Solving the G-problems in less than 500 iterations: Improved efficient constrained optimization by surrogate modeling and adaptive parameter control

Samineh Bagheri[a], Wolfgang Konen[a,*], Michael Emmerich[b], Thomas Bäck[b]

[a] *Department of Computer Science, TH Köln (University of Applied Sciences), 51643 Gummersbach, Germany*
[b] *Leiden University, LIACS, 2333 CA Leiden, The Netherlands*

Abstract

Constrained optimization of high-dimensional numerical problems plays an important role in many scientific and industrial applications. Function evaluations in many industrial applications are severely limited and no analytical information about objective function and constraint functions is available. For such expensive black-box optimization tasks, the constraint optimization algorithm COBRA was proposed, making use of RBF surrogate modeling for both the objective and the constraint functions. COBRA has shown remarkable success in solving reliably complex benchmark problems in less than 500 function evaluations. Unfortunately, COBRA requires careful adjustment of parameters in order to do so.

In this work we present a new self-adjusting algorithm SACOBRA, which is based on COBRA and capable to achieve high-quality results with very few function evaluations and no parameter tuning. It is shown with the help of performance profiles on a set of benchmark problems (G-problems, MOPTA08) that SACOBRA consistently outperforms any COBRA algorithm with fixed parameter setting. We analyze the importance of the several new elements in SACOBRA and find that each element of SACOBRA plays a role to boost up the overall optimization performance. We discuss the reasons behind and get in this way a better understanding of high-quality RBF surrogate modeling.

Keywords:
optimization; constrained optimization; expensive black-box optimization; radial basis function; self-adjustment

*Corresponding Author

Email addresses: `{samineh.bagheri,wolfgang.konen}@th-koeln.de` (Wolfgang Konen), `{m.t.m.emmerich,T.H.W.Baeck}@liacs.leidenuniv.nl` (Thomas Bäck)

1. Introduction

Real-world optimization problems are often subject to constraints, restricting the feasible region to a smaller subset of the search space. It is the goal of constraint optimizers to avoid infeasible solutions and to stay in the feasible region, in order to converge to the optimum. However, the search in constraint black-box optimization can be difficult, since we usually have no a-priori knowledge about the feasible region and the fitness landscape. This problem even turns out to be harder, when only a limited number of function evaluations is allowed for the search. However, in industry good solutions are requested in very restricted time frames. An example is the well-known benchmark MOPTA08 [17].

In the past different strategies have been proposed to handle constraints. E. g., repair methods try to guide infeasible solutions into the feasible area. Penalty functions give a negative bias to the objective function value, when constraints are violated. Many constraint handling methods are available in the scientific literature, but often demand for a large number of function evaluations (e. g., results in [35, 21]). Up to now, only little work has been devoted to *efficient* constraint optimization (severely reduced number of function evaluations). A possible solution in that regard is to use *surrogate models* for the objective and the constraint functions. While the real function might be expensive to evaluate, evaluations on the surrogate functions are usually cheap. As an example for this approach, the solver COBRA (Constrained Optimization by Radial Basis Function Approximation) was proposed by Regis [31] and outperforms many other algorithms on a large number of benchmark functions.

In our previous work [18, 19] we have studied a reimplementation of COBRA in `R` [30] enhanced by a new repair mechanism and reported its strengths and weaknesses. Although good results were obtained, each new problem required tedious manual tuning of the many parameters in COBRA. In this paper we follow a more unifying path and present SACOBRA (Self-Adjusting COBRA), an extension of COBRA which starts with the same settings on all problems and adjusts all necessary parameters internally.[1] This is *adaptive parameter control* according to the terminology introduced by Eiben et al. [10]. We present extensive tests of SACOBRA and other algorithms on a well-known and popular benchmark from the literature: The so-called G-problem or G-function benchmark was introduced by Michalewicz and Schoenauer [22] and Floudas and Pardalos [14]. It provides a set of constrained optimization problems with a wide range of different conditions.

We define the following research questions for the constrained optimization experiments in this work:

(H1) Do numerical instabilities occur in RBF surrogates and is it possible to avoid them?

(H2) Is it possible with SACOBRA to start with the same initial parameters on all G-problems and to solve them by self-adjusting the parameters on-line?

(H3) Is it possible with SACOBRA to solve all G-problems in less than 500 function evaluations?

[1] SACOBRA is available as open-source `R`-package from CRAN: `https://cran.r-project.org/web/packages/SACOBRA`

1.1. Related work

Following the surveys on constraint optimization given by Michalewicz and Schoenauer [22], Eiben and Smith [11], Coello Coello [7], Jiao et al. [16], and Kramer [20], several approaches are available for **constraint handling**:

(i) unconstrained optimization with a penalty added to the fitness value for infeasible solutions

(ii) feasible solution preference methods and stochastic ranking

(iii) repair algorithms to resolve constraint violations during the search

(iv) multi-objective optimization, where the constraint functions are defined as additional objectives

A frequently used approach to handle constraints is to incorporate static or dynamic penalty terms (i) in order to stay in the feasible region [7, 20, 24]. Penalty functions can be very helpful for solving constrained problems, but their main drawback is that they often require additional parameters for balancing the fitness and penalty terms. Tessema and Yen [38] propose an interesting adaptive penalty method which does not need any parameter tuning.

Feasible solution preference methods (ii) [9, 23] always prefer feasible solutions to infeasible solutions. They may use too little information from infeasible solutions and risk getting stuck in local optima. Deb [9] improves this method by introducing a diversity mechanism. Stochastic ranking [35, 36] is a similar and very successful improvement: With a certain probability an infeasible solution is ranked not behind, but – according to its fitness value – *among* the feasible solutions. Stochastic ranking has shown good results on all 11 G-problems. However it requires usually a large number of function evaluations (300 000 and more) and is thus not well suited for efficient optimization.

Repair algorithms (iii) try to transform infeasible solutions into feasible ones [5, 40, 19]. The work of Chootinan and Chen [5] shows very good results on 11 G-problems, but requires a large number of function evaluations (5 000 – 500 000) as well.

In recent years, multi-objective optimization techniques (iv) have attracted increasing attention for solving constrained optimization problems. The general idea is to treat the constraints as one or more objective functions to be optimized in conjunction with the fitness function. Coello Coello and Montes [8] use Pareto dominance-based tournament selection for a genetic algorithm (GA). Similarly, Venkatraman and Yen [39] propose a two-phase GA, where the second phase is formulated as a bi-objective optimization problem which uses non-dominated ranking. Jiao et al. [16] use a novel selection strategy based on bi-objective optimization and get improved reliability on a large number of benchmark functions. Emmerich et al. [12] use Kriging models for approximating constraints in a multi-objective optimization scheme.

In the field of **model-assisted optimization** algorithms for constrained problems, support vector machines (SVMs) have been used by Poloczek and Kramer [26]. They make use of SVMs as a classifier for predicting the feasibility of solutions, but achieve only slight improvements. Powell [28] proposes COBYLA, a direct search method which models the objective and the constraints using linear approximation. Recently, Regis [31] developed COBRA, an efficient solver that makes use of Radial Basis Function (RBF) interpolation to model objective and constraint functions, and outperforms most algorithms in terms of required function evaluations on a large number of benchmark functions. Tenne and Armfield [37] present an adaptive topology RBF network to tackle highly multimodal functions. But they consider only unconstrained optimization.

Most optimization algorithms need their parameter to be set with respect to the specific optimization problem in order to show good performance. Eiben et al. [10] introduced a terminology for parameter settings for evolutionary algorithms: They distinguish parameter tuning (before the run) and parameter control (online). Parameter control is further subdivided into predefined control schemes (deterministic), control with feedback from the optimization run (adaptive), or control where the parameters are part of the evolvable chromosome (self-adaptive).

Several papers deal with **adaptive or self-adaptive parameter control** in unconstrained or constrained optimization: Qin and Suganthan [29] propose a self-adaptive differential evolution (DE) algorithm. Brest et al. [2] propose another self-adaptive DE algorithm. But they do not handle constraints, whereas Zhang et al. [41] describe a constraint-handling mechanism for DE. We will compare later our results with the DE-implementation `DEoptimR`[2] which is based on both works [2, 41]. Farmani and Wright [13] propose a self-adaptive fitness formulation and test it on all 11 G-problems. They show comparable results to stochastic ranking [35], but require many function evaluations (above 300 000) as well. Coello Coello [6] and Tessema and Yen [38] propose self-adaptive penalty approaches. A survey of self-adaptive penalty approaches is given in [10].

The area of **efficient constrained optimization**, that is optimization under severely limited budget of less than 1000 function evaluations, is attracting more and more attention in recent years: Regis proposed besides the already mentioned COBRA approach [31] a trust-region evolutionary algorithm [33] which uses RBF surrogates as well and which exhibits high-quality results on many but not all G-functions in less than 1000 function evaluations. Jiao et al. [16] propose a self-adaptive selection method to combine informative feasible and infeasible solutions and they formulate it as a multi-objective problem. Their algorithm can solve some of the G-functions (G08,G11,G12) really fast in less than 500 evaluations, some others are solved with less than 10 000 evaluations, but the remaining G-functions (G01-G03,G07,G10) require 20 000 to 120 000 evaluations to be solved. Zahara and Kao [40] show similar results ($1000 - 20\,000$ evaluations) on some G-functions, but they investigate only G04, G08, and G12. To the best of our knowledge there is currently no approach which can solve all 11 G-problems in less than 1 000 evaluations. Tenne and Armfield [37] present an interesting approach with approximating RBFs to optimize highly multimodal functions in less than 200 evaluations, but their results are only for unconstrained functions and they are not competitive in terms of precision.

The rest of this paper is organized as follows: In Sec. 2 we present the constrained optimization problem and our methods: the RBF surrogate modeling technique, the COBRA-R algorithm and the SACOBRA algorithm. In Sec. 3 we perform a thorough experimental study on analytical test functions and on a real-world benchmark function MOPTA08 [17] from the automotive domain. We analyze with the help of data profiles the impact of the various SACOBRA elements on the overall performance. The results are discussed in Sec. 4

[2]R-package `DEoptimR`, available from `https://cran.r-project.org/web/packages/DEoptimR`

and we give conclusive remarks in Sec. 5.

2. Methods

2.1. Constrained optimization

A constrained optimization problem can be defined by the minimization of an objective function f subject to constraint function(s) $g_1, \dots, g_m$:

$$\begin{aligned} &\text{Minimize} \quad && f(\vec{x}) \\ &\text{subject to} && \\ & && g_i(\vec{x}) \leq 0, \qquad i = 1, 2, \dots, m, \\ & && \vec{x} \in [\vec{a}, \vec{b}] \quad \subset \mathbb{R}^d \end{aligned} \tag{1}$$

In this paper we always consider minimization problems. Maximization problems are transformed to minimization without loss of generality. Problems with equality constraints have them transformed to inequalities first (see Sec. 3.1 and Sec. 4.2).

2.2. Radial Basis Functions

The COBRA algorithm incorporates optimization on auxiliary functions, e. g. regression models over the search space. Although numerous regression models are available, we employ interpolating RBF models [4, 27], since they outperform other models in terms of efficiency and quality. In this paper we use the same notation as Regis [32]. RBF models require as input a set of design points (a training set): n points $\vec{u}^{(1)}, \dots, \vec{u}^{(n)} \in \mathbb{R}^d$ are evaluated on the real function $f(\vec{u}^{(1)}), \dots, f(\vec{u}^{(n)})$. We use an interpolating radial basis function as approximation:

$$s^{(n)}(\vec{x}) = \sum_{i=1}^{n} \lambda_i \varphi(||\vec{x} - \vec{u}^{(i)}||) + p(\vec{x}), \quad \vec{x} \in \mathbb{R}^d \tag{2}$$

Here, $|| \cdot ||$ is the Euclidean norm, $\lambda_i \in \mathbb{R}$ for $i = 1, \dots, n$, $p(\vec{x}) = c_0 + \vec{c}\vec{x}$ is a linear polynomial in d variables with $d+1$ coefficients $\vec{c}\,' = (c_0, \vec{c})^T = (c_0, c_1, \dots, c_d)^T \in \mathbb{R}^{d+1}$, and φ is of cubic form $\varphi(r) = r^3$. An alternative to cubic RBFs are Gaussian RBFs $\varphi(r) = e^{-r^2/(2\sigma^2)}$ which introduce an additional width parameter σ.

