Evolution Strategies and Threshold Selection

Thomas Bartz-Beielstein

Department of Computer Science
University of Dortmund, Germany
thomas.bartz-beielstein@udo.edu,
WWW home page: http://ls11-www.cs.uni-dortmund.de/people/tom

Abstract. A hybrid approach that combines the $(1+1)$-ES and threshold selection methods is developed. The framework of the new experimentalism is used to perform a detailed statistical analysis of the effects that are caused by this hybridization. Experimental results on the sphere function indicate that hybridization worsens the performance of the evolution strategy, because evolution strategies are well-scaled hill-climbers: the additional threshold disturbs the self-adaptation process of the evolution strategy. Theory predicts that the hybrid approach might be advantageous in the presence of noise. This effect could be observed—however, a proper fine tuning of the algorithm's parameters appears to be advantageous.

1 Introduction

Following Stützle and Hoos, metaheuristic approaches can be described as generic techniques that are used "to guide or control an underlying problem-specific heuristic method in order to improve its performance or robustness"[1]. Hybrid metaheuristics combine methods of different metaheuristics. Two contradictory trends can be observed in recent research: (i) to develop more and more new algorithms or (ii) to analyze and understand existing heuristics and to add new features only when necessary. Following (ii), we will analyze potential assets and drawbacks that arise from a combination (hybridization) of evolution strategies and threshold selection. The analysis comprehends methods from the *new experimentalism*, that is an influential trend in recent philosophy of science. The new experimentalists develop statistical methods to set up experiments, to test algorithms, and to learn from the resulting errors and successes [2].

In many cases heuristics require the determination of parameters before the optimization run is performed. In the remainder of this paper, optimization runs will be treated as experiments. From the viewpoint of an experimenter, design variables (factors) are the parameters that can be changed during an experiment. Here comes the new experimentalism into play: a systematic variation of these factors and a statistical analysis of the resulting errors and successes are the keys for an understanding of the algorithm's performance. Generally, there are two different types of factors that influence the behavior of an optimization algorithm: (i) *problem specific* and (ii) *algorithm specific factors*.

These factors will be discussed in Section 2. Evolution strategies will be introduced in Section 3, and threshold selection approaches are presented in Section 4. Section 5 considers test problems and performance measures that are used afterwards to perform the experiments. The paper closes with a summary and conclusion.

2 Experimental Designs

Algorithm specific factors will be considered first: *Endogenous* can be distinguished from *exogenous* algorithm parameters. The former are kept constant during the optimization run, whereas the latter, e.g. standard deviations in evolution strategies, are modified by the algorithms during the run. An *algorithm design* $\mathcal{D}_A$ is a set of vectors with specific settings of an algorithm. A design can be specified by defining ranges of values for the design variables, e.g. "1:1:10" denotes integers from 1 to 10, whereas "1:10" denotes real numbers from the interval $[1, 10]$, or by specifying a set of values, e.g. "$\{1, 5, 10\}$". Note that a design can contain none, one, several or even infinitely many vectors. We will consider quantitative factors only. How qualitative factors can be included into the experimental analysis is discussed in [3].

Problem designs $\mathcal{D}_P$ provide information related to the optimization problem, such as the available resources, e.g. the number of function evaluations $t_{\max}$. Furthermore it is important to specify initialization and termination criteria. An *experimental design* $\mathcal{D}$ consists of a problem design $\mathcal{D}_P$ and an algorithm design $\mathcal{D}_A$. The run of a stochastic search algorithm can be treated as an experiment with a stochastic output $Y(x_a, x_p)$, with $x_a \in \mathcal{D}_A$ and $x_p \in \mathcal{D}_P$. If random seeds are specified, the output would be deterministic. This case will not be considered further, because it is not a common practice to specify the seed that is used in an optimization run.

Performance can be measured in many ways, for example as the best or the average function value from n runs (see also Section 5). One of our goals is to find a design point $x_a^* \in \mathcal{D}_A$ that improves the performance of an optimization algorithm for one problem design point $x_p \in \mathcal{D}_P$. To test the robustness of an algorithm, more than one design point can be considered. The approach to determine good design points presented in this paper is based on the *sequential parameter optimization* (SPO) methodology developed in [4] that has been applied successfully in several contexts, e.g. [5, 6].

