Appl. Num. Anal. Comp. Math. **1**, No. 3, 413 – 433 (2004) / **DOI** 10.1002/anac.200410007

# Design and Analysis of Optimization Algorithms Using Computational Statistics

**T. Bartz–Beielstein**[*1], **K.E. Parsopoulos**[**2,3], and **M.N. Vrahatis**[***2,3]

[1] Department of Computer Science XI, University of Dortmund, D–44221 Dortmund, Germany
[2] Computational Intelligence Laboratory, Department of Mathematics, University of Patras, GR–26110 Patras, Greece
[3] University of Patras Artificial Intelligence Research Center (UPAIRC)

Received 31 October 2004, revised 2 December 2004, accepted 2 December 2004
Published online 20 December 2004

We propose a highly flexible sequential methodology for the experimental analysis of optimization algorithms. The proposed technique employs computational statistic methods to investigate the interactions among optimization problems, algorithms, and environments. The workings of the proposed technique are illustrated on the parameterization and comparison of both a population–based and a direct search algorithm, on a well–known benchmark problem, as well as on a simplified model of a real–world problem. Experimental results are reported and conclusions are derived.



## 1 Introduction

Modern search heuristics have proved to be very useful for solving complex real–world optimization problems that cannot be tackled through classical optimization techniques [1]. Many of these search heuristics involve a set of parameters that affect their convergence properties. Usually, an optimal parameter setting depends on the problem, as well as on restrictions posed by the environment, such as time and hardware constraints.

We propose an approach for determining the parameters of optimization algorithms, tailored for the optimization problem at hand (we consider only minimization cases, although the technique can be straightforwardly applied to maximization problems). The proposed approach employs techniques from computational statistics and statistical experimental design. Its workings are illustrated for both a population–based and a direct search algorithm, on a well–known benchmark problem, as well as on a simplified model of a real–life application, namely the optimization of an elevator group controller, extending the approaches proposed in [2] and [3]. For this purpose, the Particle Swarm Optimization (PSO) [4], which belongs to the class of swarm intelligence algorithms, and the Nelder and Mead Simplex method (NMS) [5] have been used.

The rest of the paper is organized as follows: the considered algorithms are briefly described in Section 2. The computational statistics background is described in Section 3, while, experimental results are reported in Sections 4 and 5. The paper closes with conclusions in Section 6.

## 2 The Considered Algorithms

### 2.1 The Particle Swarm Optimization Algorithm

PSO is a swarm intelligence optimization algorithm [6]. The main inspiration behind PSO was the flocking behavior of swarms and fish schools. PSO has proved to be very efficient in numerous applications in science and engineering [4, 7–24]. PSO's convergence is controlled by a set of parameters that are usually either determined empirically or set equal to widely used default values.

* Corresponding author: e-mail: thomas.bartz-beielstein@udo.edu, Phone: +00 49 231 9700977, Fax: +00 49 231 9700959
** e-mail: kostasp@math.upatras.gr, Phone: +30 2610 997348, Fax: +30 2610 992965
*** e-mail: vrahatis@math.upatras.gr, Phone: +30 2610 997374, Fax: +30 2610 992965

PSO belongs to the class of stochastic, population–based optimization algorithms [4]. It exploits a population of individuals to probe the search space. In this context, the population is called a *swarm* and the individuals are called *particles*. Each particle moves with an adaptable velocity within the search space, and it retains in a memory the best position it has ever visited.

There are two main variants of PSO with respect to the information exchange scheme among the particles. In the *global* variant, the best position ever attained by all individuals of the swarm is communicated to all the particles at each iteration. In the *local* variant, each particle is assigned to a neighborhood consisting of prespecified particles. In this case, the best position ever attained by the particles that comprise a neighborhood is communicated among them [4]. Neighboring particles are determined based rather on their indices than their actual distance in the search space. Clearly, the global variant can be considered as a generalization of the local variant, where the whole swarm is considered as the neighborhood for each particle. In the current work we consider the global variant only.

Assume an $n$–dimensional search space, $S \subset \mathbb{R}^n$, and a swarm consisting of $s$ particles. The $i$–th particle is an $n$–dimensional vector,

$$x_i = (x_{i1}, x_{i2}, \ldots, x_{in})^\top \in S.$$

The velocity of this particle is also an $n$–dimensional vector,

$$v_i = (v_{i1}, v_{i2}, \ldots, v_{in})^\top.$$

The best previous position encountered by the $i$–th particle (i.e., its memory) in $S$ is denoted by,

$$p_i^* = (p_{i1}^*, p_{i2}^*, \ldots, p_{in}^*)^\top \in S.$$

Assume $g$ to be the index of the particle that attained the best previous position among all the particles in the swarm, and $t$ to be the iteration counter. Then, the resulting equations for the manipulation of the swarm are [27],

$$v_i(t+1) = w v_i(t) + c_1 r_1 \left(p_i^*(t) - x_i(t)\right) + c_2 r_2 \left(p_g^*(t) - x_i(t)\right), \quad (1)$$

$$x_i(t+1) = x_i(t) + v_i(t+1), \quad (2)$$

where $i = 1, 2, \ldots, s$; $w$ is a parameter called the *inertia weight*; $c_1$ and $c_2$ are positive constants, called the *cognitive* and *social* parameter, respectively; and $r_1$, $r_2$ are vectors with components uniformly distributed in $[0, 1]$. All vector operations are performed componentwise.

Usually, the components of $x_i$ and $v_i$ are bounded as follows,

$$x_{\min} \leqslant x_{ij} \leqslant x_{\max}, \quad -v_{\max} \leqslant v_{ij} \leqslant v_{\max}, \quad j = 1, \ldots, n,$$

where $x_{\min}$ and $x_{\max}$ define the bounds of the search space, and $v_{\max}$ is a parameter that was introduced in early PSO versions to avoid swarm explosion that was caused by the lack of a mechanism for controlling the velocity's magnitude. Although the inertia weight is such a mechanism, empirical results have shown that using $v_{\max}$ can further enhance the algorithm's performance.

Experimental results indicate that it is preferable to initialize the inertia weight to a large value, in order to promote global exploration of the search space, and gradually decrease it to get more refined solutions. Thus, an initial value around 1 and a gradual decline towards 0 is considered a proper choice for $w$. If a maximum number of iterations, $t_{\max}$, has been determined, then the inertia weight can be scaled according to the following scheme: if $w_{\max}$ is the maximum value of the inertia weight, two real–valued parameters, $w_{\text{scale}}, w_{\text{iterScale}} \in [0, 1]$ are determined, such that $w$ is linearly decreased from $w_{\max}$ to $w_{\max}\, w_{\text{scale}}$, over $t_{\max}\, w_{\text{iterScale}}$ iterations. Then, for the last $t_{\max}\,(1 - w_{\text{iterScale}})$ iterations, it has a constant value, equal to $w_{\max}\, w_{\text{scale}}$. Table 1 summarizes the exogenous parameters of PSO. Proper fine–tuning of the parameters may result in faster convergence and alleviation of local minima [2, 3, 27, 28].

Different PSO versions, such as PSO with constriction factor, have been proposed [29]. In the constriction factor variant, Eq. (1) reads,

$$v_i(t+1) = \chi \left[v_i(t) + c_1 r_1 \left(p_i^*(t) - x_i(t)\right) + c_2 r_2 \left(p_g^*(t) - x_i(t)\right)\right], \quad (3)$$

**Table 1** Default settings of the exogenous parameters of the PSO algorithm. Similar designs have been used in [26] to optimize well–known benchmark functions.

| Symbol | Parameter | Range | Default | Constriction |
|---|---|---|---|---|
| $s$ | swarm size | $\mathbb{N}$ | 40 | 40 |
| $c_1$ | cognitive parameter | $\mathbb{R}_+$ | 2 | 1.4944 |
| $c_2$ | social parameter | $\mathbb{R}_+$ | 2 | 1.4944 |
| $w_{\max}$ | starting value of the inertia weight $w$ | $\mathbb{R}_+$ | 0.9 | 0.729 |
| $w_{\text{scale}}$ | final value of $w$ in percentage of $w_{\max}$ | $\mathbb{R}_+$ | 0.4 | 1.0 |
| $w_{\text{iterScale}}$ | percentage of iterations, for which $w_{\max}$ is reduced | $\mathbb{R}_+$ | 1.0 | 0.0 |
| $v_{\max}$ | maximum value of the step size (velocity) | $\mathbb{R}_+$ | 100 | 100 |

**Table 2** Default settings of the exogenous parameters of PSO with constriction factor. Recommendations from [29].

| Symbol | Parameter | Range | Default Values |
|---|---|---|---|
| $s$ | swarm size | $\mathbb{N}$ | 40 |
| $\chi$ | constriction coefficient | $\mathbb{R}_+$ | 0.729 |
| $\varphi$ | multiplier for random numbers | $\mathbb{R}_+$ | 4.1 |
| $v_{\max}$ | maximum value of the step size (velocity) | $\mathbb{R}_+$ | 100 |

where $\chi$ is the *constriction factor* [30], and it is derived analytically through the formula [29],

$$\chi = \frac{2\kappa}{|2-\varphi-\sqrt{\varphi^2-4\varphi}|}, \tag{4}$$

for $\varphi > 4$, where $\varphi = c_1 + c_2$, and $\kappa = 1$. Different configurations of $\chi$, as well as a thorough theoretical analysis of the derivation of Eq. (4), can be found in [29]. Equations (1) and (3) are algebraically equivalent. In our experiments, the so–called "canonical PSO" variant proposed in [30], which is the constriction variant defined by Eq. (3) with $c_1 = c_2$, has been used. The corresponding parameter setting for the constriction factor variant of PSO is reported in the last column (denoted as "Constriction") of Table 1, where $\chi$ is reported in terms of its equivalent inertia weight notation, for uniformity reason.

PSO is a stochastic algorithm, and, therefore, random number generators are used. An optimization practitioner is interested in robust solutions, i.e., solutions independent from the seeds of the random number generators. The proposed statistical methodology provides guidelines to design robust PSO algorithms under restrictions, such as a limited number of function evaluations and processing units. These restrictions can be modeled by considering the performance of the algorithm in terms of the (expected) best function value for a limited number of fitness function evaluations. A discussion on problems from different classes, including real–world optimization problems, is provided in [25].

### 2.2 The Nelder–Mead Simplex Algorithm

The Nelder–Mead Simplex (NMS) algorithm was developed by Nelder and Mead in 1965 [5]. It was motivated by the observation that $(n+1)$ points are adequate to identify a downhill direction in an $n$–dimensional landscape. However, $(n+1)$ points define also a non-degenerate simplex in $\mathbb{R}^n$. Thus, it seemed a good idea to exploit a simplex for probing the search space, using only function values [31].