The RBF model fit requires a distance matrix $\mathbf{\Phi} \in \mathbb{R}^{n \times n}$: $\Phi_{ij} = \varphi(||\vec{u}^{(i)} - \vec{u}^{j}||), i, j = 1, \dots, n$. The RBF model requires the solution of the following linear system of equations:

$$\begin{bmatrix} \mathbf{\Phi} & \mathbf{P} \\ \mathbf{P}^T & \mathbf{0}_{(d+1)\times(d+1)} \end{bmatrix} \begin{bmatrix} \vec{\lambda} \\ \vec{c}\,' \end{bmatrix} = \begin{bmatrix} F \\ 0_{d+1} \end{bmatrix} \tag{3}$$

for the unknowns $\vec{\lambda}, \vec{c}\,'$. Where $\mathbf{P} \in \mathbb{R}^{n \times (d+1)}$ is a matrix with $(1, \vec{u}^{(i)})$ in its ith row, $\mathbf{0}_{(d+1)\times(d+1)} \in \mathbb{R}^{(d+1)\times(d+1)}$ is a zero matrix, 0_{d+1} is a vector of zeros, $F = (f(\vec{u}^{(1)}), \dots, f(\vec{u}^{(n)}))^T$, and $\vec{\lambda} = (\lambda_1, \dots, \lambda_n)^T \in \mathbb{R}^n$. The matrix in Eq. (3) is invertible if it has full rank. For this reason it is necessary to provide independent points in the initial design. This is usually the case, if $d+1$ linearly independent points are provided. The matrix inversion can be done efficiently by using singular value decomposition (SVD) or similar algorithms.

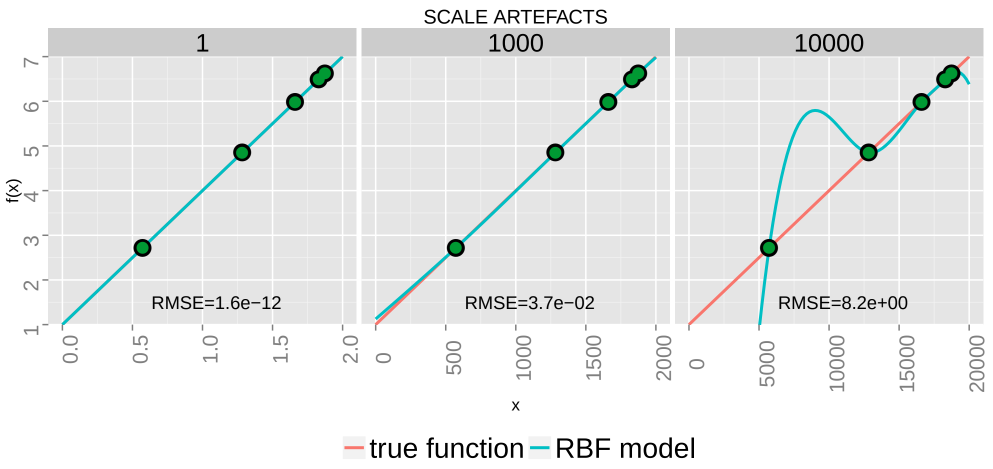


Figure 1: The influence of scaling. From left to right the plots show the RBF model fit for scale $S = 1, 1000, 10000$ (upper facet bar).

The linear polynomial $p(\vec{x})$ in Eq. (2) serves the purpose to alleviate the fit of simple linear functions $f()$ which otherwise have to be approximated by superimposing many RBFs in a complicated way. Polynomials of higher order may be used as well. We consider here the option of additional direct squares, where $p(\vec{x})$ in Eq. (2) is replaced by

$$p_{sq}(\vec{x}) = p(\vec{x}) + e_1 x_1^2 + \ldots + e_d x_d^2 \tag{4}$$

with additional coefficients $\vec{e} = (e_1, \ldots, e_d)^T$. The matrix in Eq. (3) is extended in a straightforward manner from an $(n+d+1) \times (n+d+1)$-matrix to an $(n+2d+1) \times (n+2d+1)$-matrix.

2.3. Common pitfalls in surrogate-assisted optimization

RBF models are very fast to train, even in high dimensions. They often provide good approximation accuracy even when only few training points are given. This makes them ideally suited as surrogate models for high-dimensional optimization problems with a large number of constraints.

There are however some pitfalls which should be avoided to achieve good modeling results for any surrogate-assisted black-box optimization.

2.3.1. Rescaling the input space

If a model is fitted with too large values in input space, a striking failure may occur. Consider the following simple example:

$$f(x) = 3\frac{x}{S} + 1 \tag{5}$$

where $x \in [0, 2S]$. If S is large, the x-values (which enter the RBF-model) will be large, although the output produced by Eq. (5) is exactly the same. Since the function $f(x)$ to be

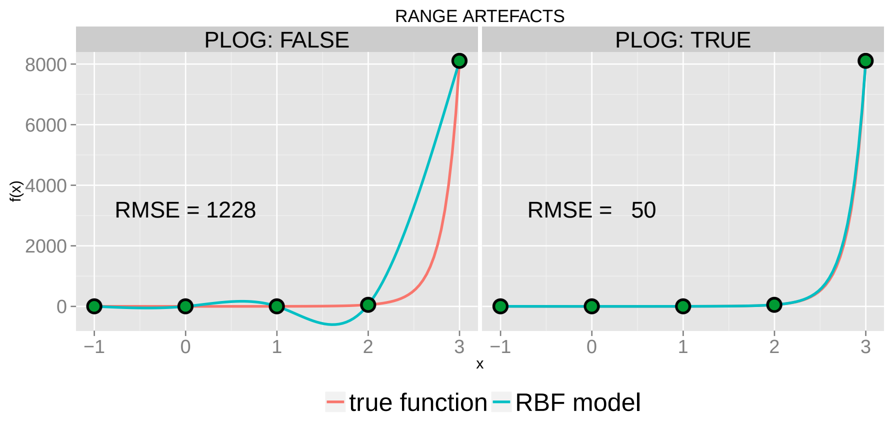


Figure 2: The influence of large output ranges. Left: Fitting the original function with a cubic RBF model. Right: Fitting the *plog*-transformed function with an RBF-model and transforming the fit back to original space with $plog^{-1}$.

modeled is exactly linear and the RBF-model contains a linear tail as well, one would expect at first sight a perfect fit (small RMSE) for each surrogate model. But – as Fig. 1 shows – this is not the case for large S: The fit (based on the same set of five points) is perfect for $S = 1$, weaker for $S = 1000$, and extremely bad in extrapolation for $S = 10000$.

The reason for this behavior: Large values for x lead to computationally singular (ill-conditioned) coefficient matrices, because the cubic coefficients tend to be many orders of magnitude larger than the coefficients for the linear part. Either the linear equation solver will stop with an error or it produces a result which may have large RMSE, as it is demonstrated in the right plot of Fig. 1. The solver sets the linear tail of the RBF model to zero in order to avoid numerical instabilities. The RBF model thus attempts to approximate the linear function with a superposition of cubic RBFs. This is bound to fail if the RBF model has to extrapolate beyond the green sample points.

This effect exactly occurs in problem G10, where the objective function is a simple linear function $x_1 + x_2 + x_3$ and the range for the input dimensions is large and different, e.g. $[100, 10000]$ for x_1 and $[10, 1000]$ for x_3.

The solution to this pitfall is simple: Rescale a given problem in all its input dimensions to a small and identical range, e.g. either to [0,1] or to [-1,1] for all x_i.

2.3.2. Logarithmic transform for large output ranges

Another pitfall are large output ranges in objective or constraint functions. As an example consider the function

$$f(x) = e^{x^2} \tag{6}$$

which has small values < 10 in the interval [-1,1] around its minimum, but quickly grows to large values above 8000 at $x = 3$. If we fit the original function with an RBF model using the green sample points shown in Fig. 2, we see in the left plot an oscillating behavior in the RBF function. This results in a large RMSE (approximation error).

The reason is that the RBF model tries to avoid large slopes. Instead the fitted model is similar to a spline function. Therefore it is a useful remedy to apply a logarithmic transform which puts the output into a smaller range and results in smaller slopes. Regis and Shoemaker [34] define the function

$$\text{plog}(y) = \begin{cases} +\ln(1+y) & \text{if} \quad y \geq 0 \\ -\ln(1-y) & \text{if} \quad y < 0 \end{cases} \tag{7}$$

which has – in contrast to the plain logarithm – no singularities and is strictly monotonous for all $y \in \mathbb{R}$. The RBF model can perfectly fit the *plog*-transformed function. Afterward we transform the fit with $plog^{-1}$ back to the original space and the back-transform takes care of the large slopes. As a result we get a much smaller approximation error RMSE in the original space, as the right-hand side of Fig. 2 shows.

We will apply the *plog*-transform only to functions with steep slopes in our surrogate-assisted optimization SACOBRA. For functions with flat or constant slope (e. g. linear functions) our experiments have shown that – due to the nonlinear nature of *plog* – the RBF approximation for $plog(f)$ is less accurate.

2.4. COBRA-R

The COBRA algorithm has been developed by Regis [31] with the aim to solve constrained optimization tasks with severely limited budgets. The main idea of COBRA is to do most of the costly optimization on surrogate models (RBF models, both for the objective function f and the constraint functions g_i). We reimplemented this algorithm in `R` [30] with a few small modifications. We give a short review of this algorithm COBRA-R in the following.

COBRA-R starts by generating an initial population P with n_{init} points (i. e. a random initial design[3], see Fig. 3) to build the first set of surrogate models . The minimum number of points is $n_{init} = d + 1$, but usually a larger choice $n_{init} = 3d$ gives better results.

Until the budget is exhausted, the following steps are iterated on the current population $P = \{\vec{x}_1, \ldots, \vec{x}_n\}$: The constrained optimization problem is executed by optimizing *on the surrogate functions*: That is, the true functions $f, g_1, \ldots, g_m$ are approximated with RBF surrogate models $s_0^{(n)}, s_1^{(n)}, \ldots, s_m^{(n)}$, given the n points in the current population P. In each iteration the COBRA-R algorithm solves with any standard constrained optimizer[4] the constrained surrogate subproblem

$$\begin{aligned} \text{Minimize} \quad & s_0^{(n)}(\vec{x}) \\ \text{subject to} \quad & \vec{x} \in [\vec{a}, \vec{b}] \subset \mathbb{R}^d, \\ & s_i^{(n)}(\vec{x}) + \epsilon^{(n)} \leq 0, \qquad i = 1, 2, \ldots, m \\ & \rho_n - ||\vec{x} - \vec{x}_j|| \leq 0, \qquad j = 1, \ldots, n. \end{aligned} \tag{8}$$

Compared to the original problem in Eq. (1) this subproblem uses surrogates and it contains two new elements $\epsilon^{(n)}$ and ρ_n which are explained in the next subsections. Before going into

[3] usually a latin hypercube sampling (LHS)

[4] Regis [31] uses MATLAB's FMINCON, an interior-point optimizer, which is not available in the `R` environment. In COBRA-R we use mostly Powell's COBYLA, but other constrained optimizer like ISRES are implemented in our `R`-package `https://cran.r-project.org/web/packages/SACOBRA` as well.