3 The Two Membered Evolution Strategy

The two membered evolution strategy, or $(1+1)$-ES, is included in our analysis for three reasons: (i) It is easy to implement, (ii) it requires only a few exogenous parameters, and (iii) it defines a standard for comparisons. Many optimization practitioners apply the $(1 + 1)$-ES (Figure 1) to their optimization problem. Schwefel [7] describes this algorithm as "the minimal concept for an imitation of organic evolution". The standard deviation σ will be referred to as *step-width*

Procedure $(1+1)$-ES/TS.

Initialization: Initialize the iteration counter: $t = 1$. Determine: (i) a point $X_1^{(t)}$ with associated position vector $x_1^{(t)} \in \mathbb{R}^d$, (ii) a standard deviation $\sigma^{(t)}$, and (iii) a threshold value $\tau^{(t)}$. Determine the function value $y_1 = f(x_1^{(t)})$.

`while` some stopping criterion is not fulfilled `do`

`repeat` M times:

Mutation: Generate a new point $X_2^{(t)}$ with associated position vector $x_2^{(t)}$ as follows:

$$x_2^{(t)} = x_1^{(t)} + z, \tag{1}$$

where z is a d-dimensional vector. Each component of z is the realization of a normal random variable Z with mean zero and standard deviation $\sigma^{(t)}$.

Evaluation: Determine the function value $y_2 = f(x_2^{(t)})$.

Selection: Accept $X_2^{(t)}$ as $X_1^{(t+1)}$ if

$$y_2 + \tau^{(t)} < y_1, \tag{2}$$

otherwise retain $X_1^{(t)}$ as $X_1^{(t+1)}$. Increment t.

`end.`

Adaptation:

$$\text{Update} \quad \sigma^{(t)}. \quad \text{Update} \quad \tau^{(t)}. \tag{3}$$

`done.`

Fig. 1: *The hybrid evolution/threshold selection strategy (ES/TS). The two membered evolution strategy or* $(1+1)$*-ES for real-valued search spaces uses* $M = 1$ *and* $\tau^{(t)} \equiv 0$*. The symbol* f *denotes an objective function* $f : \mathbb{R}^d \to \mathbb{R}$ *to be minimized. Threshold selection (TS) uses a constant step-size* $\sigma^{(t)} \equiv \sigma$ *and a threshold adaptation scheme.*

or *mutation strength*. We will consider the following two ES-variants:

(ES-1) Constant Step Length. The basic $(1+1)$ algorithm does not modify the step-size $\sigma^{(t)}$ in Equation 3 and uses a zero threshold $\tau^{(t)}$. It is expected to be outperformed by other algorithms. However, sometimes unexpected results may occur. Probably nothing unexpected may happen, "but if something did happen, that would be a stupendous discovery" [8]. This algorithm requires the specification of a (constant) step-size $\sigma^{(t)}$ value only.

(ES-2) Step-Length Adaptation. Step-length adaptation relies on the following heuristic: *The step-size (standard deviation) should be adapted during the search. It should be increased, if many successes occur, otherwise it should be reduced.* The 1/5 success rule derived by Rechenberg [9] while analyzing the $(1+1)$-ES on two basically different objective functions for selecting appropriate step lengths can be seen as one instance of this heuristic: *From time to time during the optimization obtain the frequency of successes,*

i.e., the ratio of the number of the successes to the total number of trials (mutations). If the ratio is greater than 1/5, increase the variance, if it is less than 1/5, decrease the variance.

A more precise formulation is required to implement the 1/5 success rule. "From time to time during the optimization run" can be interpreted as "after every s_n mutations." "Increase the variation" can be implemented as a multiplication with a step-size adjustment factor. Other schemes are possible, e.g. to additive or exponential variations. The ratio of the number of the successes to the total number of mutations, the so-called *success rate* s_r, might be modified as well as the factor by which the variance is reduced or increased, the so-called *step size adjustment* factor s_a. We analyze the following two variants to implement the 1/5 rule:

(intv) A success counter $c \in \mathbb{N}_0$ is initialized at iteration $t = 1$. If a successful mutation occurs, c is increased. Every s_n iterations, the success rate is determined as c/s_n and c is set to zero.

