Comparing Parameter Tuning Methods for Evolutionary Algorithms

S.K. Smit A.E. Eiben

Abstract— **Tuning the parameters of an evolutionary algorithm (EA) to a given problem at hand is essential for good algorithm performance. Optimizing parameter values is, however, a non-trivial problem, beyond the limits of human problem solving.In this light it is odd that no parameter tuning algorithms are used widely in evolutionary computing. This paper is meant to be stepping stone towards a better practice by discussing the most important issues related to tuning EA parameters, describing a number of existing tuning methods, and presenting a modest experimental comparison among them. The paper is concluded by suggestions for future research – hopefully inspiring fellow researchers for further work.**

Index Terms— **evolutionary algorithms, parameter tuning**

I. Background and objectives

Evolutionary Algorithms (EA) form a rich class of stochastic search methods that share the basic principles of incrementally improving the quality of a set of candidate solutions by means of variation and selection [7], [5]. Algorithms in this class are all based on the same generic framework whose details need to be specified to obtain a particular EA. It is customary to call these details *EA parameters*, and designing an EA for a given application amounts to selecting good values for these parameters.

Setting EA parameters is commonly divided into two cases, parameter tuning and parameter control [6]. In case of parameter control the parameter values are changing during an EA run. In this case one needs initial parameter values and suitable control strategies, that in turn can be deterministic, adaptive, or self-adaptive. Parameter tuning is easier in that the parameter values are not changing during a run, hence only a single value per parameter is required. Nevertheless, even the problem of tuning an EA for a given application is hard because there is a large number of options, but only little knowledge about the effect of EA parameters on EA performance. EA users mostly rely on conventions (mutation rate should be low), ad hoc choices (why not use uniform crossover), and experimental comparisons on a limited scale (testing combinations of three different crossover rates and three different mutation rates).

The main objective of this paper is to illustrate the feasibility of using tuning algorithms, thereby motivating their usage. To this end, we describe three different approaches to algorithmic parameter tuning (meta-EA, meta-EDA, SPO) and show their (dis)advantages when tuning EA parameters for solving the Rastrigin function. While the limited scale (one single fitness landscape and one algorithm to be tuned) prevents general conclusions, we do obtain a convincing showcase and some very interesting insights whose generalization requires much more experimental research.

Vrije Universiteit Amsterdam, The Netherlands, {sksmit, gusz}@cs.vu.nl

II. Parameters, Tuners, and Utility Landscapes

Intuitively, there is a difference between choosing a good crossover operator and choosing a good value for the related crossover rate p_c. This difference can be formalized if we distinguish parameters by their domains. The parameter `crossoveroperator` has a finite domain with no sensible distance metric, e.g., $\{\texttt{onepoint}, \texttt{uniform}, \texttt{averaging}\}$, whereas the domain of the parameter p_c is a subset of $\mathbb{R}$ with the natural metric for real numbers.

This difference is essential for searchability. For parameters with a domain that has a distance metric, 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 enumeration.

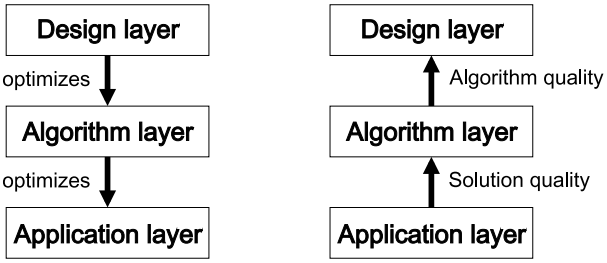
For a clear distinction between these cases we can use the terms *symbolic parameter*, e.g., `crossoveroperator`, and *numeric parameter*, e.g., crossover rate. For both types of parameters the elements of the parameter's domain are called *parameter values* and we *instantiate* a parameter by allocating a value to it.

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

For positioning algorithms for parameter tuning it is helpful to distinguish three layers: application layer, algorithm layer, and design layer, see Figure 1.

The lower part of this three-tier hierarchy consists of an EA on the algorithm layer trying to find an optimal solution for the problem on the application layer, e.g., the traveling salesman problem. Simply put, the EA is iteratively generating (candidate) solutions, e.g., permutations of city names, whose quality is determined by the given problem on the application layer.

