise. In *EMO '01: Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*, pages 329–343, London, UK, 2001. Springer-Verlag.

[74] F. Hutter, T. Bartz-Beielstein, H.H. Hoos, K. Leyton-Brown, and K.P. Murphy. Sequential model-based parameter optimisation: an experimental investigation of automated and interactive approaches. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors, *Empirical Methods for the Analysis of Optimization Algorithms*, chapter 15, pages 361–411. Springer, 2010.

[75] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proc. of LION-5*, page 507523, 2011.

[76] F. Hutter, H. H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. In *Proc. of the Twenty-Second Conference on Artifical Intelligence (AAAI '07)*, pages 1152–1157, 2007.

[77] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Bayesian optimization with censored response data. In *NIPS workshop on Bayesian Optimization, Sequential Experimental Design, and Bandits*, 2011. Published online.

[78] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stutzle. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, October 2009.

[79] IEEE Computational Intelligence Society. *Proceedings of the 2010 IEEE Congress on Evolutionary Computation*, Barcelona, Spain, 2010. IEEE Press.

[80] Yaochu Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing*, 9(1):3–12, 2005.

[81] Yaochu Jin and J. Branke. Evolutionary optimization in uncertain environments-a survey. *Evolutionary Computation, IEEE Transactions on*, 9(3):303–317, 2005.

[82] D. S. Johnson. A theoretician's guide to the experimental analysis of algorithms. In M. H. Goldwasser, D. S. Johnson, and C. C. McGeoch, editors, *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, pages 215–250. American Mathematical Society, Providence, 2002.

[83] Terry Jones. Crossover, macromutation, and population-based search. In L.J Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 73–80. Morgan Kaufmann, San Francisco, 1995.

[84] Giorgos Karafotias, Evert Haasdijk, and A. E. Eiben. An algorithm for distributed on-line, on-board evolutionary robotics. In em et al [46], pages 171–178.

[85] Giorgos Karafotias, S.K. Smit, and A.E. Eiben. A generic approach to parameter control. In to appear, editor, *Proceedings of Evo* 2012*, 2012.

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

[87] V K Koumousis and C P Katsaras. A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance. *IEEE Transactions on Evolutionary Computation*, 10(1):19–28, 2006.

[88] Christian W. G. Lasarczyk. *Genetische Programmierung einer algorithmischen Chemie*. PhD thesis, Technische Universiteit Dortmund, 2007.

[89] Michael A. Lee and Hideyuki Takagi. Dynamic control of genetic algorithms using fuzzy logic techniques. In S Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 76–83. Morgan Kaufmann, San Francisco, 1993.

[90] F.G. Lobo, C.F. Lima, and Z. Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*. Springer, 2007.

[91] MendAmar Majig and Masao Fukushima. Adaptive fitness function for evolutionary algorithm and its applications. *Informatics Research for Development of Knowledge Society Infrastructure, International Conference on*, pages 119–124, 2008.

[92] O. Maron and A. Moore. The racing algorithm: Model selection for lazy learners. In *Artificial Intelligence Review*, volume 11, pages 193–225. Kluwer Academic Publishers Norwell, MA, USA, April 1997.

[93] A. Martinoli. *Swarm Intelligence in Autonomous Collective Robotics from Tools to the Analysis and Synthesis of Distributed Control Strategies*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 1999.

[94] Jorge Maturana, F. Lardeux, and Frederic Saubion. Controlling behavioral and structural parameters in Evolutionary Algorithms. In *International Conference on Artificial Evolution (EA'2009)*, volume 5926 of *Lecture Notes in Computer Science*, pages 110–121. Springer, 2009.

[95] Jorge Maturana and Frdric Saubion. On the design of adaptive control strategies for evolutionary algorithms. In *Artificial Evolution*, volume 4926 of *Lecture Notes in Computer Science*, pages 303–315. Springer, 2008.

[96] Michael Meissner, Michael Schmuker, and Gisbert Schneider. Optimized particle swarm optimization (OPSO) and its application to artificial neural network training. *BMC Bioinformatics*, 7:125, 2006.

[97] R.E. Mercer and J.R. Sampson. Adaptive search using a reproductive metaplan. *Kybernetes*, 7:215–228, 1978.

[98] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. Exploratory landscape analysis. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 829–836, New York, NY, USA, 2011. ACM.

[99] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs (3nd, extended ed.)*. Springer-Verlag New York, Inc., New York, NY, USA, 1996.

[100] Brad L. Miller and David E. Goldberg. Genetic algorithms, selection schemes, and the varying effects of noise. *Evol. Comput.*, 4:113–131, June 1996.

[101] H. Mühlenbein and R. Höns. The Estimation of Distributions and the Minimum Relative Entropy Principle. *Evolutionary Computation*, 13(1):1–27, 2005.

[102] Richard Myers and Edwin R. Hancock. Empirical modelling of genetic algorithms. *Evolutionary Computation*, 9(4):461–493, 2001.

[103] V. Nannen and A. E. Eiben. Efficient Relevance Estimation and Value Calibration of Evolutionary Algorithm Parameters. In *IEEE Congress on Evolutionary Computation*, pages 103–110. IEEE, 2007.

[104] V. Nannen and A. E. Eiben. Relevance Estimation and Value Calibration of Evolutionary Algorithm Parameters. In Manuela M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1034–1039. Hyderabad, India, 2007.