(cont) A success vector $v^{(t)} \in \mathbb{B}^{s_n}$ is initialized at iteration $t = 1$: $v_k^{(t)} = 0$, $1 \leq k \leq s_n$. If a successful mutation occurs at iteration t, the $(1+t \mod s_n)$-th bit is set to 1, otherwise it is set to 0. After an initialization phase of s_n iterations, the success rate is determined in every iteration as $\sum_{k=1}^{s_n} v_k^{(t)}/s_n$.

The related algorithm designs are summarized in Table 1.

Table 1: *Factors of the two membered evolution strategy. Based on the default values, the step size σ is multiplied by 0.85, if the success rate is larger than $1/s_r = 1/5$ or equivalently, if more than 20 out of 100 mutations have been successful.*

Symbol	Factor	Range	Default
s_n	adaptation interval	$\mathbb{N}$	100
s_r	1/success rate	$\mathbb{R}_+$	5
s_a	step size adjustment factor	$\mathbb{R}_+$	0.85
$\sigma^{(0)}$	starting value of the step size σ	$\mathbb{R}_+$	1
$s_{1/5}$	step size update rule	{intv, cont }	cont

4 Threshold Selection Algorithms

Threshold rejection (TR) and *threshold acceptance* (TA) are complementary strategies. Threshold rejection has been proposed as a selection method for evolutionary algorithms, that accepts new candidates if their function values are significantly better than the values of the other candidates [10]. "Significant" is equivalent to "by at least a margin of τ". Threshold acceptance accepts a new candidate even if its function value is worse [11–13]. The term *threshold selection* (TS) subsumes both selection strategies. The hybrid approach presented in

this paper analyzes how threshold selection can be integrated into the (1+1)-ES strategy (Figure 1). Threshold selection provides the opportunity to escape from local optima and is implemented in many algorithms, for example in simulated annealing: *During the initial iterations of a search algorithm it is advantageous to accept worse candidate solutions. The probability of accepting a worse candidate should be continuously reduced as the number of iterations increases.* However, it is a kind of art to choose a suitable annealing schedule [13]. The annealing schedule can be seen as one instance of a more general acceptance heuristic: *The probability of accepting a worse candidate solution should be adapted during the search. It should be reduced if a candidate solution is accepted, otherwise it should be increased.* We implemented three variants of threshold selection that have been integrated into evolution strategies. Table 2 summarizes the factors used in the threshold selection algorithms.

(TS-1) Constant. To integrate a threshold mechanism into an $(1 + 1)$-ES, a non-zero threshold value τ has to be determined. This threshold affects Equation 2.

(TS-2) Linear. This variant modifies the threshold value linearly. Negative threshold values values are increased during the search process as follows: $\tau^{(t)} = \tau(-1 + t/t_{\max})$, with $\tau^{(t)} \in [-\tau, 0]$. If positive threshold values are specified, the rule $\tau^{(t)} = \tau(1 - t/t_{\max})$, with $\tau^{(t)} \in [\tau, 0]$ is used to modify the threshold.

(TS-3) Self-adaptive. We integrated a self-adaptive annealing schedule into the algorithm. Although there is no obvious analogy for the "temperature" T with respect to the optimization problem, we will use T, because it is an established term to describe the variation of the acceptance probability during the search process. Let y_i denote the function values as defined in Figure 1. If the mutation was successful, the temperature T is modified according to $T = T/(1 + b\tau T)$, otherwise $T = T/(1 - \tau T)$. The new candidate solution is accepted with probability $\exp(\delta/T)$, with $\delta = y_2 - y_1$ (Equation 2). Note, that τ defines how much the temperature (that determines the probability of accepting a worse candidate solution) is decremented at each step as the cooling proceeds, and b specifies a balance factor.