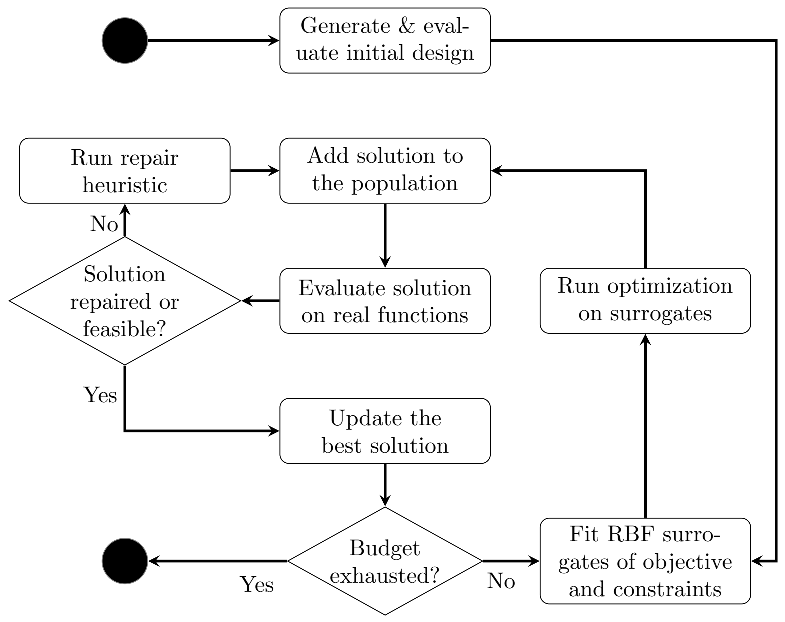

Generate & evaluate initial design
Run repair heuristic
Add solution to the population
No
Solution repaired or feasible?
Evaluate solution on real functions
Run optimization on surrogates
Yes
Update the best solution
Budget exhausted?
Yes
No
Fit RBF surrogates of objective and constraints

Figure 3: COBRA-R flowchart

these details we finish the description of the main loop: The optimizer returns a new solution $\vec{x}_{n+1}$. If $\vec{x}_{n+1}$ is not feasible, a repair algorithm RI2 described in our previous work [19] tries to replace it with a feasible solution in the vicinity.[5] In any case, the new solution $\vec{x}_{n+1}$ is evaluated on the true functions $f, g_1, \ldots, g_m$. It is compared to the best feasible solution found so far and replaces it if better. The new solution $\vec{x}_{n+1}$ is added to the population $P = \{\vec{x}_1, \ldots, \vec{x}_{n+1}\}$ and the next iteration starts with incremented n.

2.4.1. Distance requirement cycle

COBRA [31] applies a distance requirement factor which determines how close the next solution $\vec{x}_{n+1} \in \mathbb{R}^d$ is allowed to be to all previous ones. The idea is to avoid frequent updates in the neighborhood of the current best solution. The distance requirement can be passed by the user as external parameter vector $\Xi = \langle \xi^{(1)}, \xi^{(2)}, \ldots, \xi^{(\kappa)} \rangle$ with $\xi^{(i)} \in \mathbb{R}^{\geq 0}$. In each iteration n, COBRA selects the next element $\rho_n = \xi^{(i)}$ of Ξ and adds the constraints $||\vec{x} - \vec{x}_j|| \geq \rho_n, \quad j = 1, ..., n$ to the set of constraints. This measures the distance between the proposed infill solution and all n previous infill points. The distance requirement cycle (DRC) is a clever idea, since small elements in Ξ lead to more exploitation of the search space, while larger elements lead to more exploration. If the last element of Ξ is reached, the selection starts with the first element again. The size of Ξ and its single components can be arbitrarily chosen.

[5]RI2 is only rarely invoked on the G-problem benchmark but more often in the MOPTA08 case.

2.4.2. Uncertainty of constraint predictions

COBRA [31] aims at finding feasible solutions by extensive search on the surrogate functions. However, as the RBF models are probably not exact, especially in the initial phase of the search, a factor $\epsilon^{(n)}$ is used to handle wrong predictions of the constraint surrogates. Starting with $\epsilon^{(n)} = 0.005 \cdot l$, where l is the diameter of the search space, a point $\vec{x}$ is said to be *feasible in iteration* n if

$$s_i^{(n)}(\vec{x}) + \epsilon^{(n)} \leq 0 \quad \forall \quad i = 1, \ldots, m \tag{9}$$

holds. That is, we tighten the constraints by adding the factor $\epsilon^{(n)}$ which is adapted during the search. The $\epsilon^{(n)}$-adaptation is done by counting the feasible and infeasible infill points C_{feas} and C_{infeas} over the last iterations. When the number of these counters reaches the threshold for feasible or infeasible solutions, T_{feas} or T_{infeas}, respectively, we divide or double $\epsilon^{(n)}$ by 2 (up to a given maximum ϵ_{max}). When $\epsilon^{(n)}$ is decreased, solutions are allowed to move closer to the real constraint boundaries (the imaginary boundary is relaxed), since the last T_{feas} infill points were feasible. Otherwise, when no feasible infill point is found for T_{infeas} iterations, $\epsilon^{(n)}$ is increased in order to keep the points further away from the real constraint boundary.

2.4.3. Differences COBRA vs. COBRA-R

Although COBRA and COBRA-R are sharing many common principles, there are several differences which can lead to different results on identical problems. The main differences between COBRA [31] and COBRA-R [18] are listed as follows:

- COBRA is implemented in MATLAB while COBRA-R is implemented in `R`.
- *Internal optimizer:* COBRA uses MATLAB's FMINCON, an interior-point optimizer, COBRA-R uses COBYLA.[6]
- *Skipping phase 1:* COBRA has an additional phase 1 for searching the first feasible point.[7]
- *Repair algorithm:* COBRA-R has an additional repair algorithm RI2 [19].
- *Rescaling the input space*: COBRA rescales each dimension to [0,1], COBRA-R in its initial form [18] does not. See Sec. 2.5.1 for further remarks on rescaling.

2.5. SACOBRA

COBRA and COBRA-R achieve good results on most of the G-problems and on MOPTA08 as studies from Regis [31] and our previous work [19] have shown. However, it was necessary in both papers to carefully adjust the parameters of the algorithm to each problem and

[6]Other optimizers like ISRES and unconstrained optimizers with penalty are also available in COBRA-R / SACOBRA package, but not used in this paper.

[7] Phase 1 uses an objective function which rewards constraint fulfillment. We implemented this in COBRA-R as well but found it to be unnecessary for our problems. In this paper, COBRA-R always skips phase 1 and directly proceeds with phase 2 even if no feasible solution is found in the initialization phase.

Algorithm 1 SACOBRA. **Input:** Objective function f, set of constraint function(s) $\mathbf{g} = (g_1, \dots, g_m) : [\vec{a}, \vec{b}] \subset \mathbb{R}^d \to \mathbb{R}$ (see Eq. (1)), initial starting point $\vec{x}_{init} \in [\vec{a}, \vec{b}]$, maximum evaluation budget N_{max}. **Output:** The best solution $\vec{x}_{best}$ found by the algorithm.

```
 1: function SACOBRA($f, \mathbf{g}, \vec{x}_{init}, N_{max}$)
 2:     Rescale the input space to $[-1, 1]^d$
 3:     Generate a random initial population: $P = \{x_1, x_2, \cdots, x_{3 \cdot d}\}$
 4:     $(\widehat{FR}, \widehat{GR_i})$=AnalyseInitialPopulation($P, f, \mathbf{g}$)
 5:     $\widetilde{\mathbf{g}} \leftarrow$ AdjustConstraintFunctions($\widehat{GR_i}$, $\mathbf{g}$)
 6:     $\Xi \leftarrow$ AdjustDRC($\widehat{FR}$)
 7:     $Q \leftarrow$ AnalysePlogEffect($f, P, \vec{x}_{init}$)
 8:     $\vec{x}_{best} \leftarrow \vec{x}_{init}$
 9:     while (budget not exhausted, $|P| < N_{max}$) do
10:         $n \leftarrow |P|$
11:         $\widetilde{f}() \leftarrow (\; Q > 1 \;?\; plog(f()) \;:\; f() \;)$        ▷ see function plog in Eq. (7)
12:         Build surrogate models $\vec{s}^{(n)}$=($s_0^{(n)}, s_1^{(n)}, \cdots, s_m^{(n)}$) for ($\widetilde{f}, \widetilde{g}_1, \cdots, \widetilde{g}_m$)
13:         Select $\rho_n \in \Xi$ and $\epsilon_i^{(n)}$ according to COBRA base algorithm
14:         $\vec{x}_{start} \leftarrow$ RandomStart($\vec{x}_{best}, N_{max}$)
15:         $\vec{x}_{new} \leftarrow$ OptimCOBRA($\vec{x}_{start}, \vec{s}^{(n)}$)        ▷ see Eq. (8)
16:         Evaluate $\vec{x}_{new}$ on the real functions $\widetilde{f}, \widetilde{\mathbf{g}}$
17:         if ($|P| \mod 10 == 0$) then        ▷ every 10th iteration
18:             $Q \leftarrow$ AnalysePlogEffect($f, P, \vec{x}_{new}$)
19:         end if
20:         $\vec{x}_{new} \leftarrow$ repairRI2($\vec{x}_{new}$)        ▷ see Koch et al. [19] for details on RI2 (repair algo)
21:         $(P, \vec{x}_{best}) \leftarrow$ updateBest($P, \vec{x}_{new}, \vec{x}_{best}$)
22:     end while
23:     return $\vec{x}_{best}$
24: end function

25: function updateBest($P, \vec{x}_{new}, \vec{x}_{best}$)
26:     $P \leftarrow P \cup \{\vec{x}_{new}\}$
27:     if ($\vec{x}_{new}$ is feasible $AND$ $\vec{x}_{new} < \vec{x}_{best}$ ) then
28:         return $(P, \vec{x}_{new})$
29:     end if
30:     return $(P, \vec{x}_{best})$
31: end function
```

Algorithm 2 SACOBRA adjustment functions

```
1: function AnalyseInitialPopulation($P, f, \mathbf{g}$)
2:     $\widehat{FR} = \max_P f(P) - \min_P f(P)$                              ▷ range of objective function
3:     $\widehat{GR}_i = \max_P g_i(P) - \min_P g_i(P) \quad \forall i = 1, \ldots, m$
4: end function

5: function AdjustConstraintFunction($\widehat{GR}_i$,$\mathbf{g}$)
6:     $g_i() \leftarrow g_i() \cdot \frac{\mathrm{avg}(\widehat{GR}_i)}{\widehat{GR}_i} \quad \forall i = 1, \ldots, m$          ▷ see Eq. (10)
7:     return g
8: end function

9: function AdjustDRC( $\widehat{FR}$)
10:     if $\widehat{FR} > FR_l$ then                                          ▷ Threshold $FR_l = 1000$
11:         $\Xi = \Xi_s = \langle 0.001, 0.0 \rangle$
12:     else
13:         $\Xi = \Xi_l = \langle 0.3, 0.05, 0.001, 0.0005, 0.0 \rangle$
14:     end if
15: end function

16: function AnalysePlogEffect($f, P, \vec{x}_{new}$)                        ▷ $\vec{x}_{new} \notin P$
17:     $S_f \leftarrow$ surrogate model for $f()$ using all points in $P$
18:     $S_p \leftarrow$ surrogate model for $plog(f())$ using all points in $P$        ▷
    see function plog in Eq. (7)
19:     $E \leftarrow E \cup \left\{ \frac{|S_f(\vec{x}_{new}) - f(\vec{x}_{new})|}{|plog^{-1}(S_p(\vec{x}_{new})) - f(\vec{x}_{new})|} \right\}$    ▷ $E$, the set of approximation error ratios, is initially empty
20:     return $Q = \log_{10}(\mathrm{median}(E))$
21: end function
```