[105] V. Nannen, S. K. Smit, and A. E. Eiben. Costs and benefits of tuning parameters of evolutionary algorithms. In Rudolph *et al* [121], pages 528–538.

[106] Stefano Nolfi and Dario Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press, Cambridge, MA, 2000.

[107] Andreas Ostermeier, Andreas Gawelczyk, and Nikolaus Hansen. A derandomized approach to self adaptation of evolution strategies. *Evolutionary Computation*, 2(4):369–380, 1994.

[108] Magnus Erik Hvass Pedersen and et al. Parameter tuning versus adaptation: proof of principle study on differential evolution. Technical report, Hvass Laboratories, 2008.

[109] M.E.H. Pedersen and A.J. Chipperfield. Simplifying particle swarm optimization. *Applied Soft Computing*, 10(2):618–628, 2010.

[110] Martin Pelikan and Jürgen Branke, editors. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2010)*. ACM, 2010.

[111] Paola Pellegrini, Thomas Stützle, and Mauro Birattari. Off-line vs. on-line tuning: A study on max–min ant system for the tsp. In Marco Dorigo, Mauro

Birattari, Gianni Di Caro, René Doursat, Andries Engelbrecht, Dario Floreano, Luca Gambardella, Roderich Groß, Erol Sahin, Hiroki Sayama, and Thomas Stützle, editors, *Swarm Intelligence*, volume 6234 of *Lecture Notes in Computer Science*, pages 239–250. Springer Berlin / Heidelberg, 2010.

[112] Mike Preuss. Adaptability of algorithms for real-valued optimization. In Mario Giacobini et al., editors, *Applications of Evolutionary Computing, EvoWorkshops 2009. Proceedings*, volume 5484 of *Lecture Notes in Computer Science*, pages 665–674, Berlin, 2009. Springer.

[113] K. Price, R. Storn, and J. Lampinen. *Differential Evolution - A Practical Approach to Global Optimization*. Springer, 2005.

[114] A.K. Qin and P.N. Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In CEC-2005 [27], pages 1785– 1791.

[115] I.C.O. Ramos, M.C. Goldbarg, E.G. Goldbarg, and A.D.D. Neto. Logistic regression for parameter tuning on an evolutionary algorithm. In CEC-2005 [27], pages 1061–1068.

[116] I. Rechenberg. *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution*. Fromman-Hozlboog Verlag, Stuttgart, 1973.

[117] Craig W. Reynolds. Evolution of corridor following behavior in a noisy world. In *SAB94: Proceedings of the third international conference on Simulation of adaptive behavior : from animals to animats 3*, pages 402–410, Cambridge, MA, USA, 1994. MIT Press.

[118] Enda Ridge and Daniel Kudenko. Sequential experiment designs for screening and tuning parameters of stochastic heuristics. In *In: Workshop on Empirical Methods for the Analysis of Algorithms, Reykjavik, Iceland*, pages 27–34. Online Proceedings, 2006.

[119] Charlotte J. Rietveld, Gijs Hendrix, Frank Berkers, Nadine N. Croes, and S.K. Smit. Avoiding simplification strategies by introducing Multi-Objectiveness in real world problems. In *The Tenth International Conference on Intelligent System Design and Applications (ISDA 2010)*, Cairo, Egypt, 2010.

[120] Yosef Rinott. On two-stage selection procedures and related probability-inequalities. *Communications in Statistics - Theory and Methods*, 7(8):799 – 811, 1978.

[121] Günter Rudolph *et al*, editor. *Parallel Problem Solving from Nature - PPSN X, 10th International Conference Dortmund, Germany, September 13-17, 2008, Proceedings*, volume 5199 of *Lecture Notes in Computer Science*. Springer, 2008.

[122] Thomas Philip Runarsson. Constrained evolutionary optimization by approximate ranking and surrogate models. In *Parallel Problem Solving from Nature (PPSN VIII)*, pages 401–408. Springer-Verlag, 2004.

[123] J.D. Schaffer, R.A. Caruana, L.J. Eshelman, and 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*, pages 51–60, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[124] B.W. Schmeiser. Simulation experiments. *Handbooks in Operations Research and Management Science*, 2:295–330, 1990.

[125] Nicol N. Schraudolph and Richard K. Belew. Dynamic parameter encoding for genetic algorithms. *Machine Learning*, 9:9–21, 1992.

[126] R. N. Shepard. Toward a universal law of generalization for psychological science. *Science*, 237(4820):1317–1323, 1987.

[127] S. K. Smit and A. E. Eiben. Comparing parameter tuning methods for evolutionary algorithms. In *Proceedings of the 2009 IEEE Congress on Evolutionary Computation*, pages 399–406, Trondheim, May 18-21 2009. IEEE Press.

[128] S. K. Smit and A. E. Eiben. Beating the 'world champion' evolutionary algorithm via revac tuning. In CEC-2010 [79], pages 1–8.

[129] S. K. Smit and A. E. Eiben. Parameter tuning of evolutionary algorithms: Generalist vs. specialist. In C. et al. Di Chio, editor, *Applications of Evolutionary Computation*, volume 6024 of *Lecture Notes in Computer Science*, pages 542–551. Springer, 2010.

[130] S. K. Smit and A. E. Eiben. Using entropy for parameter analysis of evolutionary algorithms. In Thomas Bartz-Beielstein, Marco Chiarandini, Luís Paquete, and Mike Preuss, editors, *Empirical Methods for the Analysis of Optimization Algorithms*, Natural Computing Series, pages 287–310. Springer, 2010.