Symbol	Factor	Range	Default
τ	threshold value	$\mathbb{R}$	0
b	balance factor	$\mathbb{R}_+$	5
σ	value of the step size	$\mathbb{R}_+$	1

Table 2: *Factors of the threshold selection strategies. Note, that τ influences the acceptance probability in the self-adaptive threshold heuristic.*

5 Experiments

Classical experimental approaches in evolutionary computation (i) define a set of test (standard) functions, (ii) run a certain number of algorithms, and (iii) finally compare the obtained results. The new experimentalism proposes a different methodology: (i) Formulate a set of questions (hypotheses or goals), (ii) select an appropriate set of test functions, (iii) run a certain number of algorithms, and (iv) search for environments in which these results cannot be repeated. However, the new experimentalism can benefit from the huge number of test functions available in the optimization literature. Besides standard measures to determine the algorithm's performance such as the average, median, minimum, maximum function values, and associated standard deviations, we report a measure based on bootstrap, that reflects the goals of optimization practitioners to select the best results from several runs and to skip the others:

1. Generate n results.
2. `repeat` k times:
   (a) Select (without replacement) a set M_i of $m < n$ values.
   (b) Determine $m_i := \min M_i$.
   `end.`
3. Calculate $\sum_i^k m_i/k$. The resulting value will be referred to as $\min_{\text{boot}}$.

The first goal of our experimental analysis is to find a suitable algorithm design x^*_{1+1} for the $(1+1)$ ES. The next goal is to find environments where this design does not work. The final analysis tries to find explanations, why special environments do not permit a generalization of the results found so far. If not stated otherwise, the methods used in this article do not require any assumption on the underlying distributions.

5.1 How to Determine a Good Algorithm Design?

Classical designs such as fractional factorial designs are used in this pre-experimental screening phase to eliminate worse algorithm designs. A very simple configuration, which uses the sphere function $\sum_i^d x_i^2$, was chosen first (Table 3). In the second step, more complex situations have been analyzed (varied starting points, increased dimension). Forthcoming papers will investigate more complex objective functions that introduce multi-modality or noise. Starting points

Table 3: *Problem design for the first pre-experimental experiments to determine a fair experimental setup: n denotes the number of repeated runs, $t_{\max}$ is the number of function evaluations, d the problem dimension, and $x^{(0)}$ is the starting point.*

Design	n	$t_{\max}$	d	$x^{(0)}$
$x^{(0)}_{\text{sphere}}$	50	1000	1	100
$x^{(1)}_{\text{sphere}}$	50	1000	1	10:100
$x^{(2)}_{\text{sphere}}$	50	250	2	10:100
$x^{(3)}_{\text{sphere}}$	100	1000	$\{1, 2, 5, 10\}$	100
$x^{(4)}_{\text{sphere}}$	100	10^6	10:10:60	100

have been initialized deterministically (DETEQ), the run terminated after $t_{\max}$ function evaluations (EXH), and the mean best function value from n runs was reported (MBST) [14]. The problem design $x^{(0)}_{\texttt{sphere}}$from Table 3 was used to generate *run length distributions* (RLDs) [15]. The RLDs gave valuable hints to determine $t_{\max}$, the maximum number of function evaluations for the comparisons and thereby to avoid floor and ceiling effects. These effects occur if the problem is too easy or too hard, respectively. The success limit was set to 10^{-6}, that means an optimization run was classified as successful, if it was able to determine a candidate solution x with $f(x) < 10^{-6}$.

The (1 + 1)-ES with algorithm design $x^{(0)}_{1+1}$ from Table 4 was chosen for this analysis. Designs from this table are used during the screening phase to detect outliers that can disturb the analysis. Note, interactions between factors can be more important than main factor effects [14]. An analysis of the RLDs from experiments that are based on algorithm design $x^{(2)}_{1+1}$ and problem design $x^{(0)}_{\texttt{sphere}}$reveals that a budget of $t_{\max} = 500$ function evaluations is sufficient. After 1000 function evaluations, only 50 percent of the runs with $s_a = 1$ attained the pre-specified function value (here: 10^{-6}), whereas 100 percent of the runs with $s_a = 0.9$ attained this border already after 300 function evaluations. This is a positive effect of the step-size adaptation on the performance: a step-size adjustment factor s_a of 1 keeps the step-width constant, whereas $s_a = 0.9$ enables an adaptation that is based on the success rate.