Nelder and Mead incorporated a set of moves that enhance the algorithm's performance, namely *reflection*, *expansion*, *contraction*, and *shrinkage*. A new point is generated at each iteration. Its function value is compared to the function values of the vertices of the simplex. One of the vertices is replaced by the new point. Reflection reflects a vertex of the simplex through the centroid of the opposite face. Expansion allows the algorithm to take a longer step from the reflection point (centroid) towards the reflected vertex, while contraction halves the length of the step, thereby resulting in a more conservative search. Finally, shrinkage reduces the length of all edges that are adjacent to the best vertex, i.e., the vertex with the smallest function value. Thus, there are 4 parameters to be specified, namely the coefficients of reflection, $\rho$, expansion, $\chi$, contraction, $\gamma$, and shrinkage, $\sigma$. Default settings of these parameters are reported later in Table 8. NMS is considered a quite robust but relatively slow algorithm [32]. However, it works reasonably well for non–differentiable functions.

## 3 Computational Statistics

The term "computational statistics" subsumes computationally intensive methods [33]. Statistical methods, such as experimental design techniques and regression analysis can be used to analyze the experimental setting of algorithms on specific test problems. One important goal in the analysis of search algorithms is to find variables that have a significant influence on the algorithm's performance, which can be quantitatively defined as the average obtained function value in a number (e.g. 50) of independent experiments. This measure was also used in [26]. Questions like "how does a variation of the swarm size influence the algorithm's performance?" or "are there any interactions between swarm size and and the value of the inertia weight?" are important research questions that provide an understanding of the fundamental principles of stochastic search algorithms such as PSO.

The approach presented here combines classical techniques from statistical *design of experiments* (DOE), *classification and regression trees* (CART), and modern *design and analysis of computer experiments* (DACE) techniques. Thorough descriptions of these methodologies can be found in [34–36]. In [37], a comparison of DOE, CART and DACE for direct search algorithms, is provided.

### 3.1 Design and Analysis of Computer Experiments (DACE)

Let $a_d$ denote a set of vectors with specific settings of an algorithm. This is called the *algorithm design*. In the case of PSO, $a_d$ can include parameters such as swarm size, the values of $c_1$ and $c_2$, etc. Note that a design can consist of one vector only. The *optimal design* is denoted as $a_d^*$. The *problem design*, $p_d$, provides information related to the optimization problem, such as the available resources (number of function evaluations), the problem's dimension, etc. The run of a stochastic search algorithm can be treated as an experiment with a stochastic output, $Y(a_d, p_d)$. One of our goals is to find an algorithm design, $a_d^*$, that optimizes the output (function value). Since DACE was introduced for deterministic computer experiments, repeated runs are necessary to apply this technique to stochastic search algorithms.

In the following, the specification of the DACE process model that will be used later to analyze our experiments is described. This specification is similar to the selection of a linear or quadratic regression model in classical regression.

DACE provides methods to predict unknown values of a stochastic process and it can be applied to interpolate observations from computationally expensive simulations. In the DACE stochastic process model, a deterministic function is evaluated at $m$ design points. The stochastic process model proposed in [38] expresses the deterministic response, $y(x)$, for a $d$–dimensional input, $x$, as a realization of a regression model, $\mathcal{F}$, and a stochastic process, $Z$,

$$Y(x) = \mathcal{F}(\beta, x) + Z(x). \tag{5}$$

The model used in classical regression, $Y = \beta X + \varepsilon$, is a special case of Eq. (5). The stochastic process $Z(\cdot)$ is assumed to have zero mean and covariance equal to $V(\omega, x) = \sigma^2 \mathcal{R}(\theta, \omega, x)$, between $Z(\omega)$ and $Z(x)$, with process variance $\sigma^2$ and correlation model $\mathcal{R}(\theta, \omega, x)$. Following the analysis in [38], we use $\rho$ functions, $f_j : \mathbb{R}^d \to \mathbb{R}$, $j = 1, \ldots, \rho$, to define the regression model,

$$\mathcal{F}(\beta, x) = \sum_{j=1}^{\rho} \beta_j f_j(x) = \langle f(x), \beta \rangle,$$

where, $f(x) = (f_1(x), \ldots, f_\rho(x))^\top$, $\beta = (\beta_1, \ldots, \beta_\rho)^\top$, and $\langle \cdot, \cdot \rangle$ denotes the inner product of the vectors. DACE also provides an estimation of the prediction error of an untried point, or the mean squared error, $MSE(x)$, of the predictor. Correlations of the form,

$$\mathcal{R}(\theta, \omega, x) = \prod_{j=1}^{d} \mathcal{R}_j(\theta, \omega_j - x_j),$$

are considered in our experiments. The correlation function should be chosen with respect to the underlying process [39]. In [40], seven different models are discussed. The Gaussian correlation function,

$$\mathcal{R}_j(\theta, h_j) = \exp\left(-\theta_j h_j^2\right), \tag{6}$$

**Table 3** Sequential approach. This approach combines methods from computational statistics and exploratory data analysis to improve (tune) the performance of direct search algorithms.

| Step | Action |
|---|---|
| (S-1) | Pre–experimental planning |
| (S-2) | Scientific hypothesis |
| (S-3) | Statistical hypothesis |
| (S-4) | Specification |
| | (a) optimization problem, |
| | (b) constraints, |
| | (c) initialization method, |
| | (d) termination method, |
| | (e) algorithm (important factors), |
| | (f) initial experimental design, |
| | (g) performance measure |
| (S-5) | Experimentation |
| (S-6) | Statistical modeling of data and prediction |
| (S-7) | Evaluation and visualization |
| (S-8) | Optimization |
| (S-9) | Termination: If the obtained solution is good enough, or the maximum number of iterations has been reached, go to step (S-11) |
| (S-10) | Design update and go to step (S-5) |
| (S-11) | Rejection/acceptance of the statistical hypothesis |
| (S-12) | Objective interpretation of the results from step (S-11) |

has been used in our experiments, with $h_j = \omega_j - x_j$, and $\theta_j > 0$.

### 3.2 Sequential Designs Based on DACE

Prior to the execution of experiments with an algorithm, the experimenter has to specify suitable parameter settings for the algorithm, i.e., an algorithm design, $a_d$. Often, designs that use sequential sampling are more efficient than designs with fixed sample sizes. First, an initial design, $a_d^{(0)}$, is specified. Information obtained in the first runs can be used for the determination of the second design, $a_d^{(1)}$, in order to chose new design points more efficiently.

Sequential sampling approaches with adaptation have been proposed for DACE. For example, in [38], sequential sampling approaches with and without adaptation were classified to the existing meta–model. We will present a sequential approach that is based on the expected improvement. In [36, p. 178], a heuristic algorithm for unconstrained global minimization problems is presented, and, if $y_{\min}^k$ denotes the smallest known minimum value after $k$ runs of the algorithm, $y(x)$ is the algorithm's response (realization of $Y(x)$ in Eq. (5)), and $x \in a_d$, i.e., $x$ represents a specific design point from the algorithm design, $a_d$, then, the improvement is defined as,

$$\text{Improvement at } x = \begin{cases} y_{\min}^k - y(x), & y_{\min}^k - y(x) > 0, \\ 0, & y_{\min}^k - y(x) \leqslant 0. \end{cases} \tag{7}$$

The discussion in [36, p. 178] leads to the conclusion that new design points are attractive "if either there is a high probability that their predicted output is below the current observed minimum and/or there is a large uncertainty in the predicted output". This result comes in line with the experimenters' intention to avoid sites that guarantee worse results.

The sequential approach consists of the twelve steps that are reported in Table 3. During the pre–experimental planning phase (S-1) the experimenter defines exactly what is to be studied and how the data are to be collected. The recognition and statement of the problem seems to be a rather obvious task. However, in practice, it is not simple to formulate a generally accepted goal. *Discovery*, *confirmation*, and *robustness* are only three possible scientific goals of an experiment. Discovery asks what happens if new operators are implemented. Confirmation

analyzes how the algorithm behaves on different problems, and robustness asks for conditions that decrease the algorithm's performance.

Statistical methods like run–length distributions (RLD) provide suitable means to measure the performance and describe the qualitative behavior of optimization algorithms. A plot of an RLD is depicted for our experiments in Fig. 6. The algorithm to be analyzed is run $k$ times, with different random number generator seeds, on a given problem instance. The maximum number of function evaluations, $t_{\max}$, is set to a relatively high value. For each successful run, the number of required function evaluations, $t_{\text{run}}$, is recorded. If the run fails, $t_{\text{run}}$ is set to infinity. The *empirical cumulative distribution function* (CDF) is the cumulative distribution function that represents these results. Let $t_{\text{run}}(j)$ be the run–length for the $j$–th successful run. Then, the empirical CDF is defined as [41],

$$\Pr\left(t_{\text{run}}(j) \leqslant t\right) = \frac{\{\#j \,|\, t_{\text{run}}(j) \leqslant t\}}{k}, \tag{8}$$

where $\{\#j \,|\, t_{\text{run}}(j) \leqslant t\}$ denotes the number of indices $j$, such that $t_{\text{run}}(j) \leqslant t$. In step (S-2), the experimental goal should be formulated as a scientific hypothesis, e.g. "does an employed scheme $A$ improve the algorithm's performance?".

A statistical hypothesis, such as "there is no difference in means comparing the performance of the two competing schemes", is formulated in the step (S-3) that follows. Step (S-4) requires at least the specification of,

(a) an optimization problem,

(b) constraints (for example the maximum number of function evaluations),

(c) an initialization method,

(d) a termination method,

(e) an algorithm, and the specification of its important factors,

(f) an initial experimental design, and,

(g) a measure to judge the performance.

Regarding (c), several methods have been used for the initialization of the population in population–based algorithms, or the determination of an initial point, $x^{(0)}$, in algorithms that use a single search point. More specifically, individuals that comprise a population can be initialized,

(I-1) deterministically with equal starting points, i.e., the same point, $x^{(0)}$, is assigned to every individual. The starting point is specified by the experimenter,

(I-2) deterministically, but with different starting points, $x^{(0)}$, for each repeat run. The corresponding points $x^{(0)}$ are selected from an $n$–dimensional interval, $[x_l, x_u]^n$, where $x_l$ and $x_u$ specify lower and the upper bounds, respectively.