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

[132] S.K. Smit and A.E. Eiben. Multi-problem parameter tuning using bonesa. In JK Hao, P Legrand, P Collet, N Monmarché, E Lutton, and M Schoenauer, editors, *Artificial Evolution*, pages 222–233, 2011.

[133] S.K. Smit and A.E. Eiben. Population diversity index: A new measure for population diversity. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2011.

[134] R.E. Smith and E Smuda. Adaptively resizing populations: Algorithm, analysis and first results. *Complex Systems*, 9(1):47–72, 1995.

[135] William M. Spears. Adapting crossover in evolutionary algorithms. In *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 367–384. MIT Press, 1995.

[136] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[137] P.N. Suganthan, N. Hansen, J.J. Liang, K.Deb, Y.-P. Chen, A. Auger, and S. Tiwari. Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization. Technical report, Nanyang Technological University, 2005.

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

[139] Jonathan Fieldsend University and Jonathan E. Fieldsend. Multi-objective optimisation in the presence of uncertainty. In CEC-2005 [27], pages 476–483.

[140] P. Vajda, A. E. Eiben, and W. Hordijk. Parameter Control Methods for Selection Operators in Genetic Algorithms. In Rudolph *et al* [121], pages 620–630.

[141] Rob V. van Nieuwpoort, Jason Maassen, Gosia Wrzesinska, Rutger Hofman, Ceriel Jacobs, Thilo Kielmann, and Henri E. Bal. Ibis: a flexible and efficient Java based grid programming environment. *Concurrency and Computation: Practice and Experience*, 17(7-8):1079–1107, June 2005.

[142] Thomas Voß, Heike Trautmann, and Christian Igel. New uncertainty handling strategies in multi-objective evolutionary optimization. In Robert Schaefer, Carlos Cotta, Joanna Kolodziej, and Günter Rudolph, editors, *PPSN*, volume 6239 of *Lecture Notes in Computer Science*, pages 260–269. Springer, 2011.

[143] B. L. Welch. The generalization of "student's" problem when several different population variances are involved. *Biometrika*, 34(1–2):28–35, 1947.

[144] G.J. Woldinga. *Xenophons Symposium*. PhD thesis, Vrije Universiteit te Amsterdam, 1938.

[145] Y.-Y. Wong, K.-H. Lee, K.-S. Leung, and C.-W. Ho. A novel approach in parameter adaptation and diversity maintenance for genetic algorithms. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 7:506–515, 2003.

[146] Xin Yao *et al*, editor. *Parallel Problem Solving from Nature - PPSN VIII, 8th International Conference, Birmingham, UK, September 18-22, 2004, Proceedings*, volume 3242 of *Lecture Notes in Computer Science*. Springer, 2004.

[147] B. Yuan and M. Gallagher. Combining Meta-EAs and Racing for Difficult EA Parameter Tuning Tasks. In Lobo et al. [90], pages 121–142.

[148] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001.