The upper part of the hierarchy contains a design method that is trying to find optimal parameters for the EA on the



(a) Control flow (b) Information flow

Fig. 1. The 3-layered hierarchy of parameter tuning

algorithm layer. The design method can be, for instance, a heuristic procedure (algorithm) or an interactive session with the user itself. We can formalize the problem to be solved here by denoting the symbolic 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 numeric parameters.[1] The problem of parameter tuning can then be seen as a search problem in the parameter space.

$$S = Q_1 \times Q_2 \cdots \times Q_m \times R_1 \times R_2 \cdots \times R_n \qquad (1)$$

Solutions of the parameter tuning problem can then be defined as parameter vectors with maximum utility, where the *utility* of a given parameter vector $\bar{p} \in S$ is the performance of the EA using the values of $\bar{p}$. Using this nomenclature we can define the *parameter-performance landscape*, or *utility landscape*, as an abstract landscape where the locations are the parameter vectors in S and the height of a $\bar{p} \in S$ is its utility. Intuitively it is quite obvious that fitness landscapes –commonly used in EC, i.e., within the context of the lower part of the hierarchy– have a lot in common with utility landscapes –as introduced here for the upper part. 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 "height", and a method to assess the quality of a point in the search space (evaluation vs. testing). Finally, we have a search method (an evolutionary algorithm vs. a tuning procedure) that is seeking for a point with maximum height. Table I provides a quick overview of the related vocabulary.

TABLE I
ONE-GLANCE OVERVIEW OF PARAMETER TUNING TERMINOLOGY

	Lower part	**Upper part**
Method at work	EA	tuning procedure
Search space	solution vectors	parameter vectors
Quality	fitness	utility
Assessment	evaluation	testing

Despite the obvious analogies between the upper and the lower halfs, there are two differences we want to note here. First, 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, is based on the performance of the EA that can be defined in essentially different ways, for instance, based on solution quality or algorithm speed. Furthermore, performance can be average or peak performance over a number a EA runs. Consequently, the definition of a good solution is more sensitive for user preferences on the upper half (in the context of parameter tuning) than on the lower half (in the context of an EA application). Second, the performance of the EA depends on the problem the EA is solving, that is, the definition of utility depends on the definition of fitness.

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

III. Algorithmic Approaches to Parameter Tuning

As mentioned in Section II, an EA has symbolic and numeric parameters. In general, the space of symbolic parameters does not have a searchable structure and can only be treated by enumeration or grid search methods. Therefore we focus on the numeric parameters here and describe three different approaches to optimizing them.

Finding a good set of parameter values is a complex optimization task with a nonlinear objective function, interacting variables, multiple local optima, noise (by the stochastic nature of the EA to be tuned), and a lack of analytic solvers. Ironically, it is exactly this type of problems where EAs are very competitive heuristic solvers. It is therefore a natural idea to use an evolutionary approach to optimize the parameters of an evolutionary algorithm. Two of the three methods we describe in the following are based on this idea.

A. Meta Evolutionary Algorithm

Mercer and Sampson [13] were the first to introduce a meta-EA, but due to the large computational costs, their research was very limited. Greffenstette [8] did conduct more extensive experiments with his Meta-GA and showed its effectiveness.

The individuals used in such a meta-EA (on the design layer) are vectors of numerical values. Each of those values belong to one of the parameters of the baseline EA to be tuned. To evaluate the utility of such a vector, the baseline EA is ran several times using the given parameter values. Using this representation and utility as (meta) fitness, basically any evolutionary algorithm can be used as the meta-EA, if only it can cope with real-valued vectors as individuals.

In this paper we use an Evolution Strategy (ES) with Covariance Matrix Adaptation (CMA) as proposed by Hansen [9] as a meta-EA. This choice is motivated by the good reputation of Evolutionary Strategies as numerical optimizers. The CMA-ES is currently the state-of-the-art improvement of the standard ES.

