Considerations of budget allocation for Sequential Parameter Optimization (SPO)

Thomas Bartz–Beielstein** and Mike Preuss

Dortmund University, 44221 Dortmund, Germany,
http://ls11-www.cs.uni-dortmund.de/people/

Abstract. Obviously, it is not a good idea to apply an optimization algorithm with wrongly specified parameter settings, a situation which can be avoided by applying algorithm tuning. Sequential tuning procedures are considered more efficient than single-stage procedures. [1] introduced a sequential approach for algorithm tuning that has been successfully applied to several real-world optimization tasks and experimental studies. The sequential procedure requires the specification of an initial sample size k. Small k values lead to poor models and thus poor predictions for the subsequent stages, whereas large values prevent an extensive search and local fine tuning. This study analyzes the interaction between global and local search in sequential tuning procedures and gives recommendations for an adequate budget allocation. Furthermore, the integration of hypothesis testing for increasing effectiveness of the latter phase is investigated.

1 Introduction

This work deals with a tuning procedure, the *sequential parameter optimization* (SPO) [1]. It does not challenge applying tuning methods for experimental analysis of optimization algorithms per se. Nor does it focus on any of the particular questions such methods may help to answer as has been done elsewhere [2], although our investigations shall lead to additional insight in this respect, too. Let us assume that SPO is useful for obtaining, e.g., algorithm designs for (near-) optimal performance and parameter interactions. However, the tuning procedure itself is not as fixed as may seem: Integration of several techniques (*Design and analysis of computer models* (DACE), space filling designs, noise reduction by increase of repeats) left some of the interfaces in a rather provisional state. Stated differently, we have to admit that even this parameter tuning method *does have parameters.*

As neither theoretical results nor a long empirical tradition of applying SPO exist, it remains unclear how appropriate the currently employed default methods and values are. The SPO tuning procedure is model-based and thus consists of two subsequent phases: a) building a first model, and b) repeatedly locating and evaluating points with maximal expected improvement. The latter phase shall lead to good solutions for the tuning problem and a better model at the same time. Given that the tuned system usually contains nondeterministic optimization algorithms, e.g. *evolutionary algorithms* (EAs), three questions emerge:

1. How much effort shall be put into obtaining the first model?
2. Which method shall be utilized for determining an initial set of design points?
3. How often shall second-phase design points be evaluated?

We consider it impossible to answer these questions in a too general context, without further specifying the tuned algorithm–problem system. However, focusing on a very small number of cases implies the danger of generating results without much practical value due to loss of transferability to other such systems. We face this difficulty by selecting test problems that are different in their most important properties, e.g., unimodal vs. multimodal and real-valued vs. binary coded problems. Furthermore, the chosen algorithm-problem systems are well known, thereby enabling to utilize the experience collected within the *evolutionary computation* (EC) community over years. But still, our

**corresponding author: Thomas.Bartz-Beielstein@udo.edu

Algorithm 1 Sequential parameter optimization

procedure SPO($\mathcal{D}_A$, $\mathcal{D}_P$) ▷ Algorithm und problem design
  Select $p \in \mathcal{D}_P$ and set $t = 0$ ▷ Select problem instance
  $X_A^{(t)} = \{x_1, x_2, \ldots, x_k\}$ ▷ Sample k initial points, e.g., LHS
  repeat
    $y_{ij} = Y_j(x_i, p) \forall x_i \in X_A^{(t)}$ and $j = 1, \ldots, r^{(t)}$ ▷ Fitness evaluation
    $\overline{Y}_i^{(t)} = \sum_{j=1}^{r^{(t)}} y_{ij}^{(t)} / r^{(t)}$ ▷ Sample statistic for the ith design point
    x_b with $b = \arg\min_i(\overline{y}_i)$ ▷ Determine best point
    $Y(x) = \mathcal{F}(\beta, x) + Z(x)$ ▷ DACE model from Eq. 1
    $X_S = \{x_{k+1}, \ldots, x_{k+s}\}$ ▷ Generate s sample points, $s \gg k$
    $y(x_i)$, $i = 1, \ldots, k+s$ ▷ Predict fitness from the DACE model
    $I(x_i)$ for $i = 1, \ldots, s+k$ ▷ Expected improvement, cf. [6]
    $X_A^{(t+1)} = X_A^{(t)} \cup \{x_{k+i}\}_{i=1}^{m} \notin X_A^{(t)}$ ▷ Add m promising points
    if $x_b^{(t)} = x_b^{(t+1)}$ **then**
      $r^{(t+1)} = 2r^{(t)}$ ▷ Increase number of repeats
    end if
    t = t+1 ▷ Increment iteration counter
  until Budget exhausted
end procedure

study can by no means be comprehensive; a certain arbitrariness remains and will only vanish when much more experimental effort than practicable in this work has been put into exploration and analysis of SPO performance. Although we restrict ourselves here to SPO as the only employed tuning procedure, the obtained conclusions may still be of interest for the application of other model-based methods, in particular such that are built on regression techniques. These have to face the same three questions enumerated above.