SIKS Dissertatiereeks

2009

2009-01 Rasa Jurgelenaite (RUN) *Symmetric Causal Independence Models*
2009-02 Willem Robert van Hage (VU) *Evaluating Ontology-Alignment Techniques*
2009-03 Hans Stol (UvT) *A Framework for Evidence-based Policy Making Using IT*
2009-04 Josephine Nabukenya (RUN) *Improving the Quality of Organisational Policy Making using ...*
2009-05 Sietse Overbeek (RUN) *Bridging Supply and Demand for Knowledge Intensive Tasks - Based on ...*
2009-06 Muhammad Subianto (UU) *Understanding Classification*
2009-07 Ronald Poppe (UT) *Discriminative Vision-Based Recovery and Recognition of Human Motion*
2009-08 Volker Nannen (VU) *Evolutionary Agent-Based Policy Analysis in Dynamic Environments*
2009-09 Benjamin Kanagwa (RUN) *Design, Discovery and Construction of Service-oriented Systems*
2009-10 Jan Wielemaker (UVA) *Logic programming for knowledge-intensive interactive applications*
2009-11 Alexander Boer (UVA) *Legal Theory, Sources of Law and the Semantic Web*
2009-12 Peter Massuthe (TUE) *Operating Guidelines for Services*
2009-13 Steven de Jong (UM) *Fairness in Multi-Agent Systems*
2009-14 Maksym Korotkiy (VU) *From ontology-enabled services to service-enabled ontologies*
2009-15 Rinke Hoekstra (UVA) *Ontology Representation - Design Patterns and Ontologies that Make Sense*
2009-16 Fritz Reul (UvT) *New Architectures in Computer Chess*
2009-17 Laurens van der Maaten (UvT) *Feature Extraction from Visual Data*
2009-18 Fabian Groffen (CWI) *Armada, An Evolving Database System*
2009-19 Valentin Robu (CWI) *Modeling Preferences, Strategic Reasoning and Collaboration in ...*
2009-20 Bob van der Vecht (UU) *Adjustable Autonomy: Controling Influences on Decision Making*
2009-21 Stijn Vanderlooy (UM) *Ranking and Reliable Classification*
2009-22 Pavel Serdyukov (UT) *Search For Expertise: Going beyond direct evidence*
2009-23 Peter Hofgesang (VU) *Modelling Web Usage in a Changing Environment*
2009-24 Annerieke Heuvelink (VUA) *Cognitive Models for Training Simulations*
2009-25 Alex van Ballegooij (CWI) *RAM: Array Database Management through Relational Mapping*
2009-26 Fernando Koch (UU) *An Agent-Based Model for the Development of Intelligent Mobile Services*
2009-27 Christian Glahn (OU) *Contextual Support of social Engagement and Reflection on the Web*
2009-28 Sander Evers (UT) *Sensor Data Management with Probabilistic Models*
2009-29 Stanislav Pokraev (UT) *Model-Driven Semantic Integration of Service-Oriented Applications*
2009-30 Marcin Zukowski (CWI) *Balancing vectorized query execution with bandwidth-optimized storage*
2009-31 Sofiya Katrenko (UVA) *A Closer Look at Learning Relations from Text*
2009-32 Rik Farenhorst (VU) and Remco de Boer (VU) *Architectural Knowledge Management*
2009-33 Khiet Truong (UT) *How Does Real Affect Affect Affect Recognition In Speech?*
2009-34 Inge van de Weerd (UU) *Advancing in Software Product Management: An Incremental Method ...*
2009-35 Wouter Koelewijn (UL) *Privacy en Politiegegevens; Over geautomatiseerde normatieve ...*
2009-36 Marco Kalz (OUN) *Placement Support for Learners in Learning Networks*
2009-37 Hendrik Drachsler (OUN) *Navigation Support for Learners in Informal Learning Networks*
2009-38 Riina Vuorikari (OU) *Tags and self-organisation: a metadata ecology for learning resources in ...*
2009-39 Christian Stahl (TUE) *Service Substitution – A Behavioral Approach Based on Petri Nets*
2009-40 Stephan Raaijmakers (UvT) M*ultinomial Language Learning: Investigations into the Geometry ...*
2009-41 Igor Berezhnyy (UvT) *Digital Analysis of Paintings*
2009-42 Toine Bogers *Recommender Systems for Social Bookmarking*
2009-43 Virginia Nunes Leal Franqueira (UT) *Finding Multi-step Attacks in Computer Networks using ...*
2009-44 Roberto Santana Tapia (UT) *Assessing Business-IT Alignment in Networked Organizations*
2009-45 Jilles Vreeken (UU) *Making Pattern Mining Useful*
2009-46 Loredana Afanasiev (UvA) *Querying XML: Benchmarks and Recursion*

2010

2010-01 Matthijs van Leeuwen (UU) *Patterns that Matter*
2010-02 Ingo Wassink (UT) *Work flows in Life Science*
2010-03 Joost Geurts (CWI) *A Document Engineering Model and Processing Framework for ...*
2010-04 Olga Kulyk (UT) *Do You Know What I Know? Situational Awareness of Co-located Teams in ...*
2010-05 Claudia Hauff (UT) *Predicting the Effectiveness of Queries and Retrieval Systems*
2010-06 Sander Bakkes (UvT) *Rapid Adaptation of Video Game AI*
2010-07 Wim Fikkert (UT) *Gesture interaction at a Distance*
2010-08 Krzysztof Siewicz (UL) *Towards an Improved Regulatory Framework of Free Software*
2010-09 Hugo Kielman (UL) *A Politiele gegevensverwerking en Privacy, Naar een effectieve ...*
2010-10 Rebecca Ong (UL) *Mobile Communication and Protection of Children*
2010-11 Adriaan Ter Mors (TUD) *The world according to MARP: Multi-Agent Route Planning*