B. Meta Estimation of Distribution Algorithm

Nannen and Eiben have introduced a method for Relevance Estimation and Value Calibration of parameters (REVAC) in [16], [15]. Although the REVAC method was not designed

with Estimation of Distribution Algorithms (EDA) in mind, it is based on the same general idea [10]. Like all EDAs, REVAC tries to find an optimal parameter vector by estimating the distribution of promising values over the domain of each parameter and creating specific vectors by drawing values from these distributions. REVAC has a characteristic way of updating the distributions after having evaluated newly drawn vectors. In essence, REVAC is a population-based stochastic search method, where the population consists of parameter vectors of the baseline EA and one individual (i.e., one vector) is replaced in each cycle. After termination of the algorithm, the estimated distributions per parameter represent a model of the utility landscape. This model is rather simple (separated by coordinates, resp. parameters, hence blind for parameter interactions), but it can be used to get insights into the sensitivity and relevance of the different parameters and the costs and benefits of tuning each parameter [17].

In this paper we use REVAC with the settings from earlier publications, not adjusted, let alone optimized, for the present case study. In this respect, it is different from the other two methods, where we use variants that have been much studied and improved since their 'birth'.

C. Sequential Parameter Optimization

Sequential Parameter Optimization (SPO), as introduced by Bartz-Beielstein et al. [4], [1], is a search-method specifically designed for parameter tuning and parameter analysis. The approach shows some similarities is similar to a meta-EDA in that it relies on a model of the utility landscape.

SPO starts with a initial population of vectors. Those are tested several times to determine their utility. Based on the results, a (regression) model is fitted to represent the relation between the vectors and the results. Then s new vectors are generated and tested using this model. The most promising points are then added to the population. Although in [3], [2] regression models are used to model the utilities, succeded by stochastic models [4], it is in principle a general framework suited for a large range of modeling techniques.

In this paper we have chosen to use Kriging models for approximating the utility landscape, because of their excellent performance on tuning problems with numerical parameters [4].

IV. Add-ons for Parameter Tuning Algorithms

A careful study of related work discloses that besides the principal parameter tuning algorithms, like meta-EA, REVAC or SPO, there are a number of useful 'add-ons', i.e., methods for increasing search efficiency, that are independent from the main tuner and can be combined with different tuning algorithms. It this section we highlight two of such promising add-ons.

A. Racing

Racing was introduced by Maron and Moore [12]. 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 algorithm. 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 iteratively increase the number of tests for those vectors only that are not significantly worse or better than the good ones. This method can save a substantial number of tests compared to the simple each-vector-n-tests approach.

Yuan et al. [19] used this feature in their (1 + λ) ES for tuning an evolutionary algorithm. In their approach, at each generation, a set of λ new vectors is created using a Gaussian distribution centered at the current best vector. Racing is then used to determine which vector has the highest utility. This approach can be easily extended to a (μ + λ) ES, by using racing to determine the μ best individuals instead of the single best.

B. Sharpening

Sharpening has not been introduced before as a separate technique for testing, although it has been used previously, inside the SPOmethod by Bartz-Beielstein et al. [2]. Thus, in this paper we do not invent it, but designate it as an independent add-on, and give it the name sharpening. The purpose of sharpening is to decrease the number of tests needed to estimate the quality of parameter vectors as compared to the simple each-vector-n-tests approach. Like racing, it is to reduce the total runtime of a tuner algorithm.

The main idea is to start the tuning algorithm with a small number of tests per vector, but when a certain threshold is reached the amount of tests per vector is doubled. This means that the algorithm is 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.

C. Combining Racing and Sharpening

Observe that racing and sharpening are opposing forces. Sharpening is increasing the number of tests, while racing is reducing them. Nevertheless, they can be combined very easily. In a combined setup sharpening will increase the maximum number of tests 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 'budget', but 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.

V. System Description and Experimental Setup

As described in Section II, the complete system consists of three different layers with a control and information flow. On the application level we have chosen to use a 20 dimensional Rastrigin function. The Rastrigin function [18] is a popular non-linear, highly multimodal, scalable benchmark function with an optimal value[2] of 0. In this system, we have chosen for the implementation from the ECJ [11] library which is open source and freely available.