In the following section, we give a short summary of the SPO tuning procedure, going into details only insofar as necessary for discussing our experimental results. Section 3 discusses initial designs issues, whereas Sect. 4 considers second-phase points. The paper closes with a summary and an outlook in Sect. 5.

2 Sequential Parameter Optimization

SPO is a methodology for the experimental analysis of optimization algorithms to determine improved *algorithm designs*[1] and to learn, how the algorithm works. It employs computational statistic methods to investigate the interactions among optimization problems, algorithms, and environments. We consider each algorithm design with associated output as a realization of a stochastic process. We apply stochastic process models as introduced in [3], [4], and [5] to optimization algorithms such as EAs and *particle swarm algorithms* (PSO). Consider a set of m design points $x = (x^{(1)}, \ldots, x^{(m)})^T$ with $x^{(i)} \in \mathbb{R}^d$. In the DACE *stochastic process model*, a deterministic function is evaluated at the m design points x. The vector of the m responses is denoted as $y = (y^{(1)}, \ldots, y^{(m)})^T$ with $y^{(i)} \in \mathbb{R}$. The process model proposed in [3] expresses the deterministic response $y(x^{(i)})$ for a d-dimensional input $x^{(i)}$ as a realization of a regression model[2] $\mathcal{F}$ and a stochastic process Z,

$$Y(x) = \mathcal{F}(\beta, x) + Z(x). \tag{1}$$

Note that SPO inherits this process model but utilizes it for the situation of nondeterministic responses as these are usually obtained from the heuristic optimization algorithms we want to

[1] These contain all parameter settings determining a specific algorithm instance, whereas *problem designs* subsume parameters related to the optimization problem and domain specific restrictions such as problem dimension or allowed function evaluations.

[2] Based on experiences from previous studies, quadratic regression models have been used in our studies.

investigate. Algorithm 1 [2] describes the SPO in a formal manner. [3] The selection of a suitable problem instance is done in the pre-experimental planning phase to avoid floor and ceiling effects (l.2). Latin hypercube sampling can be used to determine an initial set of design points (l.3). After the algorithm has been run with these k initial parameter settings (l.5), the DACE process model is used to discover promising design points (l.10). Note that other sample statistics than the mean, e.g., the median, can be used in l.6. The m points with the highest expected improvement[4] are added to the set of design points, where m should be small compared to s. The update rule for the number of reevalutions $r^{(t)}$ (l.13-15) guarantees that the new best design point $x_b^{(t+1)}$ has been evaluated at least as many times as the previous best design point $x_b^{(t)}$. Obviously, this is a very simple update rule and more elaborate rules are possible. Other termination criteria exist besides the budget based termination (l.17).

3 Initial Designs

There is no simple or generic rule for choosing adequate initial designs. In classical *design and analysis of experiments* (DoE), this is a chicken and egg problem: To determine a suitable regression model, design points are needed. But, the choice of optimal design points depends on the model [7]. Experiments reported in [8] indicated the superiority of space filling design over classical factorial designs in the context of parameter tuning for stochastic search algorithms. Therefore, we consider space-filling designs, which are nicely motivated in [4]. So, are we lucky and can simply ignore the complex determination of adequate designs by filling the design space randomly with design points? Unfortunately not, because even with space filling designs, certain design criteria can be chosen. We will use Latin hypercube designs (LHD) for the following experiments. LHDs haven been chosen because they are easy to understand and implement—and not because they are proven to be superior to other space filling designs. [6, p. 149] state: *It has not been demonstrated that* LHD*s are superior to any designs other than simple random sampling (and they are only superior to simple random sampling in some cases).*

In many real-world optimization scenarios, a limited budget, say time or function evaluations, for performing the optimization task is available. At least two situations, in which an allocation of the resources is possible, can be mentioned here: First, the distribution of the available resources among the tuning phase and the optimization phase, and second, the allocation of the resources to the initial design and the second phase (sequential steps) during tuning. We will consider the latter problem in this paper. The analysis will be restricted to the case where the number of initial repeats is very low, e.g., $r = 2$. This is reasonable, because (i) in many real-world optimization scenarios large r values are prohibitive and (ii) we are interested in a quick exploration of the search space in the first step of this sequential procedure. So, the experimental goal can be formulated in the SPO framework as follows:

Goal 1. For a given number of function evaluations, N_b, determine a number k of initial points for the initial design as introduced in Step 3 of Algorithm 1.