2010-12 Susan van den Braak (UU) *Sensemaking software for crime analysis*
2010-13 Gianluigi Folino (RUN) *High Performance Data Mining using Bio-inspired techniques*
2010-14 Sander van Splunter (VU) *Automated Web Service Reconfiguration*
2010-15 Lianne Bodenstaff (UT) *Managing Dependency Relations in Inter-Organizational Models*
2010-16 Sicco Verwer (TUD) *Efficient Identification of Timed Automata, theory and practice*
2010-17 Spyros Kotoulas (VU) *Scalable Discovery of Networked Resources: Algorithms, ...*
2010-18 Charlotte Gerritsen (VU) *Caught in the Act: Investigating Crime by Agent-Based Simulation*
2010-19 Henriette Cramer (UvA) *People's Responses to Autonomous and Adaptive Systems*
2010-20 Ivo Swartjes (UT) *Whose Story Is It Anyway? How Improv Informs Agency and Authorship ...*
2010-21 Harold van Heerde (UT) *Privacy-aware data management by means of data degradation*
2010-22 Michiel Hildebrand (CWI) *End-user Support for Access to Heterogeneous Linked Data*
2010-23 Bas Steunebrink (UU) *The Logical Structure of Emotions*
2010-24 Dmytro Tykhonov *Designing Generic and Efficient Negotiation Strategies*
2010-25 Zulfiqar Ali Memon (VU) *Modelling Human-Awareness for Ambient Agents*
2010-26 Ying Zhang (CWI) *XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery ...*
2010-27 Marten Voulon (UL) *Automatisch contracteren*
2010-28 Arne Koopman (UU) *Characteristic Relational Patterns*
2010-29 Stratos Idreos(CWI) *Database Cracking: Towards Auto-tuning Database Kernels*
2010-30 Marieke van Erp (UvT) *Accessing Natural History - Discoveries in data cleaning, ...*
2010-31 Victor de Boer (UVA) *Ontology Enrichment from Heterogeneous Sources on the Web*
2010-32 Marcel Hiel (UvT) *An Adaptive Service Oriented Architecture: Automatically solving ...*
2010-33 Robin Aly (UT) *Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval*
2010-34 Teduh Dirgahayu (UT) *Interaction Design in Service Compositions*
2010-35 Dolf Trieschnigg (UT) *Proof of Concept: Concept-based Biomedical Information Retrieval*
2010-36 Jose Janssen (OU) *Paving the Way for Lifelong Learning; Facilitating competence ...*
2010-37 Niels Lohmann (TUE) *Correctness of services and their composition*
2010-38 Dirk Fahland (TUE) *From Scenarios to components*
2010-39 Ghazanfar Farooq Siddiqui (VU) *Integrative modeling of emotions in virtual agents*
2010-40 Mark van Assem (VU) *Converting and Integrating Vocabularies for the Semantic Web*
2010-41 Guillaume Chaslot (UM) *Monte-Carlo Tree Search*
2010-42 Sybren de Kinderen (VU) *Needs-driven service bundling in a multi-supplier setting ...*
2010-43 Peter van Kranenburg (UU) *A Computational Approach to Content-Based Retrieval of Folk ...*
2010-44 Pieter Bellekens (TUE) *An Approach towards Context-sensitive and User-adapted Access ...*
2010-45 Vasilios Andrikopoulos (UvT) *A theory and model for the evolution of software services*
2010-46 Vincent Pijpers (VU) *e3alignment: Exploring Inter-Organizational Business-ICT Alignment*
2010-47 Chen Li (UT) *Mining Process Model Variants: Challenges, Techniques, Examples*
2010-48 Milan Lovric (EUR) *Behavioral Finance and Agent-Based Artificial Markets*
2010-49 Jahn-Takeshi Saito (UM) *Solving difficult game positions*
2010-50 Bouke Huurnink (UVA) *Search in Audiovisual Broadcast Archives*
2010-51 Alia Khairia Amin (CWI) *Understanding and supporting information seeking tasks in ...*
2010-52 Peter-Paul van Maanen (VU) *Adaptive Support for Human-Computer Teams: Exploring the Use of ...*
2010-53 Edgar Meij (UVA) *Combining Concepts and Language Models for Information Access*

2011

2011-01 Botond Cseke (RUN) *Variational Algorithms for Bayesian Inference in Latent Gaussian Models*
2011-02 Nick Tinnemeier(UU) *Organizing Agent Organizations. Syntax and Operational Semantics of ...*
2011-03 Jan Martijn van der Werf (TUE) *Compositional Design and Verification of Component-Based ...*
2011-04 Hado van Hasselt (UU) *Insights in Reinforcement Learning; Formal analysis and empirical ...*
2011-05 Base van der Raadt (VU) *Enterprise Architecture Coming of Age - Increasing the Performance ...*
2011-06 Yiwen Wang (TUE) *Semantically-Enhanced Recommendations in Cultural Heritage*
2011-07 Yujia Cao (UT) *Multimodal Information Presentation for High Load Human Computer Interaction*
2011-08 Nieske Vergunst (UU) *BDI-based Generation of Robust Task-Oriented Dialogues*
2011-09 Tim de Jong (OU) *Contextualised Mobile Media for Learning*
2011-10 Bart Bogaert (UvT) *Cloud Content Contention*
2011-11 Dhaval Vyas (UT) *Designing for Awareness: An Experience-focused HCI Perspective*
2011-12 Carmen Bratosin (TUE) *Grid Architecture for Distributed Process Mining*
2011-13 Xiaoyu Mao (UvT) *Airport under Control. Multiagent Scheduling for Airport Ground Handling*
2011-14 Milan Lovric (EUR) *Behavioral Finance and Agent-Based Artificial Markets*
2011-15 Marijn Koolen (UvA) *The Meaning of Structure: the Value of Link Evidence for Information Retrieval*
2011-16 Maarten Schadd (UM) *Selective Search in Games of Different Complexity*
2011-17 Jiyin He (UVA) *Exploring Topic Structure: Coherence, Diversity and Relatedness*
2011-18 Mark Ponsen (UM) *Strategic Decision-Making in complex games*
2011-19 Ellen Rusman (OU) *The Mind's Eye on Personal Profiles*
2011-20 Qing Gu (VU) *Guiding service-oriented software engineering - A view-based approach*