For the Algorithm layer we again choose for an implementation from the ECJ library. This library is widely used by EC practitioners and serves as a framework for a whole range of evolutionary algorithms. It is written in Java and allows users to configure Evolutionary Algorithms using Java code or parameter-files. We have chosen for a 'standard' configuration of the middle-layer algorithm (Table II).

This setup requires 6 parameters to be defined by the design algorithm from the top layer.

For the top layer, we have tested three different algorithms combined with the two additional add-ons namely:

- CMA-ES
- CMA-ES with Racing
- CMA-ES with Sharpening
- CMA-ES with Racing and Sharpening
- SPO (uses Sharpening)
- REVAC
- REVAC with Racing
- REVAC with Sharpening
- REVAC with Racing and Sharpening

The three base algorithms, CMA-ES [9], SPO [2] and Racing [14], are open source and freely available from the websites of the corresponding authors. The additional add-ons are handcrafted changes to the three algorithms and their exact implementations are algorithm specific. All of the tuning algorithms are ran with their default parameter values and setup (Table III, IV and V).

A. Control Flow

The control flow between the application and the algorithm is trivial and only requires the definition of two parameters. Both layers are defined in the ECJ library and fit into the same framework. However, implementing the the control flow between the algorithm and the design level is slightly more difficult. The CMA-ES is implemented in Java, so creating the link between both layers is quite straightforward. By implementing a Java interface that sets the algorithm parameters and executes the ECJ algorithm, both layers are connected. REVAC and SPO are both implemented in Matlab, which is also able to communicate directly with Java libraries. For REVAC the user needs to write a Matlab function that communicates with a Java class. This Java class can be the same as used with the CMA-ES implementation. SPO can be linked to any function or executable as long as it can read and write SPO property files. This requires a bit more work, but it is more flexible.

[2]Because we have a preference for maximization problems, a negative transformation is applied

B. Information Flow

The information flow between the layers uses the same interfaces or property files as used in the control flow. The information that is passed to the algorithm layer by the application layer is a single value representing the fitness of the current vector. The algorithm layer, however, does not send a single value to the design layer, but a list of utilities. The design layer algorithm defines how many tests have to be executed in order to evaluate a single parameter vector. If the design algorithm requests for s tests of a certain parameter vector, the corresponding information that is send contains s values, which represent the best fitness values from each of the s test runs. In this setup the tuning is focused on improving the fitness.

The same setup can be used to tune for speed. But in that case, the algorithm layer have to send the s number of evaluations used to solution as information for the design layer.

C. Measures

Each of the tuning-algorithms is allowed to perform a total of 1000 tests and is repeated 10 times. To measure the quality of each tuning-algorithm the following criteria are used:

- Average Performance
- Maximum Performance
- Variation of Performance
- Effectiveness of the Tuning Algorithm

The average performance is measured by the Mean Best Utility over the 10 runs. To estimate the utility, the best vector that is found in each run, is tested 50 times. The utility is equal to the average fitness values over the 50 runs. Maximum performance is measured similarly, but instead of the average utility over the 10 runs, the maximum utility is used.

To measure the variation in performance, the difference in top and lower quantiles of the utility distribution of the 10 runs is used. Finally, to measure the effectiveness of the tuning algorithm we identify the 'target' area, namely the area of the utility landscape with the highest performance. For each parameter the minimum and the maximum value is calculated over the 10 best vectors. The effectiveness of the algorithms is defined by its ability to reach the area enclosed by these values.

VI. Results

Tuning an algorithm requires a lot of computer power, while some people argue that this is a waste of time. General thumb rules as a population size of 100 and low mutation sizes are supposed to perform reasonably well. The question rises how beneficial tuning, and more specific automated tuning, is even to experienced practitioners.

For quick assessment of the added value of algorithmic tuning we tested an EA using parameter values defined by 'common sense' (Table VI).