Design	s_n	s_r	s_a
$x^{(0)}_{1+1}$	$\{10, 20, 100\}$	$\{2, 5, 10\}$	$\{0.5, 0.75, 0.9\}$
$x^{(1)}_{1+1}$	10	5	0.75
$x^{(2)}_{1+1}$	$\{10, 25\}$	5	$\{0.9, 1\}$
$x^{(3*)}_{1+1}$	2	7.25	0.758

Table 4: *(1 + 1)-ES: Algorithm designs to calibrate the experimental design and to avoid floor or ceiling effects. Threshold $\tau = 0$ and initial step size $\sigma^{(0)} = 1$ have been used.*

SPO suggests to vary settings of problem design to guarantee that the observed effect was not caused by one specific situation. Therefore, we analyzed how algorithm designs scale with the problem dimension. Figure 2 depicts the relationship between problem dimension and the empirical cumulative density function of the number of function evaluations to reach a pre-specified goal. In a similar manner as the dimension was varied, different starting points have been used. The design $x^{(5)}_{\texttt{sphere}}$ with $t_{\max} = 500$, $d = 2$, and $x^{(0)} = 100$ has been determined in this pre-experimental phase and will be used for the following experiments.

5.2 A Comparison of Different Heuristics

The algorithms will be fine-tuned in this section to generate results that enable a fair comparison. In the first experiments, the algorithm with constant step-sizes

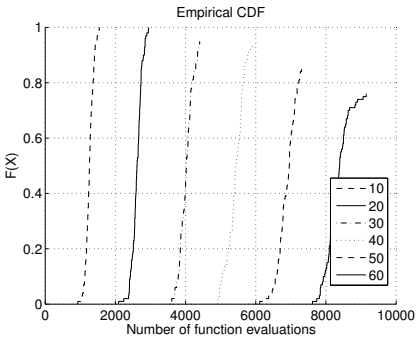

Fig. 2: *Run-length distributions for 10-60 dimensional sphere functions. Increasing the problem dimension from d to $d+1$ requires approximately 100 additional function evaluations to obtain a similar solution. Algorithm design $x^{(1)}_{1+1}$ and problem design $x^{(4)}_{\texttt{sphere}}$.*

(x_{const}) has been analyzed (Figure 3). Two variants of problem design $x^{(5)}_{\texttt{sphere}}$ have been used: $x^{(6)}_{\texttt{sphere}}$ varies problem dimensions (d=1:1:10), whereas $x^{(7)}_{\texttt{sphere}}$ uses different starting points $x^{(0)} \in \{100, 300, 700, 1000\}$.

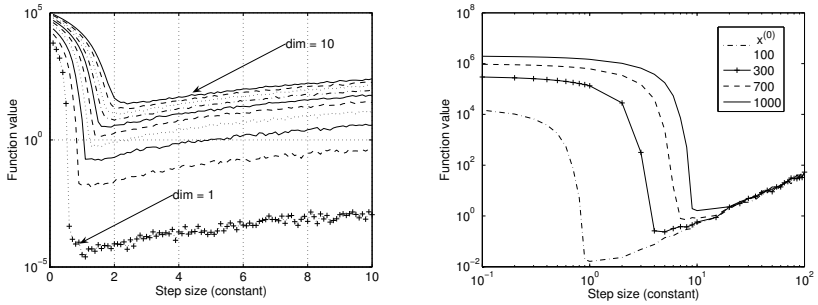

Fig. 3: *Constant step length. Sphere function. Dimensions and starting points modified. Left: $x^{(6)}_{\texttt{sphere}}$ and x^{*}_{const}. Right: $x^{(7)}_{\texttt{sphere}}$ and x^{*}_{const}. Constant step-sizes of $\sigma = 1$ or $\sigma = 1.5$ appear to be useful.*

The second series of experiments have been performed to analyze the influence of the success-rate determination scheme on the performance of the two-membered evolution strategy (problem design $x^{(5)}_{\texttt{sphere}}$ and the algorithm designs from Table 5). SPO will be used to fine-tune the algorithm design detected during the pre-experimental phase. A comparison of the RLDs shows only minor differences between the variants $x^{(4*)}_{1+1}$ and $x^{(5*)}_{1+1}$, e.g. $\min_{\text{boot}}$= 5.30e-40 and 1.28e-42 respectively (Table 6). A plot of the observed difference [14] was used to analyze the statistical significance of their difference (Figure 4). We can conclude that there is a difference in means. If 50 (500) samples are drawn, this hypothesis would be wrong in 10 (1) out of 100 experiments. However: when A and B are different treatments with associated means μ_A and μ_B, μ_A and μ_B are certain to differ in some decimal places so that $\mu_A - \mu_B = 0$ is known in advance to be

false. The observed difference is very small and large sample sizes (e.g. 500) are necessary for its statistical significance (i.e. to obtain a small p-value). Therefore we conclude that the observed difference is not scientifically meaningful. This is Step (S-12) as described in [4].