2011-21 Linda Terlouw (TUD) *Modularization and Specification of Service-Oriented Systems*
2011-22 Junte Zhang (UVA) *System Evaluation of Archival Description and Access*
2011-23 Wouter Weerkamp (UVA) *Finding People and their Utterances in Social Media*
2011-24 Herwin van Welbergen (UT) *Behavior Generation for Interpersonal Coordination with Virtual ...*
2011-25 Syed Waqar ul Qounain Jaffry (VU) *Analysis and Validation of Models for Trust Dynamics*
2011-26 Matthijs Aart Pontier (VU) *Virtual Agents for Human Communication - Emotion Regulation and ...*
2011-27 Aniel Bhulai (VU) *Dynamic website optimization through autonomous management of design patterns*
2011-28 Rianne Kaptein (UVA) *Effective Focused Retrieval by Exploiting Query Context and Document Structure*
2011-29 Faisal Kamiran (TUE) *Discrimination-aware Classification*
2011-30 Egon van den Broek (UT) *Affective Signal Processing (ASP): Unraveling the mystery of emotions*
2011-31 Ludo Waltman (EUR) *Computational and Game-Theoretic Approaches for Modeling Bounded Rationality*
2011-32 Nees-Jan van Eck (EUR) *Methodological Advances in Bibliometric Mapping of Science*
2011-33 Tom van der Weide (UU) *Arguing to Motivate Decisions*
2011-34 Paolo Turrini (UU) *Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations*
2011-35 Maaike Harbers (UU) *Explaining Agent Behavior in Virtual Training*
2011-36 Erik van der Spek (UU) *Experiments in serious game design: a cognitive approach*
2011-37 Adriana Burlutiu (RUN) *Machine Learning for Pairwise Data, Applications for Preference ...*
2011-38 Nyree Lemmens (UM) *Bee-inspired Distributed Optimization*
2011-39 Joost Westra (UU) *Organizing Adaptation using Agents in Serious Games*
2011-40 Viktor Clerc (VU) *Architectural Knowledge Management in Global Software Development*
2011-41 Luan Ibraimi (UT) *Cryptographically Enforced Distributed Data Access Control*
2011-42 Michal Sindlar (UU) *Explaining Behavior through Mental State Attribution*
2011-43 Henk van der Schuur (UU) *Process Improvement through Software Operation Knowledge*
2011-44 Boris Reuderink (UT) *Robust Brain-Computer Interfaces*
2011-45 Herman Stehouwer (UvT) *Statistical Language Models for Alternative Sequence Selection*
2011-46 Beibei Hu (TUD) *Towards Contextualized Information Delivery: A Rule-based Architecture ...*
2011-47 Azizi Bin Ab Aziz (VU) *Exploring Computational Models for Intelligent Support of Persons ...*
2011-48 Mark Ter Maat (UT) *Response Selection and Turn-taking for a Sensitive Artificial Listening Agent*
2011-49 Andreea Niculescu (UT) *Conversational interfaces for task-oriented spoken dialogues: design ...*

2012

2012-01 Terry Kakeeto (UvT) *Relationship Marketing for SMEs in Uganda*
2012-02 Muhammad Umair (VU) *Adaptivity, emotion, and Rationality in Human and Ambient Agent Models*
2012-03 Adam Vanya (VU) *Supporting Architecture Evolution by Mining Software Repositories*
2012-04 Jurriaan Souer (UU) *Development of Content Management System-based Web Applications*
2012-05 Marijn Plomp (UU) *Maturing Interorganisational Information Systems*
2012-06 Wolfgang Reinhardt (OU) *Awareness Support for Knowledge Workers in Research Networks*
2012-07 Rianne van Lambalgen (VU) *When the Going Gets Tough: Exploring Agent-based Models of Human ...*
2012-08 Gerben de Vries (UVA) *Kernel Methods for Vessel Trajectories*
2012-09 Ricardo Neisse (UT) *Trust and Privacy Management Support for Context-Aware Service Platforms*
2012-10 David Smits (TUE) *Towards a Generic Distributed Adaptive Hypermedia Environment*
2012-11 J.C.B. Rantham Prabhakara (TUE) *Process Mining in the Large: Preprocessing, Discovery, ...*
2012-12 Kees van der Sluijs (TUE) *Model Driven Design and Data Integration in Semantic Web Information ...*
2012-13 Suleman Shahid (UvT) *Fun and Face: Exploring non-verbal expressions of emotion during playful ...*
2012-14 Evgeny Knutov (TUE) *Generic Adaptation Framework for Unifying Adaptive Web-based Systems*
2012-15 Natalie van der Wal (VU) *Social Agents. Agent-Based Modelling of Integrated Internal and ...*
2012-16 Fiemke Both (VU) *Helping people by understanding them - Ambient Agents supporting task ...*
2012-17 Amal Elgammal (UvT) *Towards a Comprehensive Framework for Business Process Compliance*
2012-18 Eltjo Poort (VU) *Improving Solution Architecting Practices*
2012-19 Helen Schonenberg (TUE) *What's Next? Operational Support for Business Process Execution*
2012-20 Ali Bahramisharif (RUN) *Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer ...*
2012-21 Roberto Cornacchia (TUD) *Querying Sparse Matrices for Information Retrieval*
2012-22 Thijs Vis (UvT) *Intelligence, politie en veiligheidsdienst: verenigbare grootheden?*
2012-23 Christian Muehl (UT) *Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology ...*

afstanden aflegt en snelwegen de verbinding vormen tussen dorpen en steden zijn deze percentages misschien de beste, maar in Nederland met zijn ingewikkelde infrastructuur niet. Als de maker van het route-plan-algoritme de uitkomsten van de parameter tuning had geprobeerd te begrijpen, in plaats van enkel te publiceren, dan was zijn onderzoek ook buiten Amerika van nut geweest. Nu zal geen enkele Nederlandse bouwer zijn algoritme gaan gebruiken, ongeacht hoe goed het ook had kunnen werken als de juiste percentages zouden zijn gebruikt.