Stochastic variants of (I-1) and (I-2) are denoted as (I-3) and (I-4), respectively. Nowadays, (I-4) is commonly used in evolutionary computation, whereas (I-1) is widely used for global optimization. For example, an asymmetric initialization scheme was used in [26], where the initial positions of the particles, $x_i(0)$, $i = 1, \ldots, s$, were chosen uniformly distributed in the range $[15, 30]^n$. Initialization method (I-2) was proposed in [42].

An algorithm terminates if,

(1) the problem was solved. Domain convergence (T-1), and function value convergence (T-2) can be distinguished,

(2) the algorithm has stalled (T-3),

(3) the resources, e.g. the maximum number of function evaluations, $t_{\max}$, are exhausted (T-4).

The corresponding problem design, $p_d$, that summarizes the information from (a) to (d) for our experiments with PSO is reported in Table 4 (a description of the S–ring model is postponed until the presentation of the experimental results), while the algorithm design, $a_d$, is reported in Table 5 and summarizes steps (e) and (f). An experimental design, $e_d$, consists of both a problem and an algorithm design. The experimental goal of the sequential approach presented here can be characterized as the determination of an optimal (or improved)

**Table 4** Problem design, $p_d$, for the PSO runs. The experiment's number, the number of runs, $k$, the maximum number of function evaluations, $t_{\max}$, the problem's dimension, $n$, the initialization method, the termination criterion, the lower, $x_l$, and upper, $x_u$, bounds for the initialization, as well as the optimization problem are reported.

| Experiment | $N$ | $t_{\max}$ | $n$ | Init. | Term. | $x_l$ | $x_u$ | Problem |
|---|---|---|---|---|---|---|---|---|
| 00001 | 50 | 2500 | 10 | (I-4) | (T-4) | 15 | 30 | Rosenbrock (F2) |
| 00002 | 50 | 1000 | 12 | (I-3) | (T-4) | $-10$ | 10 | S–ring |

**Table 5** Algorithm design, $a_d$, for the inertia weight PSO variant that corresponds to the experiment 00001 of Table 4, which optimizes the 10–dimensional Rosenbrock function. $a_d^{(l)}$ and $a_d^{(u)}$ denote the lower and upper bounds to generate the LHD, respectively, and $a_d^*$ denotes the parameter settings of the improved design that was found by the sequential approach.

| | $s$ | $c_1$ | $c_2$ | $w_{\max}$ | $w_{\text{scale}}$ | $w_{\text{iterScale}}$ | $v_{\max}$ |
|---|---|---|---|---|---|---|---|
| $a_d^{(l)}$ | 5 | 1.0 | 1.0 | 0.7 | 0.2 | 0.5 | 10 |
| $a_d^{(u)}$ | 100 | 2.5 | 2.5 | 0.99 | 0.5 | 1 | 750 |
| $a_d^*$ | 21 | 2.25413 | 1.74587 | 0.788797 | 0.282645 | 0.937293 | 11.0496 |

algorithm design, $a_d^*$, for a given problem design, $p_d$. Measures to judge the performance are presented in the next sections (Table 7).

The constriction factor variant of PSO requires the determination of four exogenous strategy parameters, namely the swarm size, $s$, constriction factor, $\chi$, parameter $\varphi = c_1 + c_2$, and the maximum velocity, $v_{\max}$. At each stage, *Latin Hypercube Designs* (LHD) are used. In [43] it is reported that experience with the stochastic process model had indicated that 10 times the expected number of important factors is often adequate number of runs for the initial LHD. Thus, an LHD with at least $m = 15$ design points was chosen. This is the minimum number of design points to fit a DACE model that consists of a second order polynomial regression model and a Gaussian correlation function. The former requires $1 + \sum_{i=1}^{4} i = 11$ design points, while the latter requires 4 design points. Note that for $m = 15$ there are no degrees of freedom left to estimate the mean squared error of the predictor [36].

After that, the experiment is run (S-5). Preliminary (pilot) runs can give a rough estimate of the experimental error, run times, and the consistency of the experimental design. Again, RLDs can be very useful. Since we consider probabilistic search algorithms in our investigation, functions may be evaluated several times, as discussed in [36]. The experimental results provide the base for modeling and prediction in step (S-6). The model is fitted and a predictor is obtained for each response.

The model is evaluated in step (S-7). Several visualization techniques can be applied. Simple graphical methods from exploratory data analysis are often helpful. Histograms and scatter plots can be used to detect outliers. If the initial ranges for the designs were chosen improperly (e.g., very wide initial ranges), visualization of the predictor can guide the choice of more suitable (narrower) ranges in the next stage. Several techniques to assess the validity of the model have been proposed. Cross–validation predictions versus actual values, as well as standardized cross–validation residuals versus cross–validation predictions can be plotted. Sensitivity analysis can be used to ascertain how much a statistical model depends on its factors. In [44–47], variance–based methods that are used in sensitivity analysis are analyzed, while, in [36, pp. 193] sensitivity analyses based on ANOVA–type decompositions are described. The computation of sensitivity indices can be performed by decomposing the response into an average, main effects for each input, two–input interactions and higher–order interactions [38, pp. 417]. Additional graphical methods can be used to visualize the effects of factors and their interactions on the predictors. The 3–dimensional visualizations depicted in Fig. 2, produced with the DACE toolbox [48], have proved to be very useful. The predicted values can be plotted to support the numerical analysis, and the MSE of prediction is used to asses the accuracy of the prediction. We explicitly note here, that statistical models can provide only guidelines for further experiments, and they do not prove that a factor has a particular effect.

If the predicted values are not accurate, the experimental setup has to be reconsidered. This comprehends especially the specification of the scientific goal and the ranges of the design variables. Otherwise, new design

points in promising subregions of the search space can be determined (S-8) if further experiments are necessary. Thus, a termination criterion has to be tested (S-9). If it is not fulfilled, based on the expected improvement defined by Eq. (7), new candidate design points can be generated (S-10). A new design point is selected if there is a high probability that the predicted output is below the current observed minimum and/or there is a large uncertainty in the predicted output. Otherwise, if the termination criterion is true, and the obtained solution is good enough, the final statistical evaluation (S-11) that summarizes the results is performed. A comparison between the first and the improved configuration should be performed. Techniques from exploratory data analysis can complement the analysis at this stage. Besides an investigation of the numerical values, such as mean, median, minimum, maximum, and standard deviation, graphical presentations such as boxplots, histograms, and RLDs can be used to support the final statistical decision (e.g. see Fig. 4).

Finally, we have to decide whether the result is scientifically important (S-12), since the difference, although statistically significant, can be scientifically meaningless. Here comes the experimenter's skill into operation. The experimental setup should be reconsidered at this stage and questions like "have suitable test functions or performance measures been chosen?" or "did floor or ceiling effects occur?" must be answered. Test problems that are too easy may cause such ceiling effects. If two algorithms, $A$ and $B$, achieve their maximum level of performance (or close to it), then the hypothesis "performance of $A$ is better than performance of $B$" should not be confirmed [49]. Floor effects describe the same phenomenon on the opposite site of the performance scale, i.e., the test problem is so hard that nearly no algorithm can solve it correctly. Such effects can occur when the number of function evaluations is chosen very small. For example, in the Rosenbrock function, a starting point, $x_i^{(0)} = (10^6, 10^6)^\top$, and a budget of only 10 function evaluations can cause such effects. Performance profiles can help the experimenter to decide whether ceiling effects have occurred.

## 4 Experimental Results

Initially, we investigated a simple and well–known test function [26] to gain an intuition regarding the workings of the proposed technique. In the next step of our analysis, the S–ring model was considered. We provide a demonstration of the sequential approach by conducting a brief investigation for the Rosenbrock function, using the two variants of PSO as well as the NMS algorithm. Experimental results of evolutionary algorithms presented in empirical studies are sometimes based on a huge number of fitness function evaluations (larger than $10^5$), even for simple test functions. Our goal is to demonstrate how statistical design methods, e.g. DACE, can reduce this number significantly. The proposed approach is thoroughly analyzed for the case of the inertia weight variant of PSO.

### 4.1 Optimizing the inertia weight variant of PSO

This example describes in detail how to tune the exogenous parameters of PSO, and it extends the approach presented in [28]. Experimental results presented in [26] have been chosen as a starting point for our analysis.

(S-1) *Pre–experimental planning:* Pre–experimental tests to explore the optimization potential supported the assumption that tuning might improve the algorithm's performance. RLD revealed that there exists a configuration that was able to complete the run successfully using less than 8000 iteration, for nearly 80% of the cases. This was less than half the number of function evaluations used in the reference study, justifying the usefulness of the analysis.

(S-2) *Scientific hypothesis:* There exists a parameterization, $a_d^*$, of PSO that improves its efficiency significantly.

(S-3) *Statistical hypothesis:* PSO with the parameterization $a_d^*$ outperforms PSO with the default parameterization, $a_d^{(0)}$, which is used in [26].

(S-4) *Specification:* Results from the Sphere function that was investigated in [26], were not very meaningful. Therefore, the generalized Rosenbrock function was chosen for our comparisons. This test problem is, generally, defined as,

$$f(x) = \sum_{i=1}^{n-1} \left(100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2\right), \tag{9}$$

**Table 6** Inertia weight PSO optimizing the 10–dimensional Rosenbrock function. Each row represents the best algorithm design at the corresponding tuning stage. Note, that function values (reported in the first column) can worsen (increase) although the design is improved. This happens due to the noise in the results, $y$. The probability that a seemingly good function value that is in fact worse, might occur, decreases during the sequential procedure, because the number of re–evaluations is increased. The number of repeats is doubled if a configuration performs best twice. The corresponding configurations are marked with an asterisk.