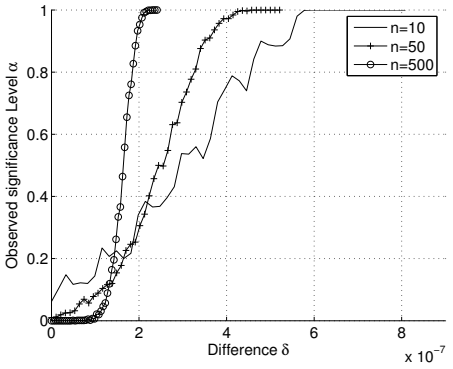

Fig. 4: *Comparing the difference in the mean between success-rate schemes intv and cont. Designs $x^{(5)}_{\texttt{sphere}}$, $x^{(4*)}_{1+1}$, and $x^{(5*)}_{1+1}$ (Table 5). 500 samples are necessary to indicate that the hypothesis "there is a difference in means as large as $2 \cdot 10^{-7}$" would be wrong only in one out of 100 experiments. See [14] for a discussion of OSL-plots.*

Why do the algorithms with the tuned designs $x^{(4*)}_{1+1}$ and $x^{(5*)}_{1+1}$ perform better than the default design? This means that algorithms with a very small memory, only two or seven bits, outperform algorithms with larger s_n values. Obviously, it takes s_n iterations to fill the memory vector. During this initial phase, no step-size adaption can occur. If the budget provides only $t_{\max} = 250$ iterations, a memory vector with more than 100 entries appears to be prohibitive. However, these considerations would explain small s_n values, but not extremely small values like $s_n = 2$. How the size of the memory vector affects the performance can be seen in Figure 5. It indicates that the $(1 + 1)$-ES is a well-scaled hill-climber. When big steps are advantageous, the algorithm takes big steps, and it takes little ones while approaching the optimum. The graph of the step size illustrates this behavior. A larger memory vector reacts too slowly, step-sizes should be adapted immediately.

Design	s_n	s_r	s_a	$s_{1/5}$
$x^{(4)}_{1+1}$	1 : 20	1 : 20	0.5 : 0.99	cont
$x^{(5)}_{1+1}$	1 : 20	1 : 20	0.5 : 0.99	intv
$x^{(4*)}_{1+1}$	7	2.40	0.83	cont
$x^{(5*)}_{1+1}$	2	2.92	0.58	intv

Table 5: *$(1 + 1)$-ES: Algorithm design to compare two success-rate determination schemes. Problem design $x^{(5)}_{\texttt{sphere}}$, $\tau = 0$ and $\sigma^{(0)} = 1$ for all experiments.*

Hybrid approaches have been considered next: Two factors, that are held constant during the optimization run, are necessary to specify the algorithm design of the first hybridization, that uses constant step-sizes and constant threshold values (x_{csct}): the step-size σ and the threshold τ. The experiments reveal

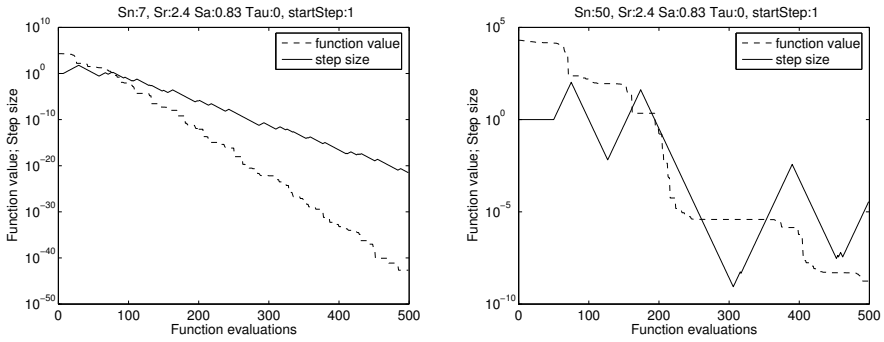

Fig. 5: *The memory effect: the influence of different memory vector sizes on the search process.* $s_r = 2.4$*,* $s_a = 0.831$*, and* $s_n = 7$ *(left) or* $s_n = 50$ *(right). The plot of the logarithm of the function value over time in the left figure shows a straight line. The problem design* $x^{(5)}_{\texttt{sphere}}$ *was used for these experiments.*