Hoewel dit een mooi voorbeeld is van het nut van hetgeen wij *scientific testing* noemen, namelijk het begrijpen en leren over algoritmen, zijn er nog vele andere voorbeelden mogelijk waarin andere aspecten een rol spelen. In het bovenstaande vraagstuk verandert enkel het probleem (van America naar Nederland), maar er zijn nog vele andere manieren waardoor de uitkomsten beinvloed kunnen worden. Is het bijvoorbeeld genoeg als een routeplan algoritme gemiddeld altijd een hele snelle weg vindt? Of moet hij altijd een gemiddeld goede weg vinden? Bij andere vraagstukken zullen aspecten meewegen zoals persoonlijke voorkeuren en het nemen van risico's. Het moge duidelijk zijn dat gewoonweg het algoritme publiceren met de bijbehorende parameters maar van geringe waarde is. Nu computers steeds snellers worden en tuners steeds beter, is er geen enkel excuus meer voor het gebruik van 'mooie' percentages of competitive testing.

In 1995 publiceerde John Hooker voor het eerst over scientific testing, maar zijn werk verdween in het archief door een gebrek aan computerkracht en werd vergeten. Nu 27 jaar later kan dat argument niet meer gebruikt worden en vormt zijn artikel van toen de inspiratie voor deze thesis. Met dit werk proberen we wetenschappers bewust te maken van het nut van competitive, maar vooral scientific testing. We laten verschillende voorbeelden zien waarmee we aantonen dat de huidige methode van gewoonweg mooie getallen gebruiken, onnodig, inefficient en van weinig waarde is. We geven aan op welke aspecten gelet moet worden bij zo'n wetenschappelijke aanpak en hoe de verschillende tuners, zoals het door ons geintroduceerde Bonesa, daarin van waarde kunnen zijn. Het zal een drastische verandering zijn, maar hij is nodig. Daarom eindigt deze thesis met dezelfde hoop en verwachting voor de toekomst die John Hooker 27 jaar geleden al uitsprak:

> "*in plaats van zichzelf te pijnigen over wat de best parameter waarden zijn, doet men wetenschappelijke experimenten om precies vast te kunnen stellen wat hun effect is op de uitkomst.*"
>
> – J.N. Hooker

is als hiervoor stond. Daar ging het om het aantal routes tussen A en B, hier om de percentages die gebruikt worden in het algoritme. Het spreekt voor zich dat we hier dus eenzelfde benadering kunnen gebruiken. Zo'n slim algoritme is dan in staat de parameters van een ander algoritme in te stellen, zonder dat algoritme zelfs te kennen. Ze gaan simpelweg op een slimme manier de beste mogelijke parameterwaarden af. Een dergelijk algoritme wordt in deze thesis een *automatische parameter tuning algoritme* of *tuner* genoemd. De fabrikant van het navigatiesysteem zou bijvoorbeeld zo'n tuner kunnen gebruiken om, voordat het systeem in de markt wordt gezet, de beste percentages te bepalen van het route-plan-algoritme. In deze thesis beschrijven we tientallen van dit soort automatische parameter tuning algoritmes, en vergelijken we ze aan de hand van hun verschillende kwaliteiten. Ook introduceren we twee nieuwe methoden: REVAC++ en Bonesa. Deze laatste heeft een aantal unieke kwaliteiten die geen van de andere tuners heeft. We laten dan ook het nut zien van deze kwaliteiten, en tonen aan dat Bonesa behoort tot de absolute top.

Scientific Testing

De bovenstaande manier van parameter tuning wordt in deze thesis *competitive testing* genoemd, omdat het enkel neerkomt op een competitie tussen de verschillende parameter-instellingen. Er wordt niets geleerd over het algoritme, aan het einde weten we nog steeds niet *waarom* iets een goede instelling is, enkel dat het een goede instelling is. Hoewel dit voor sommige toepassingen ook helemaal niet nodig is (een paar voorbeelden daarvan staan ook in deze thesis), is dat verre van een wetenschappelijke aanpak. Stel dat bij de publicatie van dit route-plan-algoritme, bijvoorbeeld door een Amerikaanse wetenschapper, wordt vermeld dat de beste percentages 5%, 94% en 1% zijn, hebben wij daar in Nederland dan wat aan? We willen nog steeds simpelweg van A naar B. Het doel verandert niet, dus er lijkt niets op tegen om deze percentages ook in Nederland te gebruiken. Maar, bij nadere inspectie valt direct de grote waarde op voor richting de snelweg gaan: 94%. Waarom zou dat zo hoog zijn? Wat is daarvan de reden?

Dit is precies het probleem van competitive testing, er is niets geleerd over de parameters. De tuner is blind aan het werk gezet, en de percentages kwamen er vanzelf uit. Hoewel dat al een hele verbetering is ten opzcihte van gewoon 'mooie' getallen gebruiken, is het nog steeds zeer beperkt. Als de maker van Nederlandse navigatiesystemen zomaar deze percentages had overgenomen, had hij namelijk een heleboel boze klanten gehad. In Amerika, waar men grote

zal het navigatiesysteem je dan aanbevelen. Hierdoor heb je dan weliswaar niet altijd de snelste route (want dat zou betekenen dat je precies elke keer goed hebt gekozen), maar in ieder geval heb je wel een binnen een paar seconden een redelijke goede route.