TABLE II
SETUP OF THE ALGORITHM LAYER

	Value	**Parameters**
Population	Single population	Population size (POPSIZE)
Parent Selection	Tournament selection	Tournament proportion (TOURPROP)
Elitism	Yes	Generation gap (GENGAP)
Crossover	Uniform	Crossover probability (CROSSPROB)
Mutation	Gaussian	Mutation probability (MUTPROB) and σ (MUTSIZE)
Termination	20.000 fitness evaluations	

TABLE III
PARAMETER VALUES CMA-ES

Parameter	**Value**
μ	4
λ	9
Racing	
Minimum # evaluations	2
Maximum # evaluations	5
Sharpening	
Factor	1.5
Threshold	18

TABLE IV
PARAMETER VALUES SPO

Parameter	**Value**
# Initial design points	60
# Samples per point	1
# Candidates	10

TABLE V
PARAMETER VALUES REVAC

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

The average performance of this manually chosen parameter vector is -44.00, while the EAs using parameter vectors optimized by the tuners easily reach utility levels around -0.05.

Table VII shows the minimum and maximum performance of the 10 runs. It also indicates the median performance, and the four quantiles. Because algorithm tuning is a kind of design problem [7] the maximum(peak) performance is probably the most interesting value. Tuning an algorithm is not a repetitive task, in the sense that it is not required to deliver a good value each time it is ran. The average performance is therefore less important than the maximum performance that can be reached.

The best parameter settings are found by the CMA-ES, racing and sharpening combination, followed by SPO. REVAC shows on average a significantly worse performance, however when combined with both racing and sharpening, the best performance gets close to the performance of the other algorithms. The main cause of the bad performance of REVAC is the speed. From more detailed results, not shown here, we observe a steady increase in performance during the run. However, the 1000 allowed tests, forces REVAC to terminate prematurely.

It is clear that the combined effect of racing and sharpening increases the maximum performance in this setup, however, the effect on the CMA-ES is quite different than the effect on REVAC. When combining racing or sharpening with CMA-ES, the variance increases and sharpening alone even decreases the overall performance. However, when combined with REVAC, all three combinations decrease variance and increase the best and the overall performance. Adding those components to the algorithm is therefore probably beneficial, although it is difficult to predict the effect on performance.

Our experimental data can not only be used to learn about tuning algorithms, but it could also tell about high quality settings of the baseline EA. For this purpose we investigate the 10 best parameter vectors found in *all* runs. Table VIII shows these 10 parameter vectors, together with the tuner that found them and the corresponding utility, being the mean best fitness of the EA using the given parameter vector, averaged over 50 independent runs.

Interestingly, there are rather big differences between the top 10 parameter vectors, depending on the specific parameters. For instance, the optimized population size varies between 11 and 448. Also for the generation gap we find optimized values far from each other, e.g., 4% (in combination with population size 23) and 84% (in combination with population size 14). To obtain more information about the spreading of optimized parameter values we performed experiments with the best variant of each of the three basic methods: CMA-ES with racing and sharpening, REVAC with racing and sharpening, and SPO. We executed 10 independent runs with each of them, resulting in 3 times 10 optimized parameter vectors. The outcomes are shown in Figure 2, split by parameter, the dots showing the actual parameter values. These results show that even if we use the same tuning algorithm, we can get very different optimized values, although this picture varies per parameter and by tuning algorithm. For instance, the CMA-ES and SPO are consistent in their values for mutation probability, but this does not hold for REVAC. As for the population size, all three algorithms show a wide range of good values. For

TABLE VI

MANUAL CHOICE OF PARAMETER VALUES

	Value
Population size	100
Tournament proportion	3%
Generation gap	2%
Crossover probability	0.8
Mutation probability	1
σ	0.1