Note, that this goal occurs in any sequential tuning procedure and is not restricted to SPO. And, in general, sequential procedures are considered more efficient than procedures that use the available budget at once [9]. Therefore, we are considering the most interesting cases and our results are of interest for other situations as well. The trade-off can be described as follows: Shall we use a good initial design with only a few sequential steps or perform many sequential steps based on a poor initial design?

[3] A toolbox that implements SPO and provides additional material is available at: http://www.springer.com/3-540-32026-1.

[4] A situation in which each point has the same expected improvement can occur if (i) the objective function is constant and (ii) the optimization algorithm is deterministic. However, these situations are only of theoretical relevance.

A PSO was chosen to illustrate our approach and to generate some empirical data. The algorithm and results from experiments are outlined in [10]. Our PSO implementation is based on the PSOTOOLBOX, which was integrated into the SPO framework. The PSO algorithm design is shown in Table 1. To specify the problem design, the following parameters were considered (Table 2). The experiment's name, the number of runs n, the maximum number of function evaluations $t_{\max}$, the problem's dimension d, the initialization method, the termination criterion, the interval $[x_l, x_u]$ for the initialization of the object variables, as well as the optimization problem and the performance measure (PM) are reported. The reader is referred to [2] for a discussion of these parameters. Finally, we have to describe the SPO design (Table 3). If no sequential steps were performed, n_{LHD} design points can be evaluated. Note, that $n_{\text{LHD}} = N_b/r$, where N_b denotes the budget, i.e., the total number of algorithm runs, and r is the number of repeats used in the first design. The SPO uses a quadratic regression model and a Gaussian correlation function. Four design points with the highest *expected improvement* and the best solution found so far are evaluated in the sequential phase of the SPO run ($m = 4$).The experimental analysis clearly demonstrated that the determination of a suitable initial design is of crucial importance for the second phase, which performs a local tuning. To play safe, we recommend increasing the number of initial design points. The number of sequential optimization steps could be reduced in many situations without a significant performance loss. Furthermore, experiments with several optimization problems (Rosenbrock, Sphere, Rastrigin, Griewank, and problems from the Moré test set) revealed that a small number of function re-evaluations is beneficial. To give an example: The following best function values have been obtained on the 30-dim Rosenbrock function (with $t_{\max} = 500$ function evaluations for the PSO and $r = 2$ reevaluations for the initial design): 223 with 15 initial LHD samples, 324 with 100 initial samples, and 486 with 150 initial samples. If the number of reevaluations was increased to $r = 5$, a function value of 1663 was obtained with 15 initial LHD sample points and values of 403 (2776) with 30 (60) initial samples. All values reported here are averages from 100 repeats. [5]

4 Evaluation of Second-Phase Points

SPO subsequently evaluates second-phase points (algorithm designs) for two reasons: a) to find optimal designs, and b) to improve the model. In order to counteract the nondeterministic nature of the modeled algorithm-problem system, the standard procedure doubles the number of repeats performed of the current best point whenever an iteration fails to provide an improvement, cf. l. 14 from Algorithm 1. Consequently, to enable fair comparison between the current best and newly suggested point(s), SPO ensures that both have been sampled for the same number of times. However, this procedure can become expensive in terms of algorithm runs, resulting in rather low numbers of second-phase points if the available budget is tight.

If we had means to decide which of the points (best or suggested) leads to better performance without necessarily carrying out the suggested number of repeats in full, algorithm runs could be saved, resulting in a higher number of second-phase points without great loss of information. Concerning the search for optimal designs, this is undoubtedly an advantage. However, it could be argued that reducing the number of repeats also entails reducing the intended model improvement. Nevertheless, we have to take into account that a) second-phase points are usually evaluated much more often than first-phase points, b) only clearly worse performing points can easily be rejected, and c) saved runs are invested again into new points in the following iterations.

Table 1. PSO algorithm designs. Swarm size s, cognitive and social parameters c_1 and c_2, and $w_{\text{iterScale}}$, the percentage of the run, in which the inertia weight is decreased.

Design	s	c_1	c_2	$w_{\text{iterScale}}$
$x^{(0)}_{\text{PSO}}$	2 : 50	1 : 3	1 : 3	0.1:0.9

[5] Note, that [10] used 80 times more function evaluations to obtain similar results. They used recommendations from theory to perform their experiments, e.g., theoretically derived values for c_1 and c_2.