Een evolutionair algoritme werkt zelfs nog iets slimmer. In plaats van honderden keren vanaf het begin te beginnen, begint een evolutionair algoritme een aantal, bijvoorbeeld 50 keer vanaf het begin. Van deze 50 gaat hij eerst bekijken wat de snelste route was. Deze snelste route gebruikt hij als basis, om hopelijk een nog betere route te vinden. De eerste stap is om uit deze route willekeurig een kruispunt te kiezen. Vanaf dit kruispunt gaat de nieuwe route afwijken van de oude, want hij kiest speicifiek een andere weg dan hij eerst had gekozen. Op het moment dat hij dan weer een nieuw kruispunt bereikt, doet hij weer precies hetzelfde als voorheen: met 50% kans de ene kant, met 40% kans de andere kant, en met 10% kan een willekeurige. Als hij uiteindelijk B weer bereikt, is dat hopelijk een snellere route dan eerst. Is het inderdaad een verbetering, dan neemt hij deze nieuwe snelste route weer als basis voor een nieuwe poging. Op deze manier kun je net zo lang doorgaan totdat je er weer een paar honderd verschillende routes zijn geprobeerd. In de meeste gevallen zal dit een snellere route opleveren dan elke keer vanaf A te beginnen.

Parameter Tuning

Het bovenstaande voorbeeld van een navigatiesysteem is een mooi voorbeeld van de kracht van evolutionaire algoritmes, maar dit soort algoritmes hebben ook een bepaalde zwakte. Waarom hebben we bijvoorbeeld voor 50%, 40% en 10% gekozen? Zou 40%, 30%, 30% niet beter werken? Of 91.3%, 4.1%, 4.6%? Een goed antwoord op die vraag is er meestal niet. Over het algemeen worden zulke getallen gewoon gekozen omdat het mooie ronde getallen zijn, of omdat er 10 verschillende percentages zijn geprobeerd en dit nu eenmaal de beste was. Helaas hangt het succes van het algoritme wel heel erg af van deze *parameters*. Kies je te vaak een willekeurige weg, dan worden het nogal vreemde routes. Ga je te vaak richting de snelweg, dan rij je kilometers om als je enkel naar het winkelcentrum wil. En ga je te vaak direct richting je doel, dan kom je terecht op de langzame binnenweggetjes, in plaats van de snelweg. Om te weten wat de beste percentages zijn, moet je alle mogelijke percentages proberen en kijken wat de beste is. Het probleem is daarbij dat er oneindig veel verschillende mogelijkheden zijn, dus het is onmogelijk om ze allemaal af te gaan.

Een oplettende lezer zou nu hebben opgemerkt dat dit exact dezelfde zin

Parameter Tuning and Scientific Testing in Evolutionary Algorithms

Selmar Kagiso Smit

Vrije Universiteit, Amsterdam

Binnen algoritmes spelen parameter een belangrijke, maar vaak onderschatte rol. De instellingen van de parameter bepalen namelijk in grote mate de werking en daarmee de uitkomst van het algoritme. Je zou zelfs kunnen stellen dat ze bijna belangrijker zijn dan het model zelf, immers zonder goede instellingen werkt geen elke algoritme goed. Toch wordt dit tegenwoordig nog geregeld over het hoofd gezien en richt men zich vooral op het ontwikkelen van nieuwe algoritmes, in plaats van betere parameters. In dit proefschrift staat dan ook het *hoe* en *waarom* van het instellen van parameters centraal, specifiek binnen het vakgebied evolutionary algorithms. Evolutionary algorithms, parameter tuning en scientific testing blijken onlosmakelijk met elkaar verbonden.

Evolutionary Algorithms

Een evolutionair algoritme is een speciale manier voor het oplossen van problemen, dat door middel van willekeurige acties een oplossing probeert te vinden. Denk bijvoorbeeld aan een navigatiesysteem dat de snelste route moet vinden van A naar B. Hoewel dat simpel lijkt, aangezien je kunt berekenen hoe lang een bepaalde route duurt, is het behoorlijk inwikkeld om op te lossen. Het probleem is daarbij dat er oneindig veel verschillende mogelijkheden zijn, dus het is onmogelijk om ze allemaal af te gaan. In plaats daarvan kan je beter een manier verzinnen waardoor je altijd binnen een paar seconden een hele snelle (niet noodzakelijk de snelste) route kunt vinden. Een mogelijke manier is bijvoorbeeld om route vanaf het begin op te bouwen, door eerst simpelweg bij A te beginnen. Bij de eerste kruising die je tegenkomt kun je vervolgens op een slimme manier bepalen welke van de vier wegen je kiest. Je zou bijvoorbeeld in 50% van de gevallen kunnen kiezen voor de weg die de goede richting op gaat, in 40% van de gevallen te kiezen voor de weg die richting een snelweg gaat, en in 10% van de gevallen een willekeurige weg te kiezen. Bij de eerstvolgende kruising kan je weer precies hetzelfde doen, en hetzelfde voor alle andere kruisingen die je tegenkomt, net zolang totdat je B hebt bereikt. Een computer kan een dergelijke berekening binnen een fractie van een seconde doen, dus er is tijd genoeg om het geheel een paar honderd keer te doen. De beste van die honderden gevonden routes

In this thesis, the issue of evaluating and comparing evolutionary algorithms is adressed, and more specific, the role of parameter tuning in it. A more scientific testing approach is promoted in which the focus is on *why* and *when* an algorithm performs better, rather than *if* it is better. Such an approach is not only more informative for the users of algorithms, but also changes the work of researchers from development into science.