TABLE VII

THE MINIMUM, MAXIMUM, MEDIAN AND QUANTILES OF THE ALGORITHM UTILITIES

Algorithm	**Max**	Q_1	$Q_{.75}$	**Median**	$Q_{.5}$	$Q_{.25}$	**Min**
CMA-ES	-0.0508	-0.0508	-0.0542	-0.0589	-0.0783	-0.0844	-0.1761
CMA-ES (Racing)	-0.0545	-0.0545	-0.0608	-0.0667	-0.0747	-0.0847	-0.0847
CMA-ES (Sharpening)	-0.0508	-0.0508	-0.0542	-0.0713	-0.0910	-0.1085	-0.2317
CMA-ES (Racing , Sharpening)	-0.0388	-0.0388	-0.0490	-0.0751	-0.0966	-0.1381	-0.1381
SPO	-0.0457	-0.0457	-0.0482	-0.0623	-0.0776	-0.0776	-0.0776
REVAC	-0.1031	-0.1031	-0.1588	-0.3502	-83.6759	-40.9933	-83.6759
REVAC (Racing)	-0.0678	-0.0678	-0.2934	-0.9423	-2.3175	-2.3175	-67.7445
REVAC (Sharpening)	-0.0822	-0.0822	-0.0936	-0.2762	-5.1779	-5.1779	-357.4347
REVAC (Racing , Sharpening)	-0.0573	-0.0573	-0.0784	-0.1642	-19.7132	-19.7132	-102.5977

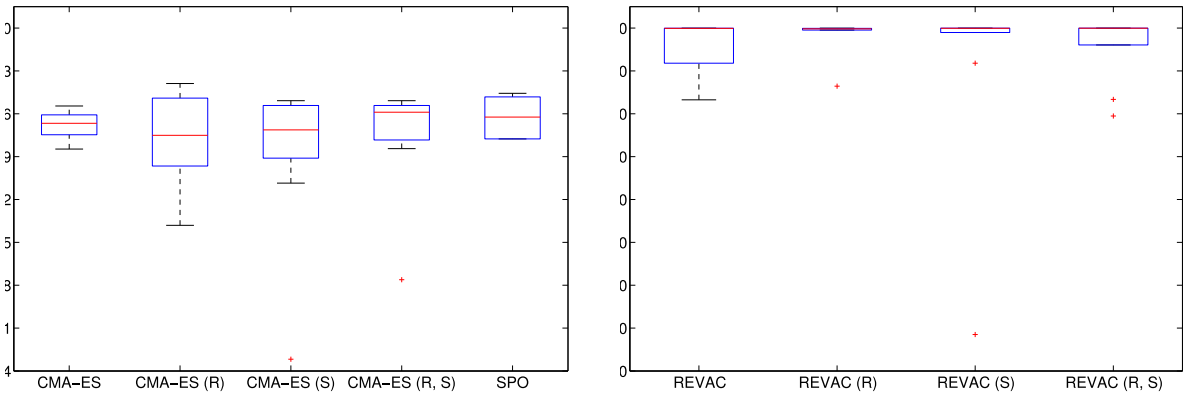


Illustration of the quantiles. Mind the different scales

comparison with the results concerning the overall top 10 vectors, we augmented Figure 2 with blocks exhibiting the 'target areas', where the upper/lower borders of the block show the maximum/minimum values from Table VIII, and the middle line belongs to the mean. The overall picture that arises is that, except the mutation parameters, it is hard for tuners to consistently reach the areas with the best EA setup. This shows that it is needed to run each tuning-algorithm several times in order to find a good parameter setup.

VII. CONCLUSIONS AND OUTLOOK

The main objective of this paper is to illustrate the feasibility of using algorithms for tuning parameters of EAs. To this end, we performed experiments with ten tuning algorithms, based on three different approaches, meta-EA, REVAC, and SPO. As mentioned before, due to computational and time limitations these tests have a limited scale (one single fitness landscape and one EA to be tuned). While this prevents general conclusions, we did obtain a convincing showcase and some very interesting insights that motivate further research and development.

Perhaps the most important conclusion is that using algorithms for tuning parameters of EAs does pay off in terms of EA performance. To be specific, the best guess (i.e., the parameter vector with the highest utility) of all of the algorithms we tested greatly outperforms the best guess of a human user. Simply put, no matter what tuner algorithm you use, you will likely get a much better EA than relying on your intuition and the usual parameter setting conventions.