Algorithm 3 RandomStart (RS). **Input**: $\vec{x}_{best}$: the ever-best feasible solution. Parameters: restart probabilities $p_1 = 0.125$, $p_2 = 0.4$. **Output**: New starting point $\vec{x}_{start}$.

```
1: function RandomStart( $\vec{x}_{best}$)
2:     $F_{low} \leftarrow (|P_{feas}|/|P| < 0.05)$     ▷ True if less than 5% of the population are feasible
3:     $p \leftarrow (F_{low} = TRUE \ ? \ p_2 \ : \ p_1)$
4:     $\epsilon \leftarrow$ a random value $\in [0, 1]$
5:     if $(\epsilon < p)$ then
6:         $\vec{x}_{start} \leftarrow$ a random point in search space
7:     else
8:         $\vec{x}_{start} \leftarrow \vec{x}_{best}$
9:     end if
10:     return $(\vec{x}_{start})$
11: end function
```

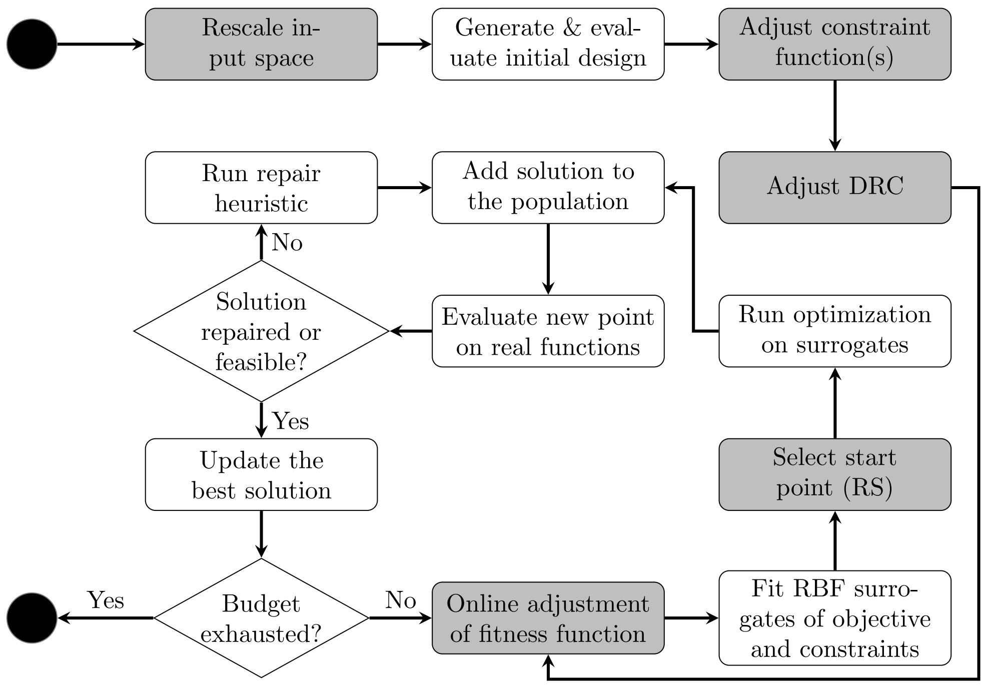

Rescale input space
Generate & evaluate initial design
Adjust constraint function(s)
Adjust DRC
Run repair heuristic
Add solution to the population
No
Solution repaired or feasible?
Evaluate new point on real functions
Run optimization on surrogates
Yes
Update the best solution
Select start point (RS)
Yes
Budget exhausted?
No
Online adjustment of fitness function
Fit RBF surrogates of objective and constraints

Figure 4: SACOBRA flowchart

sometimes even to modify the problem by applying a *plog*-transform (Eq. (7)) to the objective function or linear transformations to the constraints or by rescaling the input space. In real black-box optimization all these adjustments would probably require knowledge of the problem or several executions of the optimization code otherwise.

It is the main contribution of the current paper to present with SACOBRA (Self-Adjusting COBRA) an enhanced COBRA algorithm which has no needs for manual adjustment to the problem at hand. Instead, SACOBRA extracts during its execution information about the specific problem (either after the initialization phase or online during iterations) and takes internally appropriate measures to adjust its parameters or to transform functions.

We present in Fig. 4 the flowchart of SACOBRA where the five new elements compared to COBRA-R are highlighted as gray boxes. The complete SACOBRA algorithm is presented in detail in Algorithm 1 – 3. We describe in the following the five new elements in the order of their appearance:

2.5.1. Rescaling the input space

The input vector $\vec{x}$ is element-wise rescaled to $[-1, +1]$. This is done before the initialization phase. It helps to have a better exploration all over the search space because all dimensions are treated the same. More importantly, it avoids numerical instabilities caused by high values of $\vec{x}$ as shown in Sec. 2.3.1.

2.5.2. Adjusting constraint function(s) (aCF)

aCF is done by normalizing the range of constraint functions for each problem. The range $\widehat{GR}_i$ for the ith constraint is estimated from the initial population in Algorithm 2. Normalizing each $\widehat{GR}_i$ by the average constraint range

$$\operatorname{avg}\left(\widehat{GR}_i\right) = \frac{1}{m}\sum_i \widehat{GR}_i \tag{10}$$

helps to shift the range of all constraints as little as possible. Including this step aCF boosts up the optimization performance because all constraints operate now in a similar range.

2.5.3. Adjusting DRC parameter (aDRC)

aDRC is done after the initialization phase. Our experimental analysis showed that large DRC values can be harmful for problems with a very steep objective function, because a large move in the input space yields a very large change in the output space. This may spoil the RBF model in a sense similar to Sec. 2.3.2 and lead in consequence to large approximation errors. Therefore, we developed an automatic DRC adjustment which selects the appropriate DRC set according to the information extracted after the initialization phase. Function AnalyzePlogEffect in Algorithm 2 selects the 'small' DRC Ξ_s if the estimated objective function range $\widehat{FR}$ is larger than a threshold, otherwise it selects the 'large' DRC Ξ_l.

*2.5.4. Random start algorithm (*RS*)*

Normally COBRA starts optimization from the current best point. With RS (Algorithm 3), the optimization starts from a random point in the search space with a certain constant probability p_1. If the rate of feasibile individuals in the population P drops below 5% then we replace p_1 with a larger probability p_2. RS is especially beneficial when the search gets stuck in local optima or when it gets stuck in a region where no feasible point can be found.

2.5.5. Online adjustment of fitness function (aFF)

Our analysis in Sec. 2.3.2 has shown that a fitness function f with steep slopes poses a problem for RBF approximation. For some problems, modeling $plog(f)$ instead of f and transforming the RBF result back with $plog^{-1}$ boosts up the optimization performance significantly. On the other hand, our tests have shown that the *plog*-transform is harmful for some other problems. Therefore, a careful decision whether to use *plog* or not should be made.

The idea of our online adjustment algorithm (Algorithm 2, function AnalyzePlogEffect) is the following: Given the population P, we build RBFs for f and $plog(f)$, take a new point $\vec{x}_{new}$ not yet added to P, and calculate the ratio of approximation errors on $\vec{x}_{new}$ (line 15 of Algorithm 2). We do this in every kth iteration (usually $k = 10$) and collect these ratios in a set E. If

$$Q = \log_{10}\left(\operatorname{median}(E)\right) \tag{11}$$

is above 0, then the RBF for $plog(f)$ is better in the majority of the cases. Otherwise, the RBF on f is better.[8]

Step 11 of Algorithm 1 decides on the basis of this criterion Q which function $\widetilde{f}$ is used as RBF surrogate in the optimization step. Note that the decision for $\widetilde{f}$ taken in earlier iterations can be revoked in later iterations, if the majority of the elements in E shows that now the other choice is more promising.

This completes the description of our SACOBRA algorithm. SACOBRA is available as open-source `R`-package from CRAN.[9]

2.6. Performance Measures

In many papers on optimization the strength of an optimization technique is measured by comparing the final solution achieved by different algorithms [35]. This approach only provides the information about the quality of the results and neglects the speed of convergence which is a very important measure for expensive optimization problems. Comparing the convergence curve over time (number of function evaluations) is also one of the common benchmarking approaches [31]. Although a convergence curve provides good information about the speed of convergence and the final quality of the optimization result, it can be used to compare performance of several algorithms only on *one* problem. It is often interesting to compare the overall capability of a technique on solving a group of problems. The data and performance profiles developed by Moré and Wild [25] are a good approach to analyze the performance of any optimization algorithm on a whole test suite and are now used frequently in the optimization literature [3, 33].

2.6.1. Performance Profiles

Performance profiles are defined with the help of the performance ratio

$$r_{p,s} = \frac{t_{p,s}}{\min\limits_{\forall s' \in \mathbb{S}} \{t_{p,s'}\}}, \qquad p \in \mathbb{P} \tag{12}$$

where $\mathbb{P}$ is a set of problems, $\mathbb{S}$ is a set of solvers and $t_{p,s}$ is the number of iterations solver $s \in \mathbb{S}$ requires to solve problem $p \in \mathbb{P}$. A problem is said to be *solved* when a feasible objective value $f(x)$ is found which is not more than τ larger than the best objective f_L determined by any solver in S:

$$f(x) - f_L \leq \tau \tag{13}$$

We use $\tau = 0.05$ for all our experiments below. Smaller values are more desirable for the performance ratio $r_{p,s}$. When using the best solver s to solve problem p then $r_{p,s} = 1$. If a solver s cannot solve problem p the performance ratio is set to infinity. The performance

[8]Our experimental analysis on the G-problem test suite will show (Sec. 3.4) that a threshold 1 is slightly more robust than 0. We use this threshold 1 in step 11 of Algorithm 1, but the difference to threshold 0 is only marginal.

[9]`https://cran.r-project.org/web/packages/SACOBRA`

Table 1: Characteristics of the G-functions: d: dimension, type of fitness function, ρ^*: feasibility rate (%) after changing equality constraints to inequality constraints, FR: range of the fitness values, GR: ratio of largest to smallest constraint range, LI: number of linear inequalities, NI: number of nonlinear inequalities, NE: number of nonlinear equalities, a: number of constraints active at the optimum.

Fct.	d	type	ρ^*	FR	GR	LI	NI	NE	a
G01	13	quadratic	0.0003%	298.14	1.969	9	0	0	6
G02	10	nonlinear	99.997%	0.57	2.632	1	1	0	1
G03	20	nonlinear	0.0000%	92684985979.23	1.000	0	0	1	1
G04	5	quadratic	26.9217%	9832.45	2.161	0	6	0	2
G05	4	nonlinear	0.0919%	8863.69	1788.74	2	0	3	3
G06	2	nonlinear	0.0072%	1246828.23	1.010	0	2	0	2
G07	10	quadratic	0.0000%	5928.19	12.671	3	5	0	6
G08	2	nonlinear	0.8751%	1821.61	2.393	0	2	0	0
G09	7	nonlinear	0.5207%	10013016.18	25.05	0	4	0	2
G10	8	linear	0.0008%	27610.89	3842702	3	3	0	3
G11	2	linear	66.7240%	4.99	1.000	0	0	1	1

profile ρ_s is now defined as a function of the steerable performance factor α:

$$\rho_s(\alpha) = \frac{1}{|\mathbb{P}|}|\{p \in \mathbb{P} : r_{p,s} \leq \alpha\}|. \tag{14}$$

2.6.2. Data Profiles

Data profiles are appropriate for evaluating optimization algorithms on expensive problems. They are defined as

$$d_s(\alpha) = \frac{1}{|\mathbb{P}|}|\{p \in \mathbb{P} : \frac{t_{p,s}}{d_p + 1} \leq \alpha\}|, \tag{15}$$

with $\mathbb{P}, \mathbb{S}$ and $t_{p,s}$ defined as above and d_p as the dimension of problem p.

We prefer data profiles over performance profiles, because the performance factor α has a more intuitive meaning for data profiles: If we allow for each problem with dimension d_p a budget of $B_\alpha = \alpha(d_p + 1)$ function evaluations, then the value $d_s(\alpha)$ can be interpreted as the fraction of problems which solver s can solve within this budget B_α.

