Self-adjusting surrogate-assisted optimization techniques for expensive constrained black box problems

Bagheri, S.

Chapter 3

SACOBRA: Self-Adjusting Parameter Control

3.1 Outline

Constrained optimization of high-dimensional numerical problems plays an important role in many scientific and industrial applications. The number of function evaluations in many industrial applications are severely limited and often only little analytical information about the objective function and constraint functions is available. For such expensive black box optimization tasks, the constrained optimization algorithm COBRA (**C**onstrained **O**ptimization **B**y **R**adial Basis Function **A**pproximation) was proposed, making use of RBF (radial basis function) surrogate modeling for both objective and constraint functions [141]. COBRA and its extended version in R, the so-called COBRA-R [97], have shown remarkable success in solving reliably complex benchmark problems in less than 500 function evaluations. Unfortunately, COBRA-R requires careful adjustment of its parameters in order to do so. To the best of our knowledge there is no algorithm available in the state-of-the art that solves a diverse set of COPs both efficiently (in less than 1000 iterations) and without parameter adjustment.

In this chapter we present a new algorithm, SACOBRA (Self-Adjusting COBRA), which is an extension of COBRA-R and capable of achieving 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 listed in Appendix A and MOPTA08 explained in Sec. 3.6) that SACOBRA consistently outperforms COBRA algorithms with different fixed parameter settings. We analyze the importance of the new elements in SACOBRA and show that each element of SACOBRA plays a role to boost up the overall optimization performance. In order to keep this chapter as similar as possible to its relevant published work [19] we only take the first 11 G-problems with inequality constraints into account. Therefore, the

3 G-problems with equality constraints among the first 11 problems are modified to COPs with inequality constraints: G03mod, G05mod and G11mod. The other G-problems with equality constraints are covered in Ch. 4. The analysis in this chapter is based on the work of Bagheri et al. [19].

The rest of this chapter is organized as follows: Sec. 3.2 formulates the main motivations of this chapter and poses several research questions. The related work will be discussed in Sec. 3.3. In Sec. 3.5 we present common pitfalls in surrogate modeling in Sec. 3.4. We describe the COBRA and SACOBRA algorithms in Sec. 3.5. In Sec. 3.6, we perform a thorough experimental study on analytical test functions and on MOPTA08 [92], which represents a real-world benchmark function from the automotive domain. With the help of so-called data profiles, we analyze the impact of the various SACOBRA elements on the overall performance. The results are discussed in Sec. 3.7 and we give conclusive remarks in Sec. 3.8.

3.2 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 turns out to be even 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 [92].

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 (see, e. g., results in [150, 101]).

Up till now, only little work has been devoted to *efficient* constraint optimization (i. e., using a 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 was proposed by Regis [141] and outperforms many other algorithms on a large number of benchmark functions.

Koch et al. [97, 98] have studied a reimplementation of COBRA in R [136], 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 chapter 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 an example of *adaptive parameter control* according to the terminology introduced by Eiben et al. [53]. We present extensive tests of SACOBRA and other algorithms on a well-known benchmark from the literature: The so-called G-problem or G-function benchmark, which was initially introduced by Michalewicz and Schoenauer [114], Floudas and Pardalos [60] and later extended by other authors [150, 107], provides a set of constrained optimization problems with a wide range of different conditions (Appendix A). We define the following research questions for the constrained optimization experiments in this work:

Q3.1 Do numerical instabilities occur in RBF surrogates and is it possible to avoid them?
Stable models are vital for the success of surrogate-assisted optimization algorithms. It is important to detect reasons of possible instabilities in RBF surrogates which are used in SACOBRA and also come up with efficient solutions.

Q3.2 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?
It is difficult to find any optimizer which can handle all G-problems with one parameter configuration due to the diversity of the G-problems in properties. We aim to handle all of G-problems by means of a self-adjusting parameter control embedded in a surrogate-assisted optimizer. In this chapter we investigate the effectiveness of this method.

Q3.3 Is it possible with SACOBRA to solve all G-problems in a given, small number of function evaluations (e. g., 1000) ?
Surrogate-assisted optimizers are mainly used to reduce the required number of function evaluations for time and cost expensive optimization problems. The effectiveness of SACOBRA is not only measured by its accuracy but also by its efficiency.