| $y$ | $s$ | $c_1$ | $c_2$ | $w_{\max}$ | $w_{\text{Scale}}$ | $w_{\text{IterScale}}$ | $v_{\max}$ | Config |
|---|---|---|---|---|---|---|---|---|
| 6.61557 | 26 | 1.45747 | 1.98825 | 0.712714 | 0.481718 | 0.683856 | 477.874 | 14 |
| 18.0596 | 39 | 1.30243 | 1.84294 | 0.871251 | 0.273433 | 0.830638 | 289.922 | 19 |
| 71.4024 | 26 | 1.45747 | 1.98825 | 0.712714 | 0.481718 | 0.683856 | 477.874 | 14* |
| 78.0477 | 30 | 2.21960 | 1.26311 | 0.944276 | 0.289710 | 0.893788 | 237.343 | 3 |
| 75.6154 | 30 | 2.21960 | 1.26311 | 0.944276 | 0.289710 | 0.893788 | 237.343 | 3* |
| 91.0935 | 18 | 1.84229 | 1.69903 | 0.958500 | 0.256979 | 0.849372 | 95.1392 | 35 |
| 91.5438 | 21 | 1.05527 | 1.25077 | 0.937259 | 0.498268 | 0.592607 | 681.092 | 43 |
| 93.7541 | 11 | 1.58098 | 2.41902 | 0.728502 | 0.469607 | 0.545451 | 98.9274 | 52 |
| 93.9967 | 93 | 1.71206 | 1.02081 | 0.966302 | 0.378612 | 0.972556 | 11.7651 | 20 |
| 99.4085 | 39 | 1.30243 | 1.84294 | 0.871251 | 0.273433 | 0.830638 | 289.922 | 19* |
| 117.595 | 11 | 1.13995 | 2.31611 | 0.785223 | 0.236658 | 0.962161 | 56.9096 | 57 |
| 146.047 | 12 | 1.51468 | 2.48532 | 0.876156 | 0.392995 | 0.991074 | 261.561 | 1 |
| 147.410 | 22 | 1.72657 | 2.27343 | 0.710925 | 0.235521 | 0.574491 | 50.5121 | 54 |
| 98.3663 | 22 | 1.72657 | 2.27343 | 0.710925 | 0.235521 | 0.574491 | 50.5121 | 54* |
| 41.3997 | 21 | 2.25413 | 1.74587 | 0.788797 | 0.282645 | 0.937293 | 11.0496 | 67* |
| 43.2249 | 21 | 2.25413 | 1.74587 | 0.788797 | 0.282645 | 0.937293 | 11.0496 | 67* |
| 53.3545 | 21 | 2.25413 | 1.74587 | 0.788797 | 0.282645 | 0.937293 | 11.0496 | 67* |

where $n$ is the problem's dimension. Its global minimizer is $x^* = (1,1)^\top$, with function value $f^* = 0$, and $x^{(0)} = (-1.2,1)^\top$ is considered a good starting point [50]. We considered the 10–dimensional case the function.

Following the experimental design in [26], we recorded the mean fitness value of the best particle of the swarm at each one of the 50 runs. This value is denoted as $\tilde{f}^{(50)}$. For the generation of RLD plots, a threshold value to distinguish successful from unsuccessful runs was specified. A run configuration was classified as successful, if $\tilde{f}^{(50)} < \tilde{f}^{(\text{Shi})}$, where $\tilde{f}^{(\text{Shi})} = 96.1715$ is the value reported in [26]. The initial problem design, which consists of the objective function, its dimension, the maximum number of function evaluations, the initialization method and the termination criterion was reported in Table 4. The corresponding setting for the algorithm design is reported in Table 5. An algorithm design that covers a wide range of interesting parameter settings (region of interest) was chosen, and no problem–specific knowledge for the Rosenbrock function was used to perform the experiments, expecting that the sequential approach will guide the search into successful regions.

(S-5) *Experimentation:* Table 6 presents the optimization process. Each line in Table 6 corresponds to one optimization step in the sequential approach. At each step, two new designs are generated and the best is re–evaluated. This is similar to the selection procedure in $(1+2)$–Evolution Strategies [51]. The number of repeat runs, $k$, of the algorithm designs is increased (doubled), if a design has performed best twice or more. A starting value of $k = 2$ was chosen. For example, design 14 performs best at iteration 1 and iteration 3. It has been evaluated 4 times, therefore the number of evaluations is set to 4 for every newly generated design. This provides a fair comparison and reduces the risk of incorrectly selecting a worse design.

(S-6) *Statistical modeling and prediction:* Following [36], the response is modeled as a realization of a regression model and a random process as described in Eq. (5). A Gaussian correlation function as defined in

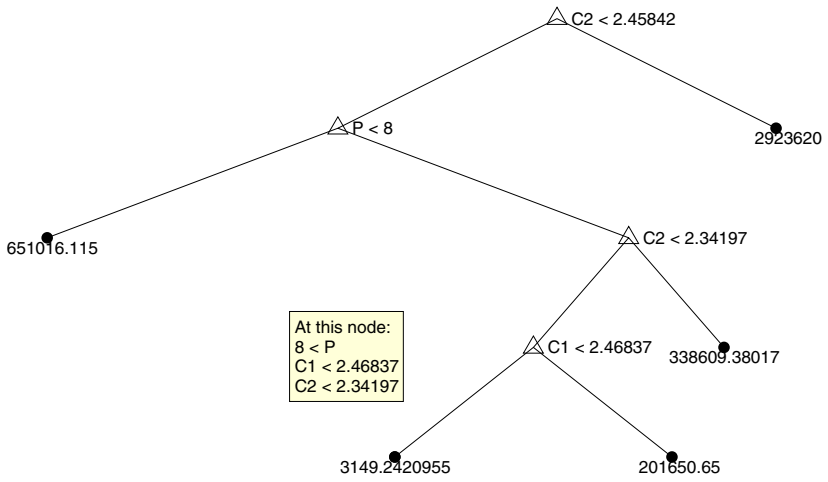


**Fig. 1** Regression tree. Values at the nodes show the average function values for the associated node. The value in the root node is the overall mean. The left son of each node contains the configurations that fulfill the condition in the node. $c_1 + c_2 > 4$ produce outliers that complicate the analysis. In addition, this analysis shows that the swarm size, $s$, should be larger than 8.

Eq. (6), and a regression model with polynomial of order 2, have been used. Hence, the model reads,

$$Y(x) = \sum_{j=1}^{\rho} \beta_j f_j(x) + Z(x), \tag{10}$$

where $Z(\cdot)$ is a random process with mean zero and covariance $V(\omega, x) = \sigma^2 \mathcal{R}(\theta, \omega, x)$, and the correlation function was chosen as,

$$\mathcal{R}(\theta, \omega, z) \prod_{j=1}^{n} \exp\left(-\theta_j (\omega_j - x_j)^2\right). \tag{11}$$

Additionally, at certain stages, a tree–based regression model, as shown in Fig. 1, was constructed to determine parameter settings that produce outliers.

(S-7) *Evaluation, visualization:* The MSE and the predicted values can be plotted to support the numerical analysis (we produced all 3–dimensional visualizations with the DACE toolbox [48]). For example, the interaction between $c_1$ and $c_2$ is shown in Fig. 2. Values of $c_1$ and $c_2$ with $c_1 + c_2 > 4$ generated outliers that might disturb the analysis. To alleviate these outliers, a design correction method has been implemented, namely $c_1 = c_2 - 4$, if, $c_1 + c_2 > 4$. The right part of Fig. 2 illustrates the estimated MSE. Since no design point has been placed in the ranges $1 < c_1 < 1.25$ and $2.25 < c_2 < 2.5$, the MSE is relatively high. This might be an interesting region where a new design point will be placed during the next iteration. Figure 3 depicts the same situation as Fig. 2 after the determination of the design correction. In this case, a high MSE is associated with the region $c_1 + c_2 > 4$, but no design point will be placed there.

(S-8) *Optimization:* Termination or design update. Based on the expected improvement defined in Eq. (7), two new design points, $a_d^{(1)}$, and, $a_d^{(2)}$, have been generated. These two designs will be evaluated and their performances will be compared to the performance of the current best design. The best design found so far is re–evaluated. The iteration terminates, if a design was evaluated for $k = 50$ times and the solution is obtained. The values in the final model read, $s = 21$, $c_1 = 2.25413$, $c_2 = 1.74587$, $w_{\max} = 0.788797$, $w_{\text{scale}} = 0.282645$, $w_{\text{iterScale}} = 0.937293$, and $v_{\max} = 11.0496$. This result is shown in the last row of Table 6.

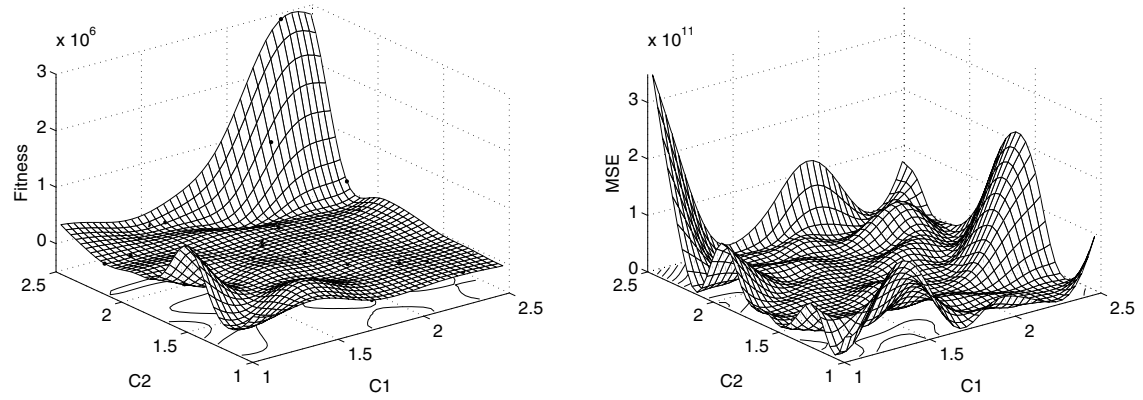


**Fig. 2** Predicted values (left) and MSE (right). As can be seen in the left figure, $c_1 + c_2 > 4$ produce outliers that complicate the analysis. The plots present the same data as the regression tree in Fig. 1.

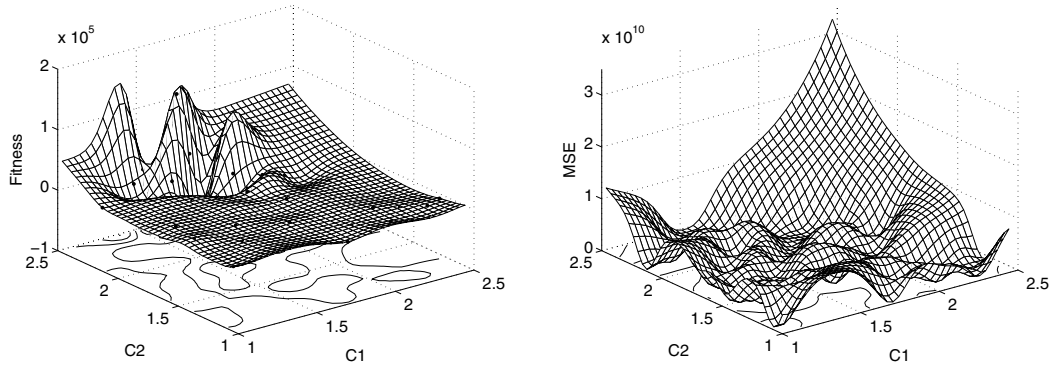


**Fig. 3** Predicted values (left) and MSE (right). The design correction avoids settings with $c_1 + c_2 > 4$ that produce outliers (left). Therefore, a high mean squared error exists in the excluded region (right).