Table 2. Problem designs for the sphere function. Similar designs have been used for the Rosenbrock, Griewank, and Rastrigin function. The maximum number of function evaluations for one algorithm run is $t_{\max}$, whereas n denotes the number of repeats for each algorithm run. Altogether, $n \times t_{\max}$ function evaluations were performed. See [10]

Design	Init.	Term.	PM	n	$t_{\max}$	d	x_l	x_u
$x^{(0)}_{\texttt{sphere}}$	DETEQ	EXH	MBST	100	500	$\{10, 20, 30\}$	15	30

Table 3. SPO algorithm designs for tuning the particle swarm optimization. "Merge" denotes the statistic that was used to compare the performance of algorithm designs with stochastically disturbed results, e.g., from EAs

Design	LHDini	N_b	r	merge
$x^{(0)}_{\text{SPO}}$	15:n_{LHD}	{100,300,1000}	2:5	{min, median, mean, max }

Assessing significant differences of two underlying systems represented by a number of samples is the intended purpose of hypothesis tests. However, the often applied t-tests require approximate normal distributions, a demand that is usually not met by results obtained from heuristic optimization algorithms. As an alternative, we therefore resort to bootstrap permutation tests ([11], section 14.5) which do not presume normality but instead anticipate identically shaped distributions. Fortunately, permutation tests are robust against small differences in shapes so that we regard this prerequisite as more realistic then to expect normality.

Goal 2. Detect if integrating bootstrap permutation tests for early rejection of bad algorithm designs can significantly improve SPO performance.

As test problems, we chose a real-valued and a binary coded one, the 30-dimensional shifted Rastrigin function from [12], and the 20-dimensional (120 bits in 20 groups) *massively multimodal deceptive problem* MMDP [13]. On these, a naïve evolution strategy (ES) as described in [14] is run, with gaussian mutation and self-adaptation for the real-valued and bit flip mutation for the binary coded problem. Additionally, the κ selection operator allows to switch between comma and plus type selection by imposing a maximum age for any individual. SPO is utilized to find good algorithm designs with a maximum run length of 10^4 evaluations and a budget of 1000 allowed algorithm runs, of which 200 (50×4 repeats) are used for the initial LHD.

Figure 1 presents box plots of the best found algorithm designs detected by SPO in 20 independent runs, with and without integrating bootstrap permuation tests. Note that the setting is deliberately chosen to be hard for the ES, so that the global optimum (-330 for the Rastrigin problem, 20 for the MMDP) is rarely attained. SPO obviously performs significantly better in both cases when tests are used. A Wilcoxon rank sum test computes the probability (p-value) for wrongly rejecting the null-hypothesis (both variants perform equally) to 0.01121 for the Rastrigin function, and 10^{-6} for the MMDP. This is suprising as we expected that result distributions consisting only of discrete values were harder to compare and thus the impact of testing for the ES on the MMDP would be small. Nevertheless, these two cases only demonstrate that there *can be* a positive effect. Further experimentation is necessary to find out if this result correctly hints to a general trend.

5 Summary and Outlook

The choice of an appropriate initial design is important for SPO. Selection of good algorithm parameter settings from initial designs produces the largest performance gain. In some situations,

Table 4. ES algorithm designs. The learning rate τ applies to the Rastrigin function, the mutation rate p_{mut} to the MMDP, resulting in 4 free parameters each.

Design	μ	κ	$\frac{\lambda}{\mu}$	τ	p_{mut}
$x^{(0)}_{\text{ES}}$	1 : 200	1 : 20	4 : 12	0.001 : 1.0	0.001 : 0.999

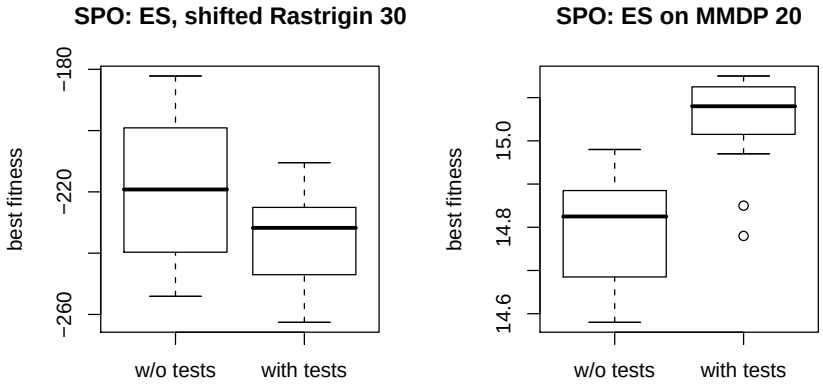

Fig. 1. Distribution of best algorithm design performances from 20 SPO runs of an ES on the shifted Rastrigin function (left, minimization) and the discrete MMDP problem (right, maximization), with and without applying bootstrap permutation tests.

e.g., where no complex stochastic process models are available (or desired), this first improvement can be sufficient. Selecting the best point from a set of space filling design points—as used in the initial phase of the SPO approach—can be regarded as a simple heuristic. This heuristic works surprisingly well and should become a standard in experimental research. It is of great importance to screen out worse algorithm design points.