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

3.3 Related Work

Most optimization algorithms need their parameters to be set with respect to the specific optimization problem in order to show good performance. Eiben et al. [53] 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 [134] propose a self-adaptive differential evolution (DE) algorithm. Brest et al. [31] propose another self-adaptive DE algorithm. But they do not handle constraints, whereas Zhang et al. [187] describe a constraint-handling mechanism for DE. We will later compare our results with the DE-implementation `DEoptimR`[2] which is based on both works [31, 187]. Farmani and Wright [56] propose a self-adaptive fitness formulation and test it on 11 G-problems. They show comparable results to stochastic ranking [150], but require many function evaluations (above 300 000) as well. Coello Coello [39], Eiben and van Hemert [52] and Tessema and Yen [173] propose self-adaptive penalty approaches. A survey of self-adaptive penalty approaches is given in [53].

The area of **efficient constrained optimization**, that is optimization under a severely limited budget of less than 1 000 function evaluations, is attracting more and more attention in recent years: Regis proposed besides the already mentioned COBRA approach [141] a trust-region evolutionary algorithm [142] which uses RBF surrogates as well and which exhibits high-quality results on many but not all G-functions in less than 1 000 function evaluations. Although many Kriging-assisted efficient optimizers have been developed in the last years [8, 153, 51], these algorithms often do not perform well on the G-problems due to the limitations of Kriging with respect to high dimensional problems. Jiao et al. [88] propose a self-adaptive selection method to combine special feasible and infeasible solutions and they formulate it as a multi-objective problem. Their algorithm can solve some of the G-functions (G08, G11) very fast in less than 500 evaluations, some others are solved in 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 [186] show similar results (1 000 – 20 000 evaluations) on some G-functions, but they investigate only G04 and G08. To the best of our knowledge there is currently no black box constrained optimization

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

approach which can solve all 11 G-problems in less than 1 000 evaluations. Tenne and Armfield [172] 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.

3.4 Pitfalls in Surrogate-Assisted Optimization

The RBF models described in Sec. 2.4.2 are very fast to train, also for high dimensional search spaces. 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. SACOBRA which will be described in detail in Sec. 3.5, uses cubic RBF with polynomial tail as surrogate. However, there are some pitfalls which should be avoided in order to achieve good modeling results for any surrogate-assisted black box optimization. These are introduced and discussed in the next sections.

3.4.1 Rescaling the Input Space

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

$$f(x) = 3\frac{x}{S} + 1 \tag{3.1}$$

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. (3.1) is exactly the same. Since the function $f(x)$ to be modeled is exactly linear and the augmented RBF-model we often use (as described in Sec. 2.4.2), contains a linear tail as well, one would expect at first sight a perfect fit for each surrogate model. But – as Fig. 3.1 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 is as follows: 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 (root mean square error), as it is demonstrated in the right plot of Fig. 3.1. 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

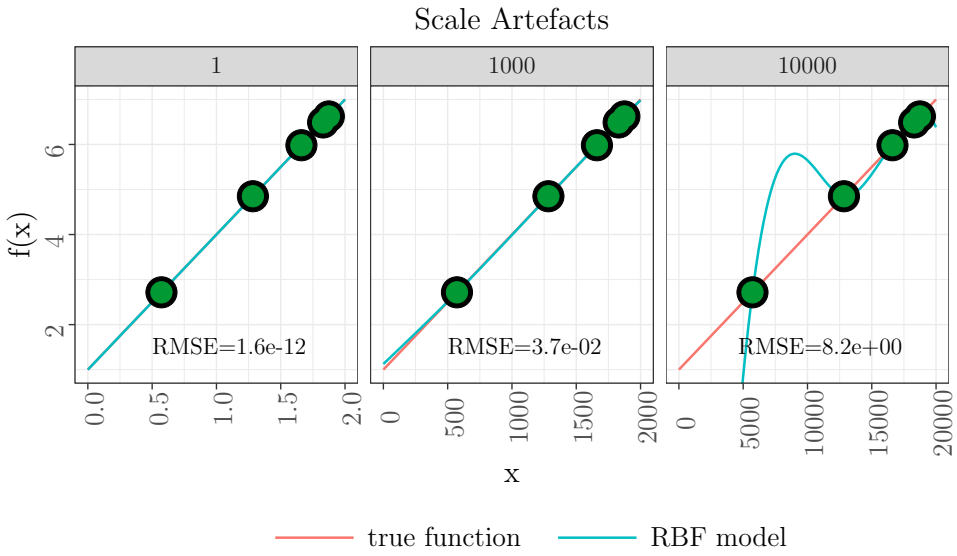

Figure 3.1: The influence of scaling. From left to right the plots show the RBF model fit (cubic RBF with polynomial tail, cf. Eq. (7.13)) for scale $S = 1, 1\,000, 10\,000$ (upper facet bar). RMSE: root mean square error.