(S-11) *Rejection/acceptance:* Finally, we compare the configuration from [26] to the optimized configuration. The final (tuned) and the first configurations are repeated $50$ times. Histograms and boxplots are illustrated in Fig. 4 for both variants of PSO. The tuned design of the inertia weight PSO variant clearly improves its performance of the PSO algorithm. Statistical analysis is reported in Table 7. Performing a classical t–test indicates that the null hypothesis "there is no difference in the mean performances of the two algorithms" can be rejected at the $5\%$ level.

(S-12) *Objective interpretation:* The statistical results from (S-11) give good reasons for the assumption that PSO with the tuned design performs better (on average) than the default design. Comparing the parameters of the improved design, $a_d^*$, reported in Table 5, with the default settings of PSO, no significant differences can be observed. Only $v_{\max}$, the maximum value of the velocity, appears to be relatively small. A swarm size of $20$ particles appears to be a good value for this problem instance.

The analysis and the tuning procedure described so far have been based solely on the average function value in $50$ runs. This value may be irrelevant in a different optimization context. For example, the best function value (minimum) or the median can be alternatively used. A similar optimization procedure could have been performed for any of these cases with the presented sequential approach. However, the optimal design presented in this study is only applicable for this specific optimization task (or experimental design). As in [26], the starting points have been initialized randomly in the range $[15, 30]^n$.

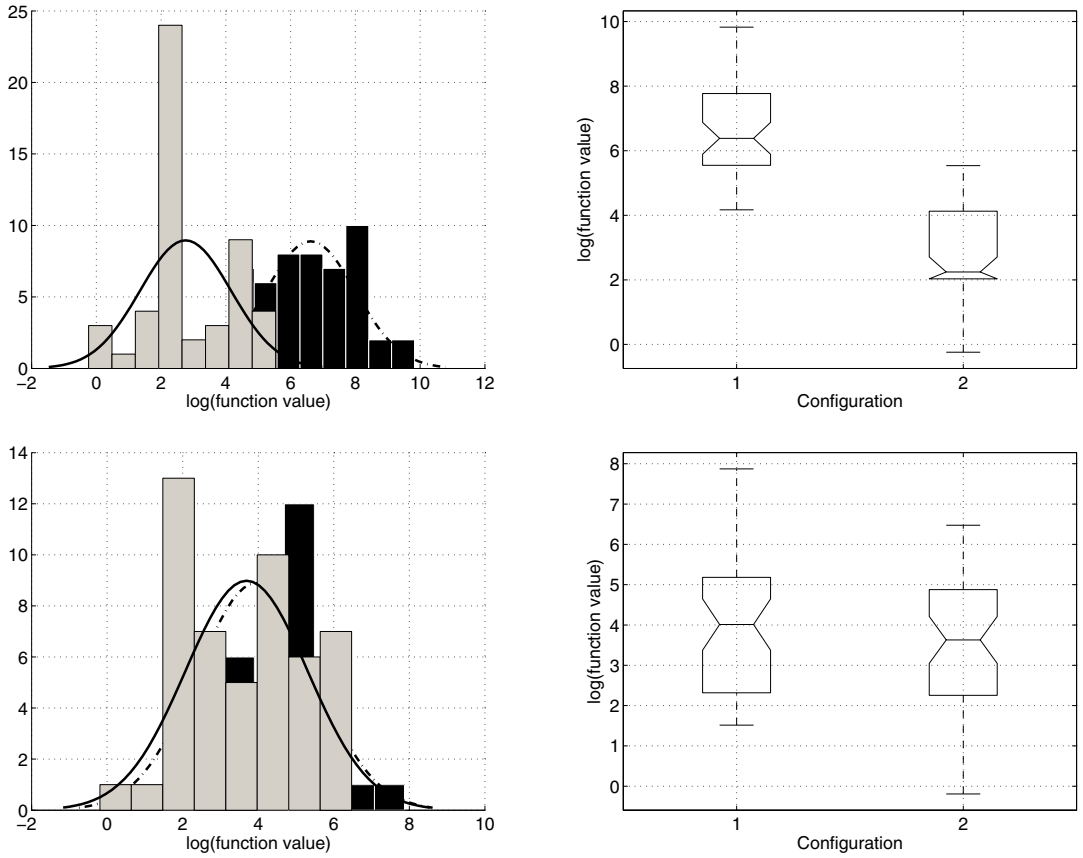


**Fig. 4** Histogram and box plot. Left: Solid lines and light bars represent the improved design. Right: The default configuration is denoted as 1, whereas 2 denotes the improved variant. Top: Both plots indicate that the tuned inertia weight PSO version performs better than the default version. Bottom: No clear difference can be detected when comparing the default with the improved constriction factor PSO variant.

**Table 7** Result table for the Rosenbrock function. Default designs from [26, 29, 32], as well as the improved design for all algorithms, for $k = 50$ runs, are reported.

| Design | Algorithm | Mean | Median | StD | Min | Max |
|---|---|---|---|---|---|---|
| $a_d^{(\text{Shi})}$ | PSO (iner.) | $1.8383 \times 10^3$ | 592.1260 | $3.0965 \times 10^3$ | 64.6365 | 18519 |
| $a_d^*$ | PSO (iner.) | 39.7029 | 9.4443 | 55.3830 | 0.7866 | 254.1910 |
| $a_d^{(\text{Clerc})}$ | PSO (con.) | 162.02 | 58.51 | 378.08 | 4.55 | $2.62 \times 10^3$ |
| $a_d^*$ | PSO (con.) | 116.91 | 37.65 | 165.90 | 0.83 | 647.91 |
| $a_d^{(\text{Lagarias})}$ | NMS | $9.0711 \times 10^3$ | $1.1419 \times 10^3$ | $2.5040 \times 10^4$ | 153.0480 | 154966 |
| $a_d^*$ | NMS | 112.92 | 109.257 | 22.1279 | 79.7893 | 173.04 |
| | Quasi–Newton | $5.4550 \times 10^{-11}$ | $5.7906 \times 10^{-11}$ | $8.6231 \times 10^{-12}$ | $1.6157 \times 10^{-11}$ | $6.2028 \times 10^{-11}$ |

Hence, different sources of randomness are mixed in this example. The following studies will be based on deterministically generated starting points, as recommended in [42].

**Table 8** Default settings of the exogenous parameters of the NMS algorithm. This design is used in the MATLAB optimization toolbox [32].

| Factor | Symbol | Parameter | Range | Default Values |
|---|---|---|---|---|
| $z_1$ | $\rho$ | reflection | $\gamma > 0$ | 1.0 |
| $z_2$ | $\chi$ | expansion | $\chi > \max\{1, \rho\}$ | 2.0 |
| $z_3$ | $\gamma$ | contraction | $0 < \gamma < 1$ | 0.5 |
| $z_4$ | $\sigma$ | shrinkage | $0 < \sigma < 1$ | 0.5 |

**Table 9** PSO constriction factor: Algorithm design. The variables $s$, $\chi$, $\varphi$, and $v_{\max}$ have been defined in Table 2. $a_d^{(l)}$ and $a_d^{(u)}$ denote the ranges of the LHD, $a_d^*$ is the improved design and $a_d^{\text{Clerc}}$ is the design suggested in [29].

| | $s$ | $\chi$ | $\varphi$ | $v_{\max}$ |
|---|---|---|---|---|
| $a_d^{(l)}$ | 105 | 0.68 | 3.0 | 10 |
| $a_d^{(u)}$ | 100 | 0.8 | 4.5 | 750 |
| $a_d^*$ | 17 | 0.759 | 3.205 | 324.438 |
| $a_d^{\text{Clerc}}$ | 20 | 0.729 | 4.1 | 100 |

### 4.2 Optimizing the PSO constriction factor variant

The design of the PSO constriction factor variant was tuned in a similar manner as the inertia weight variant. The initial LHD is reported in Table 9, where $a_d^{(l)}$ and $a_d^{(u)}$ denote the lower and upper bounds of the region of interest, respectively, $a_d^*$ is the improved design that was found by the sequential procedure, and $a_d^{\text{Clerc}}$ is the default design recommended in [29]. As can be seen from the numerical results reported in Table 7, and the corresponding graphical representations (histograms and boxplots) in Fig. 4, there is no significant difference between the performance of the tuned $a_d^*$ and $a_d^{\text{Clerc}}$ [29].

### 4.3 Optimizing the Nonlinear Simplex Algorithm

In the NMS algorithm, $4$ parameters must be specified, namely the coefficients of reflection, $\rho$, expansion, $\chi$, contraction, $\gamma$, and shrinkage, $\sigma$. Default settings are reported in Table 8. Figure 5 depicts two interesting results from the tuning of the NMS algorithm. The value of the reflection parameter, $\rho$, should be smaller than $1.5$ (left). As can be seen in the right part of Fig. 5, there exists a relatively small local optimum regarding $\chi$ (expansion parameter) and $\psi$ (contraction parameter), respectively.

### 4.4 Synopsis

In addition to the optimization algorithms analyzed so far, the performance of a Quasi–Newton method was analyzed. An implementation from the commercial MATLAB optimization toolbox was used in the experiments. Quasi–Newton clearly outperformed the rest algorithms, as can be seen from the results in Table 7. A comparison of the RLDs of the three algorithms are reported shown in Fig. 6. The results support the claim that PSO performs better than the NMS algorithm. Only the tuned version of the latter was able to complete $20\%$ of the experiments with success. Regarding the two PSO variants, it is not obvious which one performs better. After the tuning process, the inertia weight variant appears to be better, but it requires the specification of seven (compared to only four in the constriction factor variant) exogenous parameters. However, the Rosenbrock function is mostly of academic interest, since it lacks many features of a real world optimization problem.