Further to the EA performance benefits, tuner algorithms are also useful for they go for the best parameter vectors without being hindered by those rules-of-thumb human users rely on. This can lead to surprising parameter settings,

TABLE VIII
THE 10 BEST PERFORMING PARAMETER VECTORS FOUND

	Algorithm	Utility	POPSIZE	MUTPROB	MUTSIZE	CROSSPROB	TOURPROP	GENGAP
1	CMA-ES (R, S)	-0.0388	37	0.0510	0.5641	0.5225	0.8702	0.5535
2	CMA-ES (R, S)	-0.0457	14	0.0502	0.5684	0.4537	0.8782	0.8443
3	SPO	-0.0457	448	0.0540	0.6219	0.5000	0.3503	0.0215
4	SPO	-0.0472	23	0.0686	0.6130	0.7789	0.8580	0.0408
5	SPO	-0.0482	33	0.0472	0.6370	0.7536	0.6725	0.2173
6	CMA-ES (R, S)	-0.0490	122	0.0631	0.5897	0.6525	0.5108	0.1582
7	CMA-ES (S)	-0.0508	271	0.0557	0.5955	0.3886	0.9207	0.1078
8	CMA-ES	-0.0508	271	0.0557	0.5955	0.3886	0.9207	0.1078
9	CMA-ES (S)	-0.0514	11	0.0416	0.6358	0.5791	0.4946	0.6300
10	CMA-ES	-0.0514	11	0.0416	0.6358	0.5791	0.4946	0.6300
	Percentage of Total Range		54.63 %	2.70 %	7.29 %	39.02 %	57.04 %	82.28 %

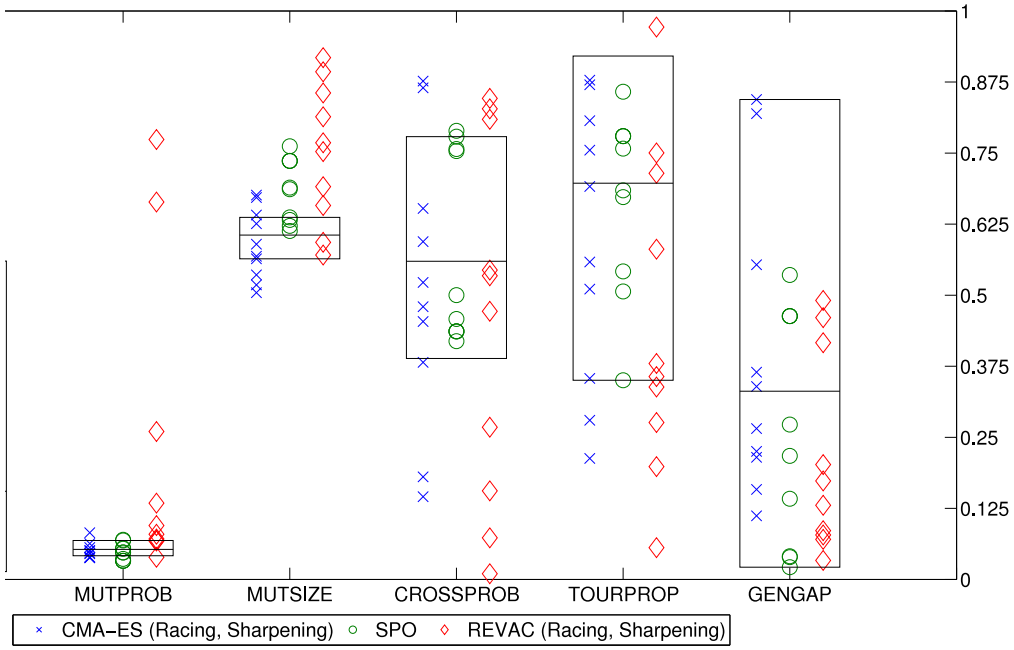


Fig. 2. The parameter values found by the best variants of our three methods (dots) and the 'target' area (blocks). See text for explanation.

thereby offering a critical look on such rules-of-thumb. For instance, in most EA publications the tournament size is typically in the range of 2 to 10. The optimal values, however, seem to higher, in the range of tens (while population sizes are rather conventional). Strictly speaking, this only holds for the Rastrigin function and the EA we investigated here, but we do believe that the conventional wisdom is wrong in many more cases and tuning algorithms can help to show this.