3. Experiments

3.1. Experimental Setup

We evaluate SACOBRA by using a well-studied test suite of G-problems described in [14, 22]. The diversity of the G-problem characteristics makes them a very challenging benchmark for optimization techniques. In Table 1 we show features of these problems. The features ρ^*, FR and GR (defined in Table 1) are measured by Monte Carlo sampling with 10^6 points in the search space of each G-problem.

Equality constraints are treated by replacing each equality operator with an inequality operator of the appropriate direction. This approach (same as in Regis' work [31]) takes

Table 2: The default parameter setting used for COBRA. l is the length of the smallest side of the search space (after rescaling, if rescaling is done). The settings for T_{feas}, T_{infeas} proportional to $\sqrt{d}$ (d: dimension of problem) are taken from [31].

parameter	value	
	COBRA-R	SACOBRA
ϵ_{init}	$0.005 \cdot l$	$0.005 \cdot l$
ϵ_{max}	$0.01 \cdot l$	$0.01 \cdot l$
T_{feas}	$\lfloor 2\sqrt{d} \rfloor$	$\lfloor 2\sqrt{d} \rfloor$
T_{infeas}	$\lfloor 2\sqrt{d} \rfloor$	$\lfloor 2\sqrt{d} \rfloor$
Ξ	$\{0.3, 0.05, 0.001, 0.0005, 0.0\}$	Adaptive
$plog(.)$	Never	Adaptive
aCF	Never	Always
RS	Never	Adaptive

as „appropriate direction“ this side of the equality hyperplane where the objective function increases.

The MOPTA08 benchmark by Jones [17] is a substitute for a high-dimensional real-world problem encountered in the automotive industry: It is a problem with $d = 124$ dimensions and with 68 constraints. The problem should be solved within $1860 = 15 \cdot d$ function evaluations. This corresponds to one month of computation time on a high-performance computer for the real automotive problem since the real problem requires time-consuming crash-test simulations.

The COBRA-R optimization framework allows the user to choose between several initialization approaches: Latin hypercube sampling (LHS), Biased and Optimized [18]. While LHS initialization is always possible (and is in fact used for all runs of the G-problem benchmark with $n_{init} = 3d$), the other algorithms are only possible if a feasible starting point is provided. In COBRA [31] the initialization is always done randomly by means of Latin hypercube sampling for functions without feasible starting point.

In the case of MOPTA08 a feasible point is known. We use the Optimized initialization approach, where an initial optimization run is started from this feasible point with the Hooke & Jeeves pattern search algorithm [15]. This initial run provides a set of $n_{init} = 500$ points in the vicinity of the feasible point. This set serves as initial design for MOPTA08.

Table 2 shows the parameter settings used for COBRA-R and SACOBRA. All G-problems were optimized with exactly the same initial parameter settings. In contrast to that, the COBRA results in Regis [31] and our previous work [18] were obtained by manually activating *plog* for some G-problems and by manually adjusting constraint factors and other parameters.

3.2. Convergence Curves

Figures 5 – 7 show the SACOBRA convergence plots for all G-problems. It is clearly visible that all problems except G02 are solved in the majority of runs, if we define *solved* as a target error below $\tau = 0.05$ in comparison to the true optimum. In some cases (G03, G05, G09, G10) the worst error does not meet the target, but in the other cases it does. In most cases, as indicated by the red squares, there is a clear improvement to Regis' COBRA results [31].

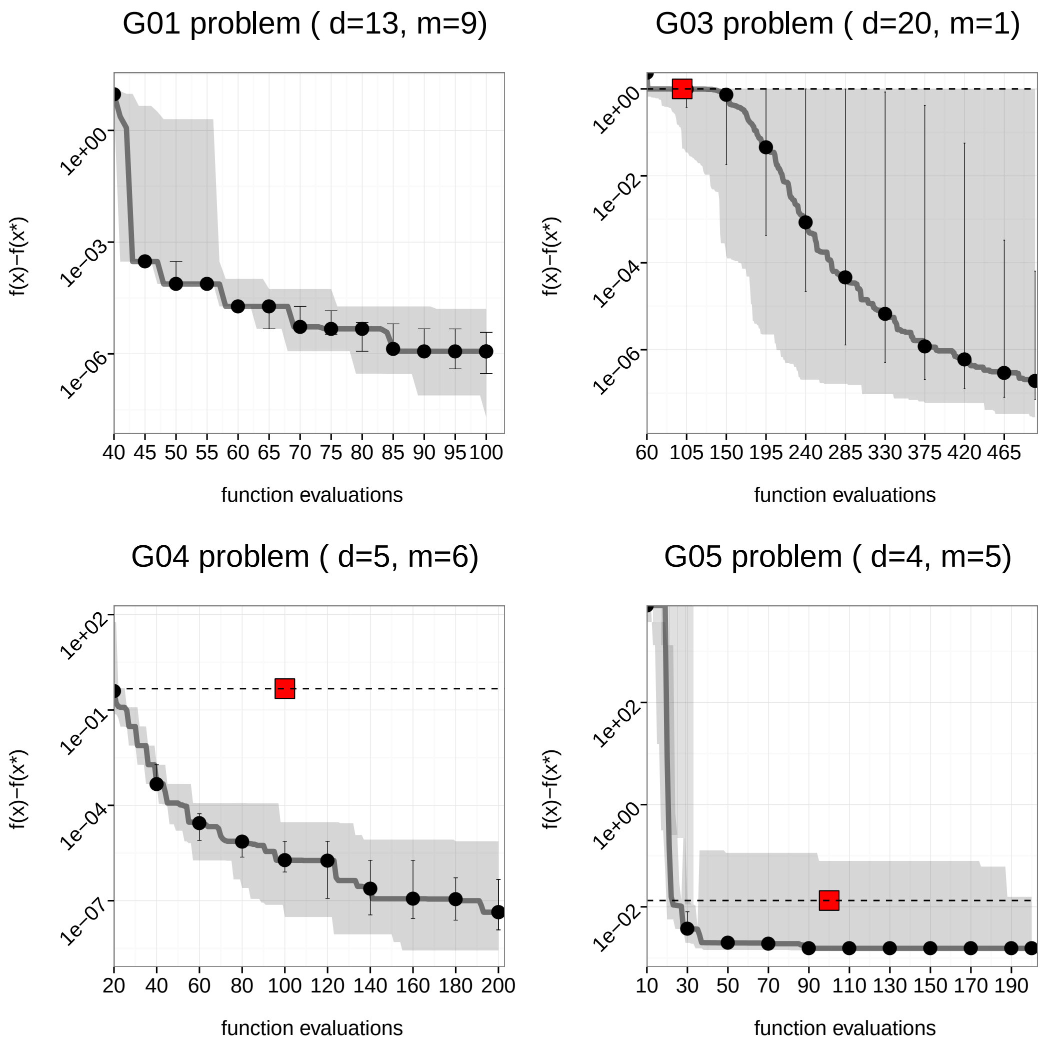


Figure 5: SACOBRA optimization process for G01 – G05. The gray curve shows the median of the error for 30 independent trials. The error is calculated with respect to the true minimum $f(x^*)$. The gray shade around the median is showing the worst and the best error. The error bars mark the 25% and 75% quartile. The red square is the result reported by Regis [31] after 100 iterations.

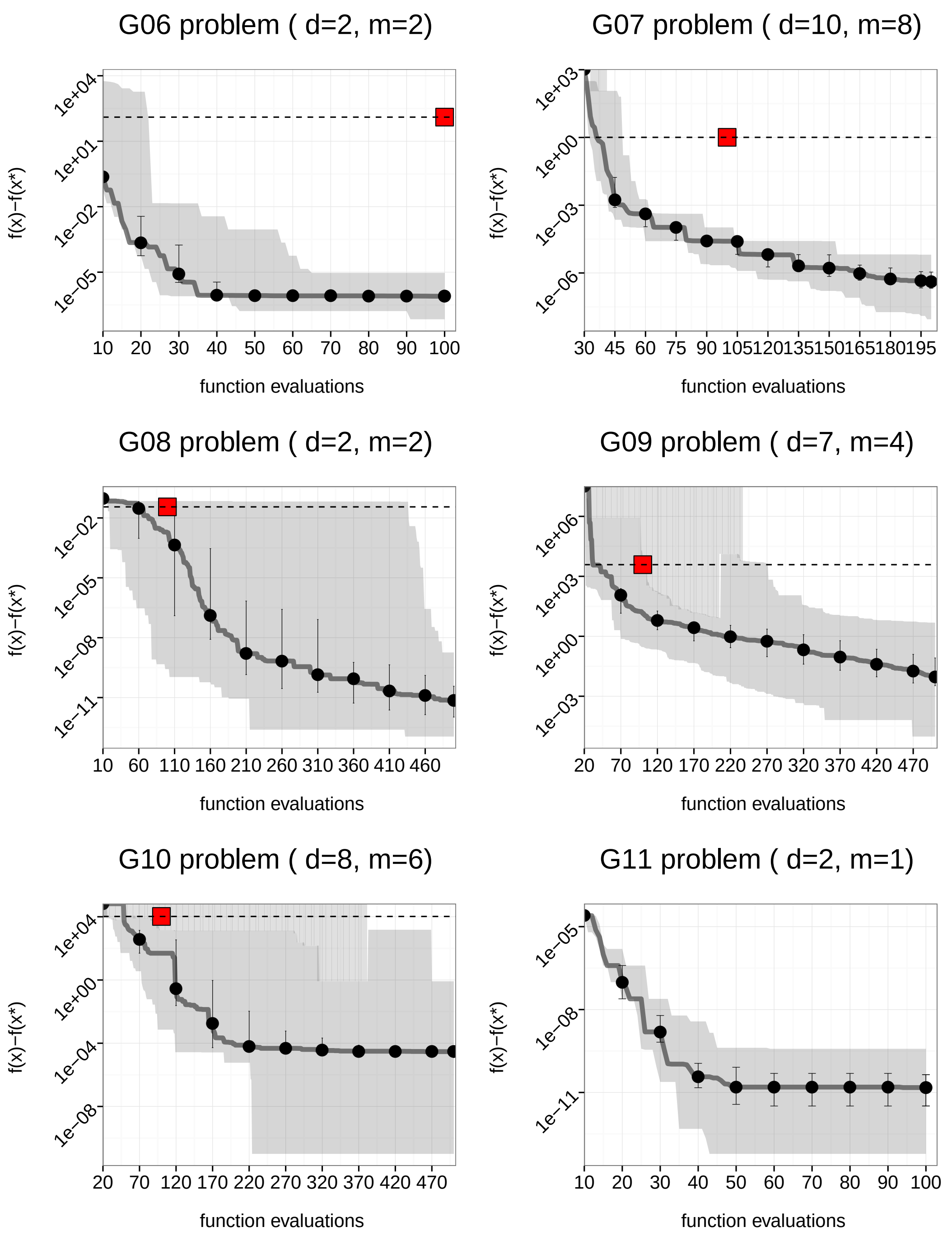
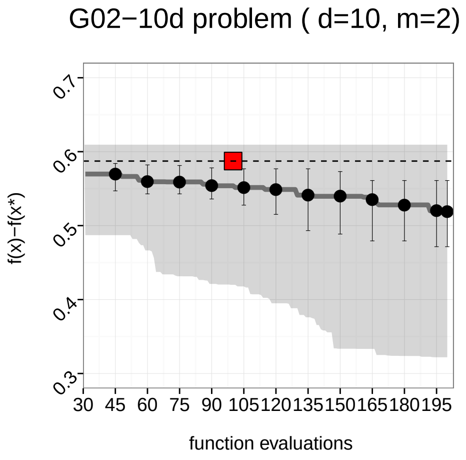


Figure 6: Same as Fig. 5 for G06 – G11.