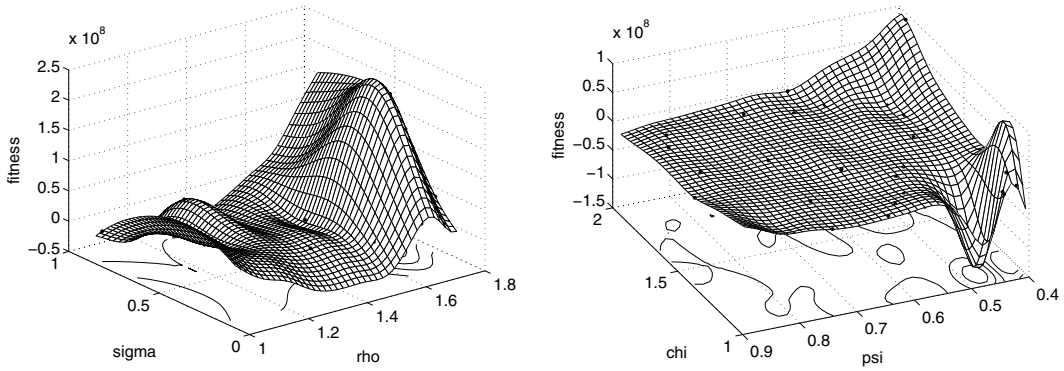


**Fig. 5** NMS simplex algorithm. Left: Predicted values. Function values (fitness) as a function of $\sigma$ and $\rho$. The figure suggests that the value of $\rho$ should be decreased. Right: Predicted values. A local optimum can be detected.

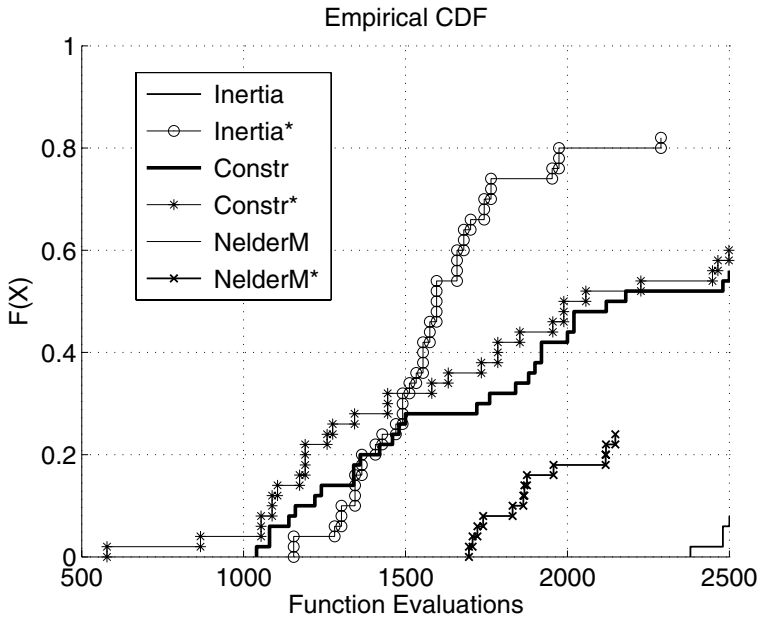


**Fig. 6** Run length distribution. Step (S-11), the final comparison of the canonical and the improved design based on RLDs. Asterisks denote improved configurations. The improved inertia weight version of PSO succeeded in more than 80% of the experiments with less than 2500 function evaluations. The standard NMS algorithm failed completely, but it was able with an improved design to succeed in 10% of the runs after 2500 function evaluations. For a given budget of 1500 function evaluations, both the constriction factor and the improved inertia weight PSO variants perform equally well.

## 5 Results for the Real World Optimization Problem

### 5.1 The Elevator Group Controller Problem

The construction of elevators for high–rise buildings is a challenging task. Today's urban life cannot be imagined without elevators. The elevator group controller is a central part of an elevator system. It assigns elevator cars to service calls in real–time, while optimizing the overall service quality, the traffic throughput, and/or the energy consumption. The Elevator Supervisory Group Control (ESGC) problem can be classified as a combinatorial optimization problem [52–54]. It exhibits the same complex behavior as many other stochastic traffic control

problems, such as materials handling systems with Automated Guided Vehicles (AGVs). Due to many difficulties in analysis, design, simulation, and control, the elevator optimization problem has been studied for a long time. First approaches were mainly based on analytical methods derived from queuing theory. However, today, Computational Intelligence (CI) methods and other heuristics are accepted as state of the art [1, 55].

The elevator group controller determines the floors where the elevator cars should go to. Since the group controller is responsible for the allocation of elevators to hall calls, a control strategy or *policy*, $\pi$, to perform this task in an optimal manner is required. One important goal in designing a better controller is the minimization of the time that passengers have to wait until they can enter an elevator car after having requested service. This time–span is called the *waiting time*. The so–called *service time* includes, additionally, the time that a passenger remains in the elevator car.

During a day, different traffic patterns can be observed. For example, in office buildings, an "up–peak" traffic is observed in the morning, when people start working, and, symmetrically, a "down-peak" traffic is observed in the evening. Most of the day there is "balanced" traffic with much lower intensity than at peak times. "Lunchtime" traffic consists of two (often overlapping) phases where people first leave the building for lunch or head for a restaurant floor, and then get back to work [56]. The ESGC problem subsumes the following problem,

> How to *assign elevators to passengers* in real–time, while optimizing different elevator configurations with respect to overall service quality, traffic throughput, energy consumption etc.

Fujitec, one of the world's leading elevator manufacturers, developed a controller that uses a Neural Network (NN) and a set of fuzzy controllers. The weights on the output layer of the NN can be modified and optimized, thereby resulting in a more efficient controller.

The associated optimization problem is quite complex. Empirical investigations have shown that the distribution of local optima in the search space is unstructured, and there are flat plateaus of equal function values. Furthermore, function values are stochastically disturbed and dynamically changing because they are influenced by the non–deterministic behavior of customers in the building. Experiments have shown that gradient–based optimization techniques cannot be applied successfully on such problems. Therefore, direct search algorithms have been chosen [57].

Elevator group controllers and the related policies are usually incomparable. To enable comparability for benchmark tests, a simplified elevator group control model, called the S–ring model, was developed. This model

(a) enables fast and reproducible simulations,

(b) is applicable to different buildings and traffic patterns,

(c) is scalable and extensible, and,

(d) can be used as a test problem generator.

The approach presented here uses both the actual Fujitec's simulator, which is depicted in Fig. 7 and has high accuracy but heavy computational cost, and the coarse (surrogate) S–ring model, which is depicted in Fig. 8 and is fast to solve, although less accurate. The proposed approach incorporates also Space Mapping (SM) techniques, which are used to iteratively update and optimize surrogate models [59], and the main goal is the computation of an improved solution with a minimal number of function evaluations.

Let $i$ denote a site in an elevator system. Then, a 2–bit state, $(s_i, c_i)$, is associated with it. The $s_i$–bit is set to 1 if a server is present on the $i$–th floor, or 0 otherwise. Correspondingly, the $c_i$–bit is set to 0 or 1 if there is no waiting passenger at site $i$, or at least one waiting passenger, respectively. Figure 8 depicts a typical S–ring configuration. The state of the system at time $t$ can be described by the vector,

$$x(t) = \Big(s_0(t), c_0(t), \ldots, s_{d-1}(t), c_{d-1}(t)\Big) \in \mathcal{B}^{2d}, \tag{12}$$

where $\mathcal{B} = \{0, 1\}$. The vector, $x(t) = (0, 1, 0, 0, \ldots, 0, 1, 0, 0)^\top$, represents the state of the system that is shown in Fig. 8, i.e., there is a customer waiting on the first floor ($c_0 = 1$), but no server present ($s_0 = 0$) etc. A state transition table is used to model the dynamics of the system. The state evolution is sequential (S–ring stands for "sequential ring"), scanning the sites from $d - 1$ down to 0, and then again around from $d - 1$. The up and down

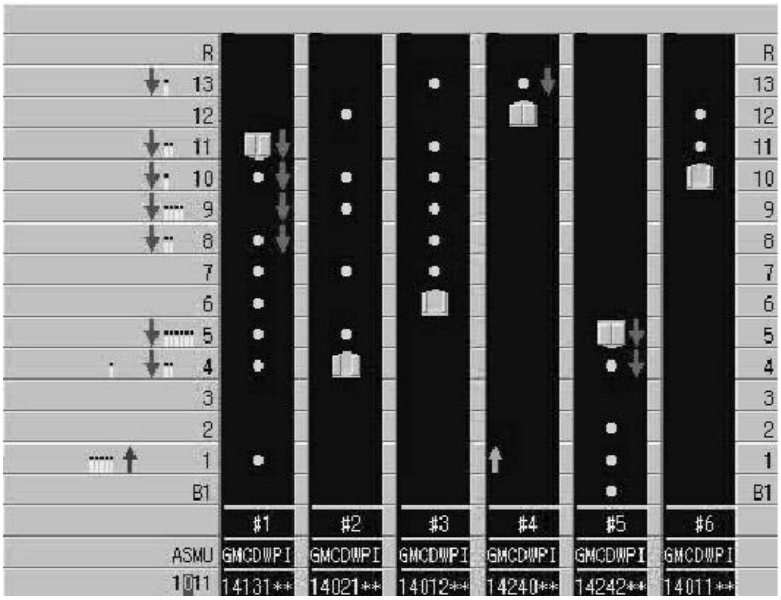


**Fig. 7** Visualization of the dynamics in an elevator system. Fujitec's elevator simulator representing the fine model. Six elevator cars are serving 15 floors. This model is computationally expensive and has a high accuracy [57].