that algorithms with zero threshold values perform best. Step sizes about 1 are preferred for the hybrid metaheuristic with constant step-sizes and constant threshold (x_{csct}). Results from the other hybridizations (constant step-sizes, linear threshold (x_{cslt}) and constant step-sizes with self-adaptive threshold (x_{sann})) indicate that non-zero threshold values worsen the algorithm's performance.

The next series of experiments have been set up to analyze whether constant or deterministically modified threshold schemes can improve the performance of evolution strategies. Experiments clearly indicate that non-zero threshold values worsen the algorithm's performance in this situation, too. Even the self-adaptive threshold scheme (x_{temp}) does not improve the algorithm's performance. The results from the experiments are summarized in Table 6. To improve comparability, results from a Nelder-Mead simplex (NMS) optimization have been added [16]. The Nelder-Mead algorithm requires the specification of four parameters (coefficients of reflection, expansion, contraction, and shrinkage), that have been tuned with SPO. The NMS optimization was able to find a candidate solution with function value 1.15e-77, a result that is significantly better then the ES/TS results. However, NMS fails completely on the sphere function in higher-dimensional cases, e.g. the best function value for $d = 50$ reads 4.77e+05, whereas the (1+1)-ES can cope with these problems (Fig. 2). This is an inherent problem of the NMS and not due to problems with the algorithm design.

Based on local performance measures, it can theoretically be shown that ES benefits from TS [10, 14] under noise. Additive Gaussian noise has been added to the objective function from design $x^{(5)}_{\texttt{sphere}}$. The algorithm design $x^{(5*)}_{1+1}$, that has been tuned on the sphere function without noise, was used for the first experiments. Constant non-zero threshold values (TS-1) were able to improve the performance significantly. However, after applying SPO to $x^{(5*)}_{1+1}$ in the noisy environment, the tuned algorithm design performed better without threshold.

Table 6: *Experimental results from the hybridizations of ES and TS. Problem design* $x^{(5)}_{\text{sphere}}$ *was used.*

Algorithm	Mean	Median	Sd	Min	Max	$\min_{\text{boot}}$
x^{*}_{const}	1.65e-02	1.05e-02	1.78e-02	1.09e-05	1.46e-01	1.74e-05
$x^{(4*)}_{1+1}$	9.67e-27	2.07e-32	9.07e-26	4.98e-40	9.06e-25	5.30e-40
$x^{(5*)}_{1+1}$	2.73e-25	2.52e-34	2.57e-24	2.61e-44	2.57e-23	1.28e-42
x^{*}_{csct}	2.18e+03	399.43	4.89e+03	0.4656	1.89e+04	0.9612
x^{*}_{cslt}	0.3414	0.0164	2.6340	1.72e-06	25.9286	5.14e-04
x^{*}_{sann}	0.0160	0.0099	0.0171	5.38e-04	0.1013	5.82e-04
x^{*}_{τ}	0.0484	0.0410	0.0342	9.23e-04	0.1361	0.0012
$x^{*}_{\tau(t)}$	0.0474	0.0414	0.0340	8.54e-04	0.2327	0.0014
x^{*}_{temp}	7.21e-07	5.24e-07	7.95e-07	1.04e-08	5.67e-06	1.62e-08
NMS	1.15e-77	–	0	–	–	–

These experiments indicate that there are situations (under noise), in which a combination of ES and TS might be beneficial.

6 Summary and Conclusion

The paper demonstrated the huge potential for the new experimentalism in computer science. Good algorithm designs can lead to impressive performance improvements and to robust algorithms that can be constructed systematically. SPO provides means for an in-depth understanding and fair comparisons of algorithms. The framework of the new experimentalism can be used to determine if statistically significant results are scientifically meaningful.