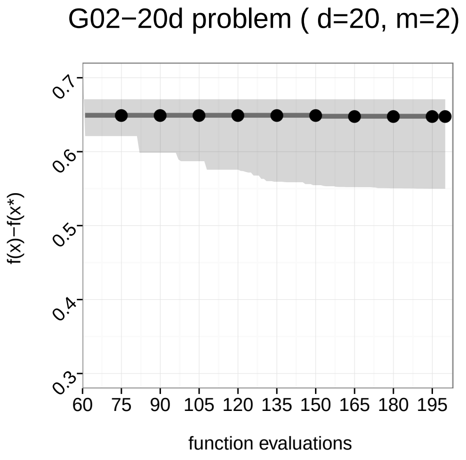


Figure 7: Same as Fig. 5 for G02 in 10 and 20 dimensions.

3.3. Performance Profiles

Our main result is shown in Fig.8. It shows the data profiles for different SACOBRA variants in comparison with the data profile for COBRA-R. COBRA-R was run with a fixed parameter set.[10] We note in passing that other fixed parameter settings for COBRA-R were tested, they were perhaps better on some of the runs but inevitably worse on other runs, so that in the end a similar or slightly worse data profile for COBRA-R would emerge. SACOBRA increases significantly the success rate on the G-problem benchmark suite.

In addition, we analyze in Fig. 8 the effect of the five elements of SACOBRA: The \-data profiles present the SACOBRA results when one specific of the five SACOBRA elements is switched off. We see that the strongest effects occur when rescale is switched off (early iterations) or when aFF is switched off (later iterations).

Fig. 9 shows that each of these elements has its relevance for some of the G-problems: The full SACOBRA method is compared with other SACOBRA- or COBRA-variants M$*$ on 30 runs. SACOBRA is significantly better than each M$*$ at least for some G-problems (each column has a dark cell). And each G-problem benefits from one or more SACOBRA extensions (each row has a dark cell). The only exception from this rule is G11, but for a simple reason: G11 is an easy problem which is solved by *all* SACOBRA variants in each run, so none is significantly better than the others.

3.4. Fitness Function Adjustment

By comparing the convergence curves of G-functions we realized that applying the logarithmic transform is strictly harmful for three of the G-functions, significantly beneficial for two other problems, and with negligible effect on the other problems. Therefore, a careful

[10]In our previous work [18, 19] we reported good results with COBRA-R, but this was with varying parameters and with tedious parameter tuning on each specific G-problem.

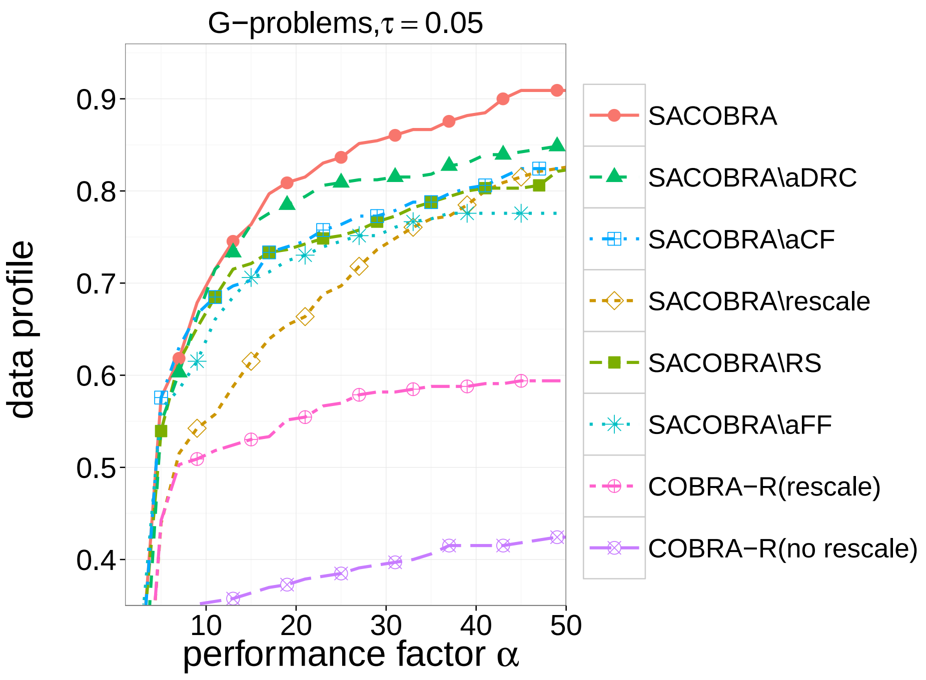


Figure 8: Analyzing the impact of different elements of SACOBRA on the G-problems. Data profile of SACOBRA, SACOBRA\rescale (SACOBRA without rescaling the input space), and other „\“-algorithms are with a similar meaning. COBRA-R is the old version of COBRA [18], i. e. SACOBRA with all adjustment extensions switched off. These algorithms are performed on 330 different problems (11 test problems from G-function suite which are initialized with 30 different initial design points).

selection should be done. Although we demonstrated in Sec. 2.3.2 that steep functions can be better modeled after the logarithmic transformation, it is not trivial to define a correct threshold to classify steep functions. Also, there is no direct relation between steepness of the function and the effect of logarithmic transformation on optimization. We defined in Sec. 2.5.5 and Algorithm 2, function AnalyzePlogEffect, a measure called Q in order to quantify online whether RBF models with and without *plog* transformation are better or worse.

Here we test by experiments whether the Q-value does a good job. Fig. 10 shows the Q-value for all G-problems. The G-problems are are ranked on the horizontal axis according to the impact of logarithmic transformation of the fitness function on the optimization outcome. This means that applying the *plog*-transformation has the worst effect for modeling the fitness of G01 and the best effect for G03. We measure the impact on optimization in the following way: For each G-problem we perform 30 runs with *plog* inactive and with *plog* active. We calculate the median of the final optimization error in both cases and take the ratio

$$R = \frac{\text{median}(E_{opt})}{\text{median}(E_{opt}^{(plog)})}. \tag{16}$$

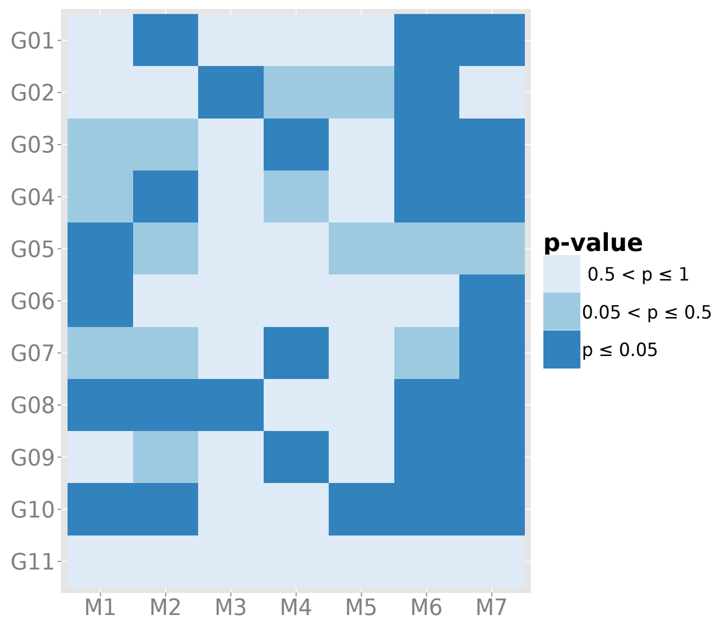


Figure 9: Wilcoxon rank sum test, paired, one sided, significance level 5%. Shown is the p-value for the hypothesis that for a specific G-problem the full SACOBRA method at the final iteration is better than another solver M$*$. Significant improvements ($p \leq 5\%$) are marked as cells with dark blue color. Optimization methods: M1: SACOBRA\rescale (SACOBRA without rescaling the input space), M2: SACOBRA\RS (SACOBRA without random start), M3: SACOBRA\aDRC, M4: SACOBRA\aFF, M5: SACOBRA\aCF, M6: COBRA ($\Xi = \Xi_s$), M7: COBRA ($\Xi = \Xi_l$).

Note that R is usually not available in normal optimization mode. If R is { close to zero, close to 1, much larger than 1 } then the effect of *plog* on optimization performance is { harmful, neutral, beneficial }. It is a striking feature of Fig. 10 that the Q-ranks are very similar to the R-ranks.[11] This means that the beneficial or harmful effect of *plog* is strongly correlated with the RBF approximation error.

Our experiments have shown that for all problems with $Q \in [-1, 1]$ the optimization performance is only weakly influenced by the logarithmic transformation of the fitness function. Therefore, in Step 19 of function AdjustFitnessFunction in Algorithm 2, any threshold in $[-1, 1]$ will work. We choose the threshold 1, because it has the largest margin to the colored bars in Fig. 10.

The G-problems for which *plog* is beneficial are G03 and G09: These are according to Table 1 the two problems with the largest fitness function range FR, thus strengthening our hypothesis from Sec. 2.3.2: For such functions a *plog*-transform should be used to get good RBF-models. The G-problems for which *plog* is harmful are G01, G07, and G10: Looking at

[11]The only notable difference, namely the switch in the order of G07 and G10, can be seen as an imperfection of measure R. Although G10 has rank 3 in R, it has weaker worst-case behavior than G07 because two G10 runs never produce a feasible solution if *plog* is active.

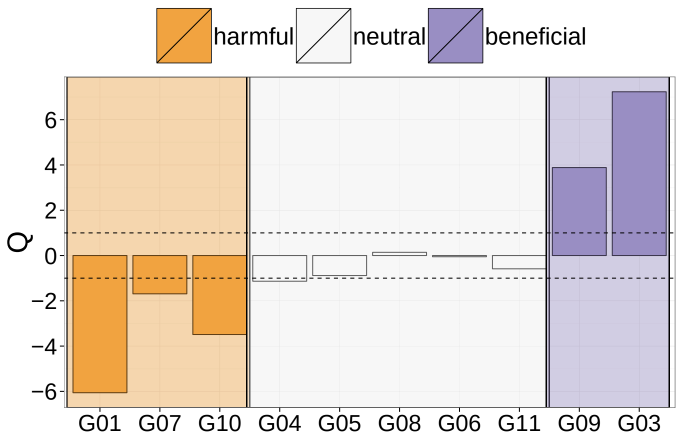


Figure 10: Q-value (Eq. (11)) at end of optimization for all G-problems. The G-problems are ordered along the x-axis according to the R-value defined in Eq. (16) which measures the impact of *plog* on the optimization performance. Any threshold for Q in $[-1, 1]$ will clearly separate the harmful from the beneficial problems. This figure shows that the online available Q is a good predictor of the impact of *plog* on the overall optimization performance.

the analytical form of the objective function in those problems[12] we can see that these are the only three functions being of quadratic type (Table 1) *and* having no mixed quadratic terms. Those functions can be fitted perfectly by the polynomial tail (Eq. (4)) in SACOBRA, if *plog* is *in*active. With *plog* they become nonlinear and a more complicated approximation by the radial basis functions is needed. This results in a larger approximation error.

3.5. Comparison with other optimizers

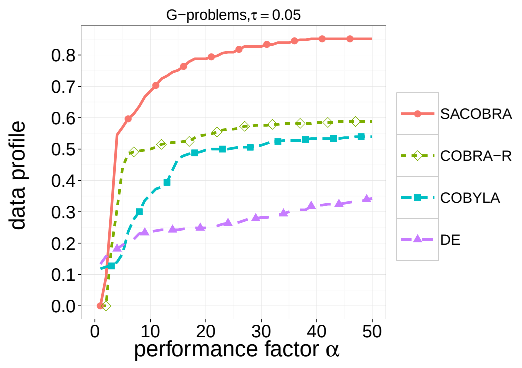
Table 3 shows the comparison with different state-of-the-art optimizers on the G-problem suite. While ISRES (Improved Stochastic Ranking [36]) and DE (Differential Evolution [2]) are the best optimizers in terms of solution quality, they require the highest number of function evaluations as well. SACOBRA has on most G-problems (except G02) the same solution quality, only G09 and G10 are very slightly worse. At the same time SACOBRA requires only a small fraction of function evaluations (fe): roughly 1/1000 as compared to ISRES and RGA and 1/300 as compared to DE (row *average fe* in Table 3).