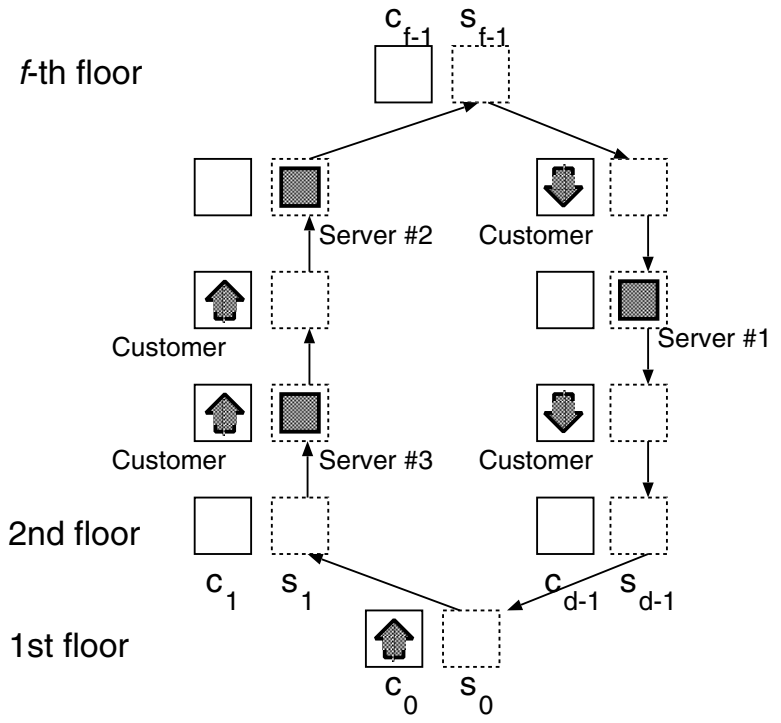


**Fig. 8** The S–ring as an elevator system. Three cars are serving 6 floors (or 10 sites). The sites are numbered from 0 to $d-1$, and $f$ denotes the floor number. There are two sites (up and down) for each floor. This is a coarse (surrogate) model that is fast to solve, but less accurate. Results obtained from this model should be transferable to other systems [58].

elevator movements can be considered as a loop. This motivates the ring structure. At each time step, one of the floor queues is considered, where passengers may arrive with a specific probability. Consider the situation at the third site (the up direction on the third floor) in Fig. 8. Since a customer is waiting, and a server is present, the controller has to make a decision. The elevator car can either serve the customer ("take" decision) or it can ignore the customer ("pass" decision). The former decision would change the values of the corresponding bits of $x(t)$, from $(1,1)$ to $(0,1)$, while the latter from $(1,1)$ to $(1,0)$. The rules of operation of this model are very simple, thus, it is easily reproducible and suitable for benchmark testing. Despite the model's simplicity, it is hard to find the optimal policy, $\pi^*$, even for a small S–ring. The true $\pi^*$ is not obvious, and its difference from heuristic suboptimal policies is non–trivial.

So far, the S–ring has been described as a simulation model. To use it as an optimization problem, it is equipped with an objective function. Consider the function that counts the sites with waiting customers at time $t$,

$$Q(t) = Q(x,t) = \sum_{i=0}^{d-1} c_i(t).$$

Then, the steady–state, time–average number of sites with waiting customers in queue is,

$$Q^* = \lim_{T\to\infty} \frac{\int_0^T Q(t)dt}{T} \quad \text{with probability 1.} \tag{13}$$

The basic optimal control problem is to find a policy, $\pi^*$, for a given S–ring configuration, such that the expected number, $Q^*$, of sites with waiting passengers that is the steady–state, time–average, as defined in Eq. (13), in the system is minimized, i.e.,

$$\pi^* = \arg\min_{\pi} Q(\pi). \tag{14}$$

A $2d$-dimensional vector, $y \in \mathbb{R}^{2d}$, can be used to represent the policy. Let, $\theta : \mathbb{R} \to \mathcal{B}$, define the Heaviside function,

$$\theta(z) = \begin{cases} 0, & z < 0 \\ 1, & z \geqslant 0, \end{cases} \tag{15}$$

$x = x(t)$ be the state at time $t$ as defined by Eq. (12), and $y \in \mathbb{R}^{2d}$ be a weight vector. A linear discriminator, or *perceptron*,

$$\pi(x,y) = \theta\left(\langle x,y\rangle\right), \tag{16}$$

can be used to model the decision process in a compact manner, where,

$$\langle x,y\rangle = \sum_i x_i y_i.$$

For a given vector, $y$, that represents the policy, and a given vector, $x$, that represents the state of the system, a "take" decision occurs if $\pi(x,y) \geqslant 0$, otherwise the elevator will ignore the customer.

The most obvious heuristic policy is the greedy one, i.e., when given the choice, always serve the customer. The $2d$–dimensional vector, $y_0 = (1,1,\ldots,1)^\top$, can be used to represent the greedy policy. This vector guarantees that the result in Eq. (16) equals 1, which is interpreted as a "take" decision. Rather counter–intuitively, this policy is not optimal, except in the case of heavy traffic. This means that a good policy must bypass some customers occasionally to prevent a phenomenon that is known as "bunching", which occurs in elevator systems when nearly all elevator cars are positioned in close proximity to one other.

The perceptron S–ring problem can serve as a benchmark problem for many optimization algorithms, since it relies on a fitness function, which maps $\mathbb{R}^{2d}$ to $\mathbb{R}$ [58, 60]. In general, $\pi$ can be realized as a look–up table of the system state, $x$, and $\pi^*$ is found by enumerating all possible $\pi$ and selecting the one with the lowest value of the function $Q$. Since this count grows exponentially with $d$, the enumerative approach would not work for any but the smallest cases.

**Table 10** Results for optimization of the S–ring model. Default designs (reported in [26, 29, 32]) and improved designs, for $k = 50$ repeats, are reported. The tuned inertia weight PSO variant is more robust than the default variant. It generates no outliers, as can be seen in the last column, but it was not able to find a better minimum.

| Design | Algorithm | Mean | Median | StD | Min | Max |
|---|---|---|---|---|---|---|
| $a_d^{(\text{Shi})}$ | PSO (iner.) | 2.6152 | 2.5726 | 0.4946 | 2.4083 | 5.9988 |
| $a_d^*$ | PSO (iner.) | 2.5171 | 2.5112 | 0.0754 | 2.4127 | 2.6454 |
| $a_d^{(\text{Clerc})}$ | PSO (con.) | 4.1743 | 2.6252 | 1.7021 | 2.5130 | 5.9999 |
| $a_d^*$ | PSO (con..) | 4.1707 | 2.6253 | 1.7055 | 2.5164 | 5.9999 |
| $a_d^{(\text{Lagarias})}$ | NMS | 4.3112 | 4.3126 | 1.7059 | 2.6200 | 5.9999 |
| | Quasi–Newton | 4.3110 | 4.3126 | 1.7060 | 2.6186 | 5.9999 |

### 5.2 Experimental Results for the S–ring Model

The Rosenbrock function that was considered in the previous sections, was chosen to provide a comprehensible introduction to the sequential DACE approach. In the following, we will present a more realistic approach. The optimization practitioner can chose among different programs, e.g., different solvers from commercial software packages or open source optimization tools that can be downloaded from the Internet and their source codes can be modified. We will compare the performance of PSO (based on the open source implementation from the PSO TOOLBOX), with the NMS algorithm, and a Quasi–Newton method. The S–ring simulator was selected to define a 12–dimensional optimization problem with noisy function values. The number of function evaluations was limited to 1000 for each optimization run. This value appears to be realistic for real–world applications. The related problem design is reported in Table 4.

Similarly to the analysis for the Rosenbrock function, the constriction factor and inertia weight variant of PSO were analyzed. The former requires only 4 exogenous strategy parameters, whereas 7 parameters have to be specified for the latter. Optimizing the PSO inertia weight variant improved the algorithm's robustness as reported in Table 10. The average function value decreased from 2.61 to 2.51, which is a significant difference. However, it is very important to note that the minimum function value could not be improved, but increased slightly from 2.4083 to 2.4127. The tuning procedure was able to find an algorithm design that prevents outliers and produces robust solutions at the cost of an aggressive exploratory behavior. However, an increased probability of finding a solution that has a minimum function value could have been specified as an optimization goal, resulting in a different "optimal" design.

Although the function values look slightly better, the tuning process produced no significant improvement for the rest of the algorithms. The constriction factor PSO variant, as well as the NMS algorithm and the Quasi–Newton method were not able to escape from local optima. In contrast to the Rosenbrock function, many real world optimization problems have many local minima on flat plateaus. The distribution of local optima in the search space is unstructured. Therefore these algorithms were unable to escape plateaus of equal fitness. This behavior occurred independently from the parameterization of their exogenous strategy parameters. The inertia weight PSO variant that required the determination of 7 exogenous strategy parameters, outperformed the rest algorithms in this comparison. Whether this improved result was caused by the scaling property of the inertia weight is subject to further investigation.

## 6 Conclusions

Comparisons of two PSO variants, NMS and a Quasi–Newton algorithm, on the classical Rosenbrock function and a highly complex and noisy real world optimization problem, were presented. The proposed sequential approach provide effective and efficient means to improve the algorithms' performance, significantly, supporting the claim that it is a useful tool to support the experimenter in the selection process of a suitable algorithm. To give a comprehensive presentation, only the mean fitness values were considered. Different results may occur, if different goals are of interest.

Experimental results indicate that there is no generic algorithm that works equally well on any given problem. Even different instances of one problem may require different algorithms, or at least different parameterizations of the employed algorithms. None of the algorithms has proved in our study to be satisfying for every problem. The Quasi–Newton method, as expected, outperformed the other algorithms on the Rosenbrock function, but it failed completely on the elevator optimization problem, where the inertia weight PSO variant, which requires twice as many parameters as the constriction factor variant, performed best. A drawback of the proposed approach, which is common to all statistical methods in this field, is the determination of a good initial design. Future research will include work towards this direction.

**Acknowledgements** T. Bartz–Beielstein would like to thank Mike Preuss and Christian Lasarczyk for valuable discussions. T. Bartz–Beielstein's research was supported by the DFG as a part of the collaborative research center "Computational Intelligence" (531). We acknowledge the partial support of the "Pythagoras" research grant awarded by the Greek Ministry of Education and Religious Affairs and the European Union.