Furthermore, SPO was able to improve the algorithm's performance drastically. For example, [10] report a mean best fitness value of $\overline{y} = 316$ on the 30-dimensional Rosenbrock function after 40,000 function evaluations. We were able to obtain an even slightly better result ($\overline{y} = 202$) with 500 function evaluations only. Problem instance selection plays an important role in the context of SPO and active experimentation. First, it might be of interest to determine an algorithm design that works well on a broad number of problem instances, i.e., it improves the robustness of the algorithm. Second, it is an invaluable tool in the experimentalist's arsenal for determining why an algorithm works.

As far we know is this the first analysis of the interplay between initial designs and sequential steps for the optimization of stochastic search algorithms. Several parameters, that have not varied in our study, are of interest for further investigations, e.g., other stochastic process models can be used. Summarizing, we can conclude that SPO is a robust tool for tuning (stochastic) search algorithms.

Acknowledgment The research leading to this paper was supported by the DFG (Deutsche Forschungsgemeinschaft) as part of the collaborative research center "Computational Intelligence" (531) and by project grant no. 252441, "Mehrkriterielle Struktur- und Parameteroptimierung verfahrenstechnischer Prozesse mit evolutionären Algorithmen am Beispiel gewinnorientierter unscharfer destillativer Trennprozesse".

References

1. Bartz-Beielstein, T., Parsopoulos, K.E., Vrahatis, M.N.: Analysis of particle swarm optimization using computational statistics. In Simos, T.E., Tsitouras, C., eds.: Proceedings International Conference Numerical Analysis and Applied Mathematics (ICNAAM), Weinheim, Germany, Wiley-VCH (2004) 34–37
2. Bartz-Beielstein, T.: Experimental Research in Evolutionary Computation—The New Experimentalism. Springer, Berlin, Heidelberg, New York (2006)
3. Sacks, J., Welch, W.J., Mitchell, T.J., Wynn, H.P.: Design and analysis of computer experiments. Statistical Science **4**(4) (1989) 409–435

4. Jones, D., Schonlau, M., Welch, W.: Efficient global optimization of expensive black-box functions. Journal of Global Optimization **13** (1998) 455–492
5. Lophaven, S., Nielsen, H., Søndergaard, J.: DACE—A Matlab Kriging Toolbox. Technical Report IMM-REP-2002-12, Informatics and Mathematical Modelling, Technical University of Denmark, Copenhagen, Denmark (2002)
6. Santner, T.J., Williams, B.J., Notz, W.I.: The Design and Analysis of Computer Experiments. Springer, Berlin, Heidelberg, New York (2003)
7. Pukelsheim, F.: Optimal Design of Experiments. Wiley, New York NY (1993)
8. Bartz-Beielstein, T., Markon, S.: Tuning search algorithms for real-world applications: A regression tree based approach. In Greenwood, G.W., ed.: Proceedings 2004 Congress on Evolutionary Computation (CEC'04), Portland OR. Volume 1., Piscataway NJ, IEEE (2004) 1111–1118
9. Wald, A.: Sequential Analysis. Wiley, New York NY (1947)
10. Shi, Y., Eberhart, R.: Empirical study of particle swarm optimization. In Angeline, P.J., Michalewicz, Z., Schoenauer, M., Yao, X., Zalzala, A., eds.: Proceedings of the Congress of Evolutionary Computation. Volume 3., Piscataway NJ, IEEE (1999) 1945–1950
11. Mccabe, G.P., Moore, D.S.: Introduction to the Practice of Statistics. W.H. Freeman & Company (2005)
12. Suganthan, P.N., Hansen, N., Liang, J.J., Deb, K., Chen, Y.P., Auger, A., Tiwari, S.: Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Technical report, Nanyang Technological University, Singapore (2005)
13. Goldberg, D.E., Deb, K., Horn, J.: Massively multimodality, deception and genetic algorithms. In Männer, R., Manderick, B., eds.: Parallel Prob. Solving from Nature II, North-Holland (1992) 37–46
14. Beyer, H.G., Schwefel, H.P.: Evolution strategies—A comprehensive introduction. Natural Computing **1** (2002) 3–52