G02 is marked in red cell color in Table 3 because it is not solved to the same level of accuracy by most of the optimizers. ISRES and RGA (Repair GA [5]) get close, but only after more than 300 000 fe. DE performs even better on G02, but requires more than 200 000 fe as well. SACOBRA and COBRA cannot solve G02.

[12]The analytical form is available in the appendices of [35] or [36].

Table 3: Different optimizers: median (m) of best feasible results and (fe) average number of function evaluations. Results from 30 independent runs with different random number seeds. Numbers in **boldface (blue)**: distance to the optimum ≤ 0.001. Numbers in *italic (red)*: reportedly better than the true optimum. COBYLA sometimes returns slightly infeasible solutions (number of infeasible runs in brackets).

Fct.	Optimum		SACOBRA	COBRA	ISRES	RGA 10%	COBYLA	DE
			[this work]	[31]	[36, 35]	[5]	[28] (infeas)	[2, 41]
G01	-15.0	m	**-15.0**	NA	**-15.0**	**-15.0**	-13.83	**-15.0**
		fe	100	NA	350000	95512	12743	59129
G02	-0.8036	m	-0.3466	NA	-0.7931	-0.7857	-0.197 (5)	**-0.8036**
		fe	400	NA	349600	331972	97391	226994
G03	-1.0	m	**-1.0**	-0.09	*-1.001*	-0.9999	**-1.0** (3)	-0.9999
		fe	300	100	349200	399804	31069	211966
G04	-30665.539	m	**-30665.539**	-30665.15	**-30665.539**	**-30665.539**	**-30665.539**	**-30665.539**
		fe	200	100	192000	26981	418	33963
G05	5126.497	m	**5126.498**	5126.51	**5126.497**	**5126.498**	**5126.498** (7)	**5126.498**
		fe	200	100	195600	39459	194	13375
G06	-6961.81	m	**-6961.81**	-6834.48	**-6961.81**	**-6961.81**	**-6961.81** (3)	**-6961.81**
		fe	100	100	168800	13577	134	2857
G07	24.306	m	**24.306**	25.32	**24.306**	24.471	**24.306** (6)	**24.306**
		fe	200	100	350000	428314	13072	94313
G08	-0.0958	m	**-0.0958**	*-0.1*	**-0.0958**	**-0.0958**	-0.0282	**-0.0958**
		fe	200	100	160000	6217	553	990
G09	680.630	m	680.761	3953.97	**680.630**	680.638	**680.630** (2)	**680.630**
		fe	300	100	271200	388453	8973	34836
G10	7049.248	m	7049.253	18031.74	**7049.248**	7049.566	7064.8 (22)	**7049.248**
		fe	300	100	348800	572629	270840	74875
G11	0.75	m	**0.75**	NA	**0.75**	**0.75**	**0.75**	**0.75**
		fe	100	NA	137200	7215	11788	2190
		average fe	218	100	261127	210012	40652	68681
		total fe	2400	800	2872400	2310133	447175	755488



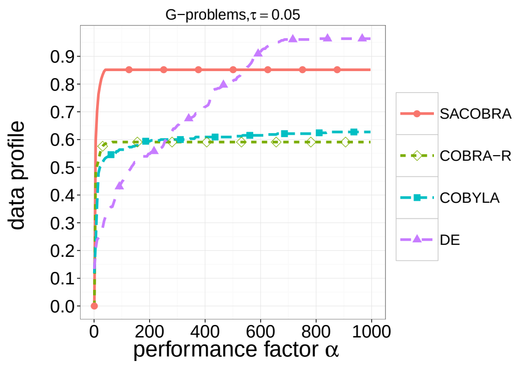


Figure 11: Comparing the performance of the algorithms SACOBRA, COBRA-R (with rescale), Differential Evolution (DE), and COBYLA on optimizing all G-problems G01-G11 (30 runs with different initial random populations).

The results in column SACOBRA, DE and COBYLA are from our own calculation in `R` while the results in column COBRA, ISRES and RGA were taken from the papers cited. In two cases (red italic numbers in Table 3) the reported solution is better than the true optimum, possibly due to a slight infeasibility. This is explicitly stated in the case of ISRES [35, p. 288], because the equality constraint $h(x) = 0$ of G03 is transformed into an approximate inequality $|h(x)| \leq \epsilon$ with $\epsilon = 0.0001$.

COBRA [31] comes close to SACOBRA in terms of efficiency (function evaluations), but it has to be noted that [31] does not present results for all G-problems (G01 and G11 are missing and G02 results are for 10 dimensions, but the commonly studied version of G02 has 20 dimensions). Furthermore, for many G-problems (G03, G06, G07, G09, G10) a manual transformations of the original fitness function or the constraint functions was done in [31] prior to optimization. SACOBRA starts without such transformations and proposes instead self-adjusting mechanisms to find suitable transformations (after the initialization phase or on-line).

COBYLA often produces slightly infeasible solutions, these are the numbers in brackets. If such infeasible runs occur, the median was only taken over the remaining feasible runs, which is in principle too optimistic in favor of COBYLA.

Fig. 11 shows the comparison of SACOBRA and COBRA-R with other well-known constraint optimization solvers available in R, namely DE[13] and COBYLA.[14] The right plot in Fig. 11 shows that DE achieves good results after many function evaluations, in accordance with Table 3. But the left plot in Fig. 11 shows that DE is not really competitive if very tight bounds on the budget are set.

Tab. 4 shows that SACOBRA greatly reduces the number of infeasible runs as compared to COBRA-R. Most of the SACOBRA variants have less than 2% infeasible runs whereas

[13]`R`-package `DEoptimR`, available from `https://cran.r-project.org/web/packages/DEoptimR`

[14]`R`-package `nloptr`, available from `https://cran.r-project.org/web/packages/nloptr`

Table 4: Number of infeasible runs among 330 runs returned by each method on the G-problem benchmark. A run is infeasible if the final best solution is infeasible.

method	infeasible runs	functions
SACOBRA	0	–
SACOBRA\ rescale	4	G05
SACOBRA\ RS	13	G03, G05, G07,G09,G10
SACOBRA\ aDRC	0	–
SACOBRA\ aFF	1	G10
SACOBRA\ aCF	0	–
COBRA-R(no rescale)	37	G03,G05,G07,G09,G10
COBRA-R(rescale)	23	G05,G07,G09,G10
COBYLA	48	G02,G03,G05,G06,G07,G09,G10
DE	0	–

Table 5: Comparing different algorithms on optimizing MOPTA08 after 1000 function evaluations.

Algorithm	best	median	mean	worst
COBRA-R [19]	226.3	227.0	227.3	229.5
TRB [33]	225.5	226.2	226.4	227.4
SACOBRA\RS	**222.4**	**223.1**	223.6	224.8
SACOBRA	223.0	223.3	**223.3**	**223.8**

COBRA-R has 7-11%. The full SACOBRA method has no infeasible runs at all.

3.6. MOPTA08

Fig. 12 shows that we get good results with SACOBRA on the high-dimensional MOPTA08 problem ($d = 124$) as well. A problem is said to be *solved* in the data profile of Fig. 12 if it is not more than $\tau = 0.4$ away from the best value obtained in all runs by all algorithms.

Table 5 shows the results after 1000 iterations for Regis' recent trust-region based approach TRB [33] and our algorithms. We can improve the already good mean best feasible results of 227.3 and 226.4 obtained with COBRA-R [19] and TRB [33], resp., to 223.3 with SACOBRA. The reason that SACOBRA\RS is slightly better than COBRA-R [19] is that SACOBRA uses an improved DRC.

4. Discussion

4.1. SACOBRA and surrogate modeling

SACOBRA is an algorithm capable of self-adjusting its parameters to a wide-ranging set of problems in constraint optimization. We analyzed the different elements of SACOBRA and their importance for efficient optimization on the G-problem benchmark. It turned out that the two most important elements are rescaling (especially in the early phase of optimization) and automatic fitness function adjustment (aFF, especially in the later phase

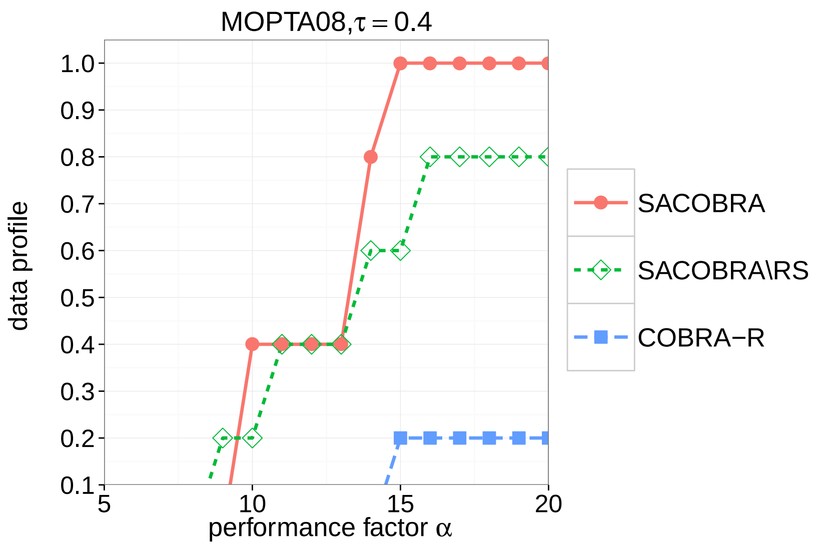


Figure 12: Data profile for MOPTA08: Same as Fig. 8 but with 10 runs on MOPTA08 with different initial designs. The curves for SACOBRA without rescale, aDRC, aFF, or aCF are identical to full SACOBRA, since in the case of MOPTA08 the objective function and the constraints are already normalized.

of optimization). Exclusion of either one of these two elements led to the largest performance drop in Fig. 8 compared to the full SACOBRA algorithm.

We may step back for a moment and ask why these two elements are important. Both of them are directly related to accurate RBF modeling, as our analysis in Sec. 2.3.1 has shown. If we do not rescale, then the RBF model for a problem like G10 will have large approximation errors due to numeric instabilities. If we do not perform the *plog*-transformation in problems like G03 with a very large fitness range FR (Tab. 1) and thus very steep regions, then such problems cannot be solved. This can be attributed to large RBF approximation errors as well.

We diagnosed that the quality of the surrogate models is in relationship with the correct choice of the DRC parameter, which controls the step size in each iteration. It is more desirable to choose a set of smaller step sizes for functions with steep slopes. An automatic adjustment step in SACOBRA can identify steep functions after a few function evaluations and decide whether to use a large DRC or a small one.

For the G-problem suite the constraint functions vary in number, type and range. Our experiments showed that handling all constraints can be challenging, especially when the constraint functions have widely different ranges. For that reason, we considered an automatic adjustment approach to normalize all the constraints by using the information gained about the constraints after the evaluation of the initial population. The SACOBRA algorithm also benefits from using a random start mechanism to avoid getting stuck in a local optimum of the fitness surrogate.

4.2. Limitations of SACOBRA

4.2.1. Highly multimodal functions

Surrogate models like RBF are a great thing for efficient optimization and probably the only way to solve constrained optimization problems in less than 500 iterations. But a current border for surrogate modeling are highly multimodal functions. G02 is such a function, it has a large number of local minima. Those functions have usually large first and higher order derivatives. If a surrogate model interpolates isolated points of such a function, it tends to overshoot in other parts of the function. To the best of our knowledge, highly multimodal problems cannot be solved so far by surrogate models, at least not for higher dimensions with high accuracy. This is also true for SACOBRA. Usually the RBF model has a good approximation only in the region of one of the local minima and a bad approximation in the rest of the search space. Further research on highly multimodal function approximation is required to solve this problem.