The SPO approach presented here can easily be applied to other algorithm–problem combinations. A recent paper discusses three scenarios to demonstrate its flexibility: (i) to analyze newly developed algorithms, (ii) to compare well-known algorithms, and (iii) to apply algorithms efficiently and effectively to complex real-world optimization problems [6]. Or, consider for example binary search spaces: mutation can be realized by random bit-flips of the position vector $x^{(t)}_1$. The probability p_m of flipping a bit can be regarded as the pendant to the mutation strength σ. Or, travelling salesperson problems can be regarded as ordering problems that require combinatorial search spaces. A search step operator defines the number of states n_s that can be reached from a parental state (neighborhood) within one move step. The number of move steps s can be seen as a pendant to the mutation strength σ.

The sphere function has been chosen as a test-function with a calculable influence on the results. Evolution strategies require only a small memory vector while optimizing the sphere—too much information (memory) is debilitating. No difference between the two step-size adaptation schemes (intv and cont) could be observed. Is this also true for higher dimensions and other test-functions?

ES clearly outperformed TS on the sphere due to its self-adaptiveness. Nevertheless, there may be other environments (problem designs), in which a hybrid

approach is beneficial (noise, multi-modality, combinatorial optimization problems). Following the methodology presented in this paper, we are seeking for environments in which the step-size adaptation does not work and hybrid approaches can be improve the performance.

References

1. H. H. Hoos and T. Stützle, *Stochastic Local Search—Foundations and Applications*. Elsevier, 2005.
2. D. G. Mayo, *Error and the Growth of Experimental Knowledge*. The University of Chicago Press, 1996.
3. T. Bartz-Beielstein and S. Markon, "Tuning search algorithms for real-world applications: A regression tree based approach," in *Proc. 2004 Congress on Evolutionary Computation (CEC'04), Portland OR*, G. W. Greenwood, Ed., vol. 1. Piscataway NJ: IEEE Press, 2004, pp. 1111–1118.
4. T. Bartz-Beielstein, K. E. Parsopoulos, and M. N. Vrahatis, "Design and analysis of optimization algorithms using computational statistics," *Applied Numerical Analysis & Computational Mathematics (ANACM)*, vol. 1, no. 2, pp. 413–433, 2004.
5. C. Lasarczyk and W. Banzhaf, "Total synthesis of algorithmic chemistries," in *GECCO 2005: Proceedings of the Genetic and Evolutionary Computation Conference*, 2005, in print.
6. T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss, "Sequential parameter optimization," in *Proc. 2005 Congress on Evolutionary Computation (CEC'05), Edinburgh*. Piscataway NJ: IEEE Press, 2005, in print.
7. H.-P. Schwefel, *Evolution and Optimum Seeking*, ser. Sixth-Generation Computer Technology. New York: Wiley Interscience, 1995.
8. I. Hacking, *Representing and intervening*. Cambridge University Press, 1983.
9. I. Rechenberg, *Evolutionsstrategie. Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart: frommann-holzboog, 1973.
10. S. Markon, D. V. Arnold, T. Bäck, T. Beielstein, and H.-G. Beyer, "Thresholding – A selection operator for noisy ES," in *Proc. 2001 Congress on Evolutionary Computation (CEC'01), Seoul*, J.-H. Kim, B.-T. Zhang, G. Fogel, and I. Kuscu, Eds. Piscataway NJ: IEEE Press, 2001, pp. 465–472.
11. J. Matyáš, "Random Optimization," *Automation and Remote Control*, vol. 26, no. 2, pp. 244–251, 1965.
12. E. C. Stewart, W. P. Kavanaugh, and D. H. Brocker, "Study of a global search algorithm for optimal control," in *Proceedings of the 5th International Analogue Computation Meeting, Lausanne*, Aug.-Sept. 1967, pp. 207–230.
13. G. Dueck and T. Scheuer, "Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing," *Journal of Computational Physics*, vol. 90, pp. 161–175, 1990.
14. T. Bartz-Beielstein, "New experimentalism applied to evolutionary computation," Ph.D. dissertation, University of Dortmund, April 2005.
15. H. H. Hoos, "Stochastic local search – methods, models, applications," Ph.D. dissertation, Technische Universität Darmstadt, 1998.
16. J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright, "Convergence properties of the Nelder–Mead simplex method in low dimensions," *SIAM J. on Optimization*, vol. 9, no. 1, pp. 112–147, 1998.