Our results also support preferences regarding the tuning algorithms to be used. For a careful advise, we need to distinguish two functionalities tuners can offer. First and foremost, they can optimize EA parameters, second they can provide insights into the (combined) effects of parameters on EA performance. Regarding the insights offered the three methods we tested are quite different. On the low end of this scale we have the CMA-ES that is a highly specialized optimizer building no model of the utility landscape. REVAC, and meta-EDAs in general, does create a model, the marginal distributions over the ranges of each parameter. The fact that these distributions only take one parameter into account means that the model is simple, it is blind to parameter interactions. On the other hand, REVAC is able to provide information about the entropy associated with the parameters, hence showing the amount of tuning each parameter requires. SPO is situated on the high end of the insights scale, since it is inherently based on a model of the utility landscape. In principle, this model is not restricted to a specific form or structure, offering the most flexibility and insights, including information on parameter interactions. Based on these considerations and the outcomes of our experiments our preferred method is the CMA-ES if a very good parameter

vector is the most important objective, and SPO if one is also interested in detailed information over the EA parameters.

Regarding future work we see a number of promising directions. The most straightforward track is to extend the scope of the present study and perform much more experiments using more objective functions and different EAs. This is needed to refine and consolidate our present findings and will most likely disclose new facts. From the practical point of view, the development of a toolbox is the most urgent task. Such a toolbox should contain one or more parameter tuning algorithms allowing their combinations with racing and/or sharpening. Furthermore, such a toolbox should be easy to use. That is, it should enable EC practitioners with limited time and computer experience to plug in their EA and the problem to be solved and produce good parameter settings.

REFERENCES

[1] T. Bartz-Beielstein, C.W.G. Lasarczyk, and M. Preuss. Sequential parameter optimization. In *IEEE Congress on Evolutionary Computation*, volume 1, pages 773–780 Vol.1. IEEE, Sept. 2005.

[2] 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, 2004.

[3] 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.

[4] 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.

[5] K.A. De Jong. *Evolutionary Computation: A Unified Approach*. The MIT Press, 2006.

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

[7] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computation*. Natural Computing Series. Springer, 2003.

[8] J.J Greffenstette. Optimisation of Control Parameters for Genetic Algorithms. In *IEEE Transactions on Systems, Man and Cybernetics*, volume 16, pages 122–128, 1986.

[9] N. Hansen. The CMA evolution strategy: a comparing review. In J.A. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea, editors, *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, pages 75–102. Springer, 2006.

[10] Pedro Larraanaga and Jose A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[11] S. Luke et al. A java-based evolutionary computation research system. http://www.cs.gmu.edu/∼eclab/projects/ecj/.

[12] O. Maron and A. Moore. The racing algorithm: Model selection for lazy learners. In *Artificial Intelligence Review*, volume 11, pages 193–225, April 1997.

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

[14] 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.

[15] V. Nannen and A. E. Eiben. Relevance Estimation and Value Calibration of Evolutionary Algorithm Parameters. In Manuela M. Veloso, editor, *IJCAI*, pages 1034–1039, 2007.

[16] V. Nannen and A.E. Eiben. A Method for Parameter Calibration and Relevance Estimation in Evolutionary Algorithms. In M. Keijzer, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2006)*, pages 183–190. Morgan Kaufmann, San Francisco, 2006.

[17] V. Nannen, S.K. Smit, and A.E. Eiben. Costs and benefits of tuning parameters of evolutionary algorithms. In G. Rudolph, Th. Jansen, S.M. Lucas, C. Poloni, and N. Beume, editors, *Parallel Problem Solving from Nature – PPSN X*, volume 5199 of *Lecture Notes in Computer Science*, pages 528–538. Springer Berlin / Heidelberg, 2008.

[18] Aimo Torn and Antanas Zilinskas. *Global optimization*. Springer-Verlag New York, Inc., New York, NY, USA, 1989.

[19] B. Yuan and M. Gallagher. Combining Meta-EAs and Racing for Difficult EA Parameter Tuning Tasks. In F.G. Lobo, C.F. Lima, and Z. Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, pages 121–142. Springer, 2007.