4.2.2. Equality constraints

The current approach in COBRA (and in SACOBRA as well) can only handle inequality constraints. The reason is that equality constraints do not work together with the uncertainty mechanism of Sec. 2.4. A reformulation of an equality constraint $h(x) = 0$ as inequality $|h(x)| \leq 0.0001$ as in [35] is not well-suited for COBRA and for RBF modeling. We used in this work the same approach as Regis [31] and replaced each equality operator with an inequality operator of the appropriate direction. It has to be noted however, that such an approach contradicts a true black-box handling of constraints and that it is – although being viable for the problems G01-G11 – not viable for more complicated objective functions having their minima on both sides of equality constraints. In a forthcoming paper [1] we will address this problem separately.

5. Conclusion

We summarize our discussion by stating that a good understanding of the capabilities and limitations of RBF surrogate models – which is not often undertaken in the surrogate literature we are aware of – is an important prerequisite for efficient and effective constrained optimization.

The analysis of the errors and problems occurring initially for some G-problems in the COBRA algorithm have given us a better understanding of RBF models and led to the development of the enhancing elements in SACOBRA. By studying a widely varying set of problems we observed certain challenges when modeling very steep or relatively flat functions with RBF. This can result in large approximation errors. SACOBRA tackles this problem by making use of a conditional *plog*-transform for the objective function. We proposed a new online mechanism to let SACOBRA decide automatically when to use *plog* and when not.

Numerical issues to train RBF models can also occur in the case of a very large input space. A simple solution to this problem is to rescale the input space. Although many other optimizers recommend to rescale the input, this work has shown the reason behind it and the importance of it by evidence. Therefore, we can answer our first research question **(H1)** positively: Numerical instabilities can occur in RBF modeling, but it is possible to avoid them with the proper function transformations and search space adjustments.

SACOBRA benefits from all its extension elements introduced in Sec. 2.5. Each element boosts up the optimization performance on a subset of all problems without harming the optimization process on the other ones. As a result, the overall optimization performance on the whole set of problems is improved by 50% as compared to COBRA (with a fixed parameter set). About 90% of the tested problems can be solved efficiently by SACOBRA (Fig. 8). The answer to **(H2)** is: SACOBRA is capable to cope with many diverse challenges in constraint optimization. It is the main contribution of this paper to propose with SACOBRA the first surrogate-assisted constrained optimizer which solves efficiently the G-problem benchmark and requires **no parameter tuning** or manual function transformations. Finally, let us provide a result to **(H3)**: SACOBRA requires **less than 500 function evaluations** to solve 10 out of 11 G-problems (exception: G02) with similar accuracy as other state-of-the-art algorithms. Those other algorithms often need between 300 and 1000 times more function evaluations.

Our future research will be devoted to overcome the current limitations of SACOBRA mentioned in Sec. 4.2. These are: (a) highly multimodal functions like G02 and (b) equality constraints.

Acknowledgements

This work has been supported by the Bundesministerium für Wirtschaft (BMWi) under the ZIM grant MONREP (AiF FKZ KF3145102, Zentrales Innovationsprogramm Mittelstand).

Supported by:

Federal Ministry
for Economic Affairs
and Energy

on the basis of a decision
by the German Bundestag

References

[1] S. Bagheri, T. Bäck, and W. Konen. Equality constraint handling for surrogate-assisted constrained optimization. In *WCCI'2016*, Vancouver, Canada, in preparation, 2016.

[2] J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Zumer. Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. *Evolutionary Computation, IEEE Transactions on*, 10(6):646–657, 2006.

[3] D. Brockhoff, T.-D. Tran, and N. Hansen. Benchmarking numerical multiobjective optimizers revisited. In *Genetic and Evolutionary Computation Conference (GECCO 2015)*, Madrid, Spain, July 2015.

[4] M. D. Buhmann. *Radial Basis Functions: Theory and Implementations.* Cambridge University Press, 2003.

[5] P. Chootinan and A. Chen. Constraint handling in genetic algorithms using a gradient-based repair method. *Computers & Operations Research*, 33(8):2263–2281, 2006.

[6] C. A. Coello Coello. Use of a self-adaptive penalty approach for engineering optimization problems. *Computers in Industry*, 41(2):113–127, 2000.

[7] C. A. Coello Coello. Constraint-handling techniques used with evolutionary algorithms. In *Proc. 14th International Conference on Genetic and Evolutionary Computation Conference (GECCO)*, pages 849–872. ACM, 2012.

[8] C. A. Coello Coello and E. M. Montes. Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. *Advanced Engineering Informatics*, 16(3):193–203, 2002.

[9] K. Deb. An efficient constraint handling method for genetic algorithms. *Computer methods in applied mechanics and engineering*, 186(2):311–338, 2000.

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

[11] A. E. Eiben and J. E. Smith. *Introduction to evolutionary computing*. Springer, 2003.

[12] M. Emmerich, K. C. Giannakoglou, and B. Naujoks. Single- and multiobjective evolutionary optimization assisted by gaussian random field metamodels. *Evolutionary Computation, IEEE Transactions on*, 10(4):421–439, 2006.

[13] R. Farmani and J. Wright. Self-adaptive fitness formulation for constrained optimization. *Evolutionary Computation, IEEE Transactions on*, 7(5):445–455, 2003.

[14] C. A. Floudas and P. M. Pardalos. *A Collection of Test Problems for Constrained Global Optimization Algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[15] R. Hooke and T. Jeeves. Direct search solution of numerical and statistical problems. *Journal of the ACM (JACM)*, 8(2):212–229, 1961.

[16] L. Jiao, L. Li, R. Shang, F. Liu, and R. Stolkin. A novel selection evolutionary strategy for constrained optimization. *Information Sciences*, 239:122 – 141, 2013.

[17] D. R. Jones. Large-scale multi-disciplinary mass optimization in the auto industry. In *Conference on Modeling And Optimization: Theory And Applications (MOPTA), Ontario, Canada*, pages 1–58, 2008.

[18] P. Koch, S. Bagheri, W. Konen, C. Foussette, P. Krause, and T. Bäck. Constrained optimization with a limited number of function evaluations. In F. Hoffmann and E. Hüllermeier, editors, *Proc. 24. Workshop Computational Intelligence*, pages 119–134. Universitätsverlag Karlsruhe, 2014.

[19] P. Koch, S. Bagheri, W. Konen, C. Foussette, P. Krause, and T. Bäck. A new repair method for constrained optimization. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference (GECCO)*, pages 273–280. ACM, 2015.

[20] O. Kramer. A review of constraint-handling techniques for evolution strategies. *Applied Computational Intelligence and Soft Computing*, 2010:1–11, 2010.

[21] O. Kramer and H.-P. Schwefel. On three new approaches to handle constraints within evolution strategies. *Natural Computing*, 5(4):363–385, 2006.

[22] Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[23] E. M. Montes and C. A. Coello Coello. A simple multimembered evolution strategy to solve constrained optimization problems. *IEEE Transactions on Evolutionary Computation*, 9(1):1–17, 2005.

[24] E. M. Montes and C. A. Coello Coello. Constraint handling in nature-inspired numerical optimization: past, present and future. *Swarm and Evolutionary Computation*, 1(4):173–194, 2011.

[25] J. J. Moré and S. M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM J. Optimization*, 20(1):172–191, 2009.

[26] J. Poloczek and O. Kramer. Local SVM constraint surrogate models for self-adaptive evolution strategies. In *KI 2013: Advances in Artificial Intelligence*, pages 164–175. Springer, 2013.

[27] M. J. D. Powell. The theory of radial basis function approximation in 1990. *Advances In Numerical Analysis*, 2:105–210, 1992.

[28] M. J. D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances In Optimization And Numerical Analysis*, pages 51–67. Springer, 1994.

[29] A. K. Qin and P. N. Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In *IEEE Congress on Evolutionary Computation (CEC), 2005*, volume 2, pages 1785–1791. IEEE, 2005.

[30] R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2013.

[31] R. G. Regis. Constrained optimization by radial basis function interpolation for high-dimensional expensive black-box problems with infeasible initial points. *Engineering Optimization*, 46(2):218–243, 2014.

[32] R. G. Regis. Particle swarm with radial basis function surrogates for expensive black-box optimization. *Journal of Computational Science*, 5(1):12–23, 2014.

[33] R. G. Regis. Trust regions in surrogate-assisted evolutionary programming for constrained expensive black-box optimization. In R. Datta and K. Deb, editors, *Evolutionary Constrained Optimization*, pages 51–94. Springer, 2015.

[34] R. G. Regis and C. A. Shoemaker. A quasi-multistart framework for global optimization of expensive functions using response surface models. *Journal of Global Optimization*, 1:1, 2012.

[35] T. P. Runarsson and X. Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, 2000.

[36] T. P. Runarsson and X. Yao. Search biases in constrained evolutionary optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 35(2):233–243, 2005.

[37] Y. Tenne and S. W. Armfield. A memetic algorithm assisted by an adaptive topology RBF network and variable local models for expensive optimization problems. In W. Kosinski, editor, *Advances in Evolutionary Algorithms*, page 468. INTECH Open Access Publisher, 2008.

[38] B. Tessema and G. G. Yen. An adaptive penalty formulation for constrained evolutionary optimization. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 39(3):565–578, 2009.

[39] S. Venkatraman and G. G. Yen. A generic framework for constrained optimization using genetic algorithms. *Evolutionary Computation, IEEE Transactions on*, 9(4):424–435, Aug 2005.

[40] E. Zahara and Y.-T. Kao. Hybrid Nelder–Mead simplex search and particle swarm optimization for constrained engineering design problems. *Expert Systems with Applications*, 36(2):3880–3886, 2009.

[41] H. Zhang and G. Rangaiah. An efficient constraint handling method with integrated differential evolution for numerical and engineering optimization. *Computers & Chemical Engineering*, 37:74 – 88, 2012.

Vitae

Samineh Bagheri received her B.Sc. degree in electrical engineering specialized in electronics from Shahid Beheshti University, Tehran, Iran in 2011. She received her M.Sc. in Industrial Automation & IT from Cologne University of Applied Sciences, where she is currently research assistant and Ph.D. student in cooperation with Leiden University. Her research interests are machine learning, evolutionary computation and constrained and multiobjective optimization tasks.

Wolfgang Konen is Professor of Computer Science and Mathematics at Cologne University of Applied Sciences, Germany. He received his Diploma in physics and his Ph.D. degree in theoretical physics from the University of Mainz, Germany, in 1987 and 1990, resp. He worked in the area of neuroinformatics and computer vision at Ruhr-University Bochum, Germany, and in several companies. He is founding member of the Research Centers Computational Intelligence, Optimization & Data Mining (`http://www.gociop.de`) and CIplus (`http://ciplus-research.de`). He co-authored more than 100 papers and his research interests include, but are not limited to: efficient optimization, neuroevolution, machine learning, data mining, and computer vision.

Dr. Michael Emmerich is Assistant Professor at LIACS, Leiden University, and leader of the Multicriteria Optimization and Decision Analysis research group. He received his doctorate in 2005 from Dortmund University (H.-P. Schwefel, promoter) and worked as a researcher at ICD e.V. (Germany), IST Lisbon, University of the Algarve (Portugal), ACCESS Material Science e.V. (Germany), and the FOM/AMOLF institute (Netherlands). He is known for pioneering work on model-assisted and indicator-based multiobjective optimization, and has co-authored more than 100 papers in machine learning, multicriteria optimization and surrogate-assisted optimization and its applications in chemoinformatics and engineering optimization.

Thomas Bäck is head of the Natural Computing Research Group at the Leiden Institute of Advanced Computer Science (LIACS). He received his PhD in Computer Science from Dortmund University, Germany, in 1994. He has been Associate Professor of

Computer Science at Leiden University since 1996 and full Professor for Natural Computing since 2002. Thomas Bäck has more than 150 publications on natural computing technologies. His main research interests are theory and applications of evolutionary algorithms (adaptive optimization methods gleaned from the model of organic evolution), cellular automata, data-driven modeling and applications of those methods in medicinal chemistry, pharmacology, and engineering.