## References

[1] H.-P. Schwefel, I. Wegener, and K. Weinert, editors. *Advances in Computational Intelligence – Theory and Practice*. Natural Computing Series. Springer, Berlin, 2003.
[2] T. Beielstein, K. E. Parsopoulos, and M. N. Vrahatis. Tuning PSO parameters through sensitivity analysis. Technical Report CI 124/02, Department of Computer Science, University of Dortmund, Dortmund, Germany, 2002.
[3] Thomas Bartz-Beielstein, Marcel de Vegt, Konstantinos E. Parsopoulos, and Michael N. Vrahatis. Designing particle swarm optimization with regression trees. Interner Bericht des Sonderforschungsbereichs 531 *Computational Intelligence* CI–173/04, Universität Dortmund, Mai 2004.
[4] J. Kennedy and R.C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, 2001.
[5] J.A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.
[6] J. Kennedy and R.C. Eberhart. Particle swarm optimization. In *Proc. IEEE Int.'l Conf. on Neural Networks*, volume IV, pages 1942–1948, Piscataway, NJ, 1995. IEEE Service Center.
[7] M.A. Abido. Optimal design of power system stabilizers using particle swarm optimization. *IEEE Trans. Energy Conversion*, 17(3):406–413, 2002.
[8] D.K. Agrafiotis and W. Cedeno. Feature selection for structure–activity correlation using binary particle swarms. *Journal of Medicinal Chemistry*, 45(5):1098–1107, 2002.
[9] A.R. Cockshott and B.E. Hartman. Improving the fermentation medium for *Echinocandin B* production part ii: Particle swarm optimization. *Process Biochemistry*, 36:661–669, 2001.
[10] P.C. Fourie and A.A. Groenwold. The particle swarm optimization algorithm in size and shape optimization. *Struct. Multidisc. Optim.*, 23:259–267, 2002.
[11] E. C. Laskari, K. E. Parsopoulos, and M. N. Vrahatis. Particle swarm optimization for integer programming. In *Proceedings of the IEEE 2002 Congress on Evolutionary Computation*, pages 1576–1581, Hawaii (HI), USA, 2002. IEEE Press.
[12] E. C. Laskari, K. E. Parsopoulos, and M. N. Vrahatis. Particle swarm optimization for minimax problems. In *Proceedings of the IEEE 2002 Congress on Evolutionary Computation*, pages 1582–1587, Hawaii (HI), USA, 2002. IEEE Press.
[13] W.Z. Lu, H.Y. Fan, A.Y.T. Leung, and J.C.K. Wong. Analysis of pollutant levels in central hong kong applying neural network method with particle swarm optimization. *Environmental Monitoring and Assessment*, 79:217–230, 2002.
[14] C.O. Ourique, E.C. Biscaia, and J. Carlos Pinto. The use of particle swarm optimization for dynamical analysis in chemical processes. *Computers and Chemical Engineering*, 26:1783–1793, 2002.
[15] K. E. Parsopoulos, E. C. Laskari, and M. N. Vrahatis. Particle identification by light scattering through evolutionary algorithms. In *Proceedings of the 1st International Conference for Mathematics and Informatics for Industry*, pages 97–108, Thessaloniki, Greece, 2003.
[16] K. E. Parsopoulos, E. I. Papageorgiou, P. P. Groumpos, and M. N. Vrahatis. Evolutionary computation techniques for optimizing fuzzy cognitive maps in radiation therapy systems. *Lecture Notes in Computer Science (LNCS)*, 3102:402–413, 2004.
[17] E. I. Papageorgiou, K. E. Parsopoulos, P. P. Groumpos, and M. N. Vrahatis. Fuzzy cognitive maps learning through swarm intelligence. *Lecture Notes in Computer Science (LNAI)*, 3070:344–349, 2004.
[18] K. E. Parsopoulos and M. N. Vrahatis. Particle swarm optimization method for constrained optimization problems. In P. Sincak, J. Vascak, V. Kvasnicka, and J. Pospichal, editors, *Intelligent Technologies–Theory and Application: New Trends in Intelligent Technologies*, volume 76 of *Frontiers in Artificial Intelligence and Applications*, pages 214–220. IOS Press, 2002.

[19] K. E. Parsopoulos and M. N. Vrahatis. Recent approaches to global optimization problems through particle swarm optimization. *Natural Computing*, 1(2–3):235–306, 2002.
[20] K. E. Parsopoulos and M. N. Vrahatis. Computing periodic orbits of nondifferentiable/discontinuous mappings through particle swarm optimization. In *Proceedings of the IEEE Swarm Intelligence Symposium*, pages 34–41, Indianapolis (IN), USA, 2003.
[21] K. E. Parsopoulos and M. N. Vrahatis. On the computation of all global minimizers through particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):211–224, 2004.
[22] T. Ray and K.M. Liew. A swarm metaphor for multiobjective design optimization. *Engineering Optimization*, 34(2):141–153, 2002.
[23] A. Saldam, I. Ahmad, and S. Al-Madani. Particle swarm optimization for task assignment problem. *Microprocessors and Microsystems*, 26:363–371, 2002.
[24] V. Tandon, H. El-Mounayri, and H. Kishawy. Nc end milling optimization using evolutionary computation. *Int.'l J. Machine Tools & Manufacture*, 42:595–605, 2002.
[25] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.
[26] Y. Shi and R.C. Eberhart. Empirical study of particle swarm optimization. In P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, editors, *Proceedings of the Congress of Evolutionary Computation*, volume 3, pages 1945–1950, 1999.
[27] R.C. Eberhart and Y. Shi. Comparison between genetic algorithms and particle swarm optimization. In V.W. Porto, N. Saravanan, D. Waagen, and A.E. Eiben, editors, *Evolutionary Programming*, volume VII, pages 611–616. Springer, 1998.
[28] T. Bartz-Beielstein, K. E. Parsopoulos, and M. N. Vrahatis. Analysis of particle swarm optimization using computational statistics. In Simos and Tsitouras [61], pages 34–37.
[29] M. Clerc and J. Kennedy. The particle swarm – explosion, stability, and convergence in a multidimensional complex space. *IEEE Trans. Evol. Comput.*, 6(1):58–73, 2002.
[30] J. Kennedy. Bare bones particle swarms. In *Proc. 2003 IEEE Swarm Intelligence Symposium*, pages 80–87. IEEE Press, 2003.
[31] R. M. Lewis, V. Torczon, and M. W. Trosset. Direct search methods: Then and now. *Journal of Computational and Applied Mathematics*, 124(1–2):191–207, 2000.
[32] 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*, 9(1):112–147, 1998.
[33] J. E. Gentle, W. Härdle, and Y. Mori, editors. *Handbook of Computational Statistics*. Springer, 2004.
[34] D. C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, New York, NY, 5th edition, 2001.
[35] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.
[36] T.J. Santner, B.J. Williams, and W.I. Notz. *The Design and Analysis of Computer Experiments*. Springer, 2003.
[37] Thomas Bartz-Beielstein and Sandor Markon. Tuning search algorithms for real-world applications: A regression tree based approach. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 1111–1118, Portland, Oregon, 20-23 June 2004. IEEE Press.
[38] J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4(4):409–435, 1989.
[39] E. H. Isaaks and R. M. Srivastava. *An Introduction to Applied Geostatistics*. Oxford University Press, 1989.
[40] S.N. Lophaven, H.B. Nielsen, and J. Søndergaard. Aspects of the Matlab Toolbox DACE. Technical Report IMM-REP-2002-13, Informatics and Mathematical Modelling, Technical University of Denmark, 2002.
[41] H. H. Hoos. *Stochastic Local Search – Methods, Models, Applications*. PhD thesis, Technische Universität Darmstadt, 1998.
[42] J.J. More, B.S. Garbow, and K.E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, 7(1):17–41, 1981.
[43] R. Aslett, R. J. Buck, S. G. Duvall, J. Sacks, and W. J. Welch. Circuit optimization via sequential computer experiments: design of an output buffer. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 47(1):31–48, 1998.
[44] K. Chan, A. Saltelli, and S. Tarantola. Sensitivity analysis of model output: variance-based methods make the difference. In *Proceedings of the 29th conference on Winter simulation*, pages 261–268. ACM Press, 1997.
[45] A. Saltelli, S. Tarantola, and F. Campolongo. Sensitivity anaysis as an ingredient of modeling. *Statistical Science*, 15(4):377–395, 2000.
[46] K. S. Chan, A. Tarantola, I. M. Saltelli, and I. M. Sobol'. Variance based methods. In *Sensitivity Analysis* [45], pages 167–197.
[47] A. Saltelli. Making best use of model evaluations to compute sensitivity indices. *Computer Physics Communications*, 145:280–297, May 2002.
[48] S.N. Lophaven, H.B. Nielsen, and J. Søndergaard. DACE - A Matlab Kriging Toolbox. Technical Report IMM-REP-2002-12, Informatics and Mathematical Modelling, Technical University of Denmark, 2002.
[49] Paul R. Cohen. *Empirical Methods for Artificial Intelligence*. MIT Press, Cambridge, MA, 1995.
[50] H.H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *Comp. J.*, 3:175–184, 1960.

[51] H.-P. Schwefel. *Evolution and Optimum Seeking*. Sixth-Generation Computer Technology. Wiley Interscience, New York, 1995.
[52] G. Barney. *Elevator Traffic Analysis, Design and Control*. Cambridge U.P., 1986.
[53] A.T. So and W.L. Chan. *Intelligent Building Systems*. Kluwer A.P., 1999.
[54] S. Markon and Y. Nishikawa. On the analysis and optimization of dynamic cellular automata with application to elevator control. The 10th Japanese-German Seminar, Nonlinear Problems in Dynamical Systems, Theory and Applications. Noto Royal Hotel, Hakui, Ishikawa, Japan, 2002.
[55] R.H. Crites and A.G. Barto. Elevator group control using multiple reinforcement learning agents. *Machine Learning*, 33(2-3):235–262, 1998.
[56] S. Markon. *Studies on Applications of Neural Networks in the Elevator System*. PhD thesis, Kyoto University, 1995.
[57] T. Beielstein, C.-P. Ewald, and S. Markon. Optimal elevator group control by evolution strategies. In E. Cantú-Paz, J. A. Foster, K. Deb, L. D. Davis, R. Roy, U.-M. O'Reilly, Hans-Georg Beyer, et al., editors, *Proc. Genetic and Evolutionary Computation Conf. (GECCO 2003), Chicago IL, Part II*, volume 2724 of *Lecture Notes in Computer Science*, pages 1963–1974, Berlin, 2003. Springer.
[58] S. Markon, D.V. Arnold, T. Bäck, T. Beielstein, and H.-G. Beyer. Thresholding – a selection operator for noisy ES. In J.-H. Kim, B.-T. Zhang, G. Fogel, and I. Kuscu, editors, *Proc. 2001 Congress on Evolutionary Computation (CEC'01)*, pages 465–472, Seoul, Korea, May 27–30, 2001. IEEE Press, Piscataway NJ.
[59] J.W. Bandler, Q.S. Cheng, S.A. Dakroury, A.S. Mohamed, M.H. Bakr, K. Madsen, and J. Søndergaard. Space mapping: the state of the art. *IEEE Transactions on Microwave Theory and Techniques*, 52(1):337–361, 2004.
[60] T. Beielstein and S. Markon. Threshold selection, hypothesis tests, and DOE methods. In David B. Fogel, Mohamed A. El-Sharkawi, Xin Yao, Garry Greenwood, Hitoshi Iba, Paul Marrow, and Mark Shackleton, editors, *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 777–782. IEEE Press, 2002.
[61] T. E. Simos and Ch. Tsitouras, editors. *International Conference on Numerical Analysis and Applied Mathematics 2004*, Weinheim, 2004. European Society of Computational Methods in Science and Engineering (ESCMSE), Wiley-VCH.
[62] P. M. A. Sloot, C. J. K. Tan, J. J. Dongarra, and A. G. Hoekstra, editors. *Computational Science - ICCS 2002: International Conference*, volume 2329 / 2002 of *Lecture Notes in Computer Science*. Springer, 2002.