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.

3.4.2 Logarithmic Transform for Large Output Ranges

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

$$f(x) = e^{x^2} \tag{3.2}$$

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 a cubic RBF model using the green sample points shown in Fig. 3.2, we see in the left plot an oscillating behavior in the RBF function. This results in a large RMSE

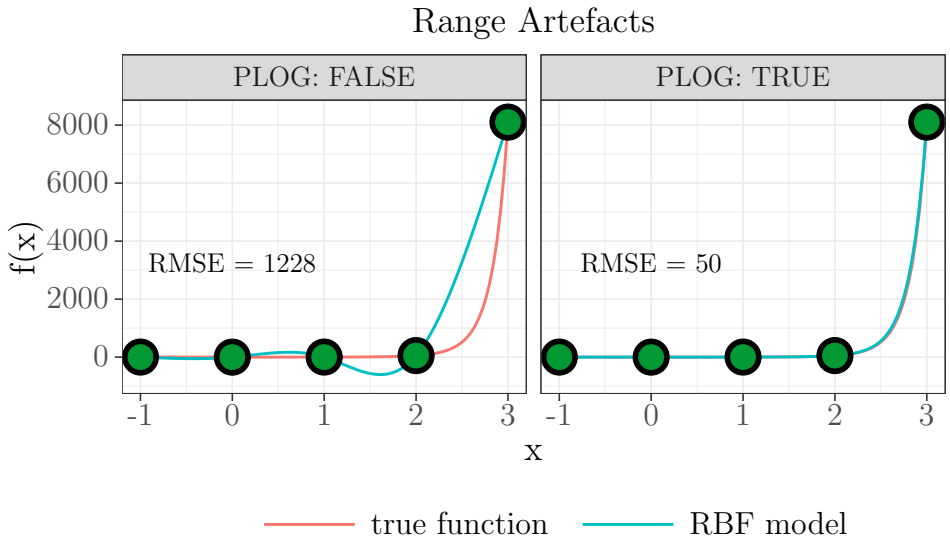

Figure 3.2: The influence of large output ranges. Left: Fitting the original function with a cubic RBF model. Right: Fitting the *plog*-transformed function with an RBF model and transforming the fit back to original space with $plog^{-1}$.

(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 [145] define the function

$$plog(y) = \begin{cases} +\ln(1+y), & \text{if} \quad y \geq 0, \\ -\ln(1-y), & \text{if} \quad y < 0, \end{cases} \tag{3.3}$$

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. 3.2 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.

3.5 Methods

A constrained optimization problem can be defined by the minimization of an objective function f subject to equality and inequality constraint functions as defined in Eq. (4.1). In this chapter we only consider minimization problems with inequality constraints:

$$\begin{array}{lll} \text{Minimize} & f(\vec{x}), & \vec{x} \in [\vec{l}, \vec{u}] \subset \mathbb{R}^d \\ \text{subject to} & g_j(\vec{x}) \leq 0, & j = 1, 2, \ldots, m, \end{array} \tag{3.4}$$

where $\vec{l}$ is the lower bound of the search space $\mathbb{S} \subseteq \mathbb{R}^d$ and the $\vec{u}$ is the upper bound. Maximization problems can be transformed to minimization without loss of generality.

3.5.1 COBRA

The COBRA algorithm has been developed by Regis [141] with the aim of solving 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 $g_j(.)$). This algorithm was reimplemented in `R` [136] with a few modifications [97, 98]. A short review of this algorithm is given in the following.

COBRA starts by generating an initial population P with n_{init} points (i. e. a random initial design[3], see Fig. 3.3) 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, where d is the dimension of the problem.

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 algorithm solves with any

[3]Usually a Latin hypercube sampling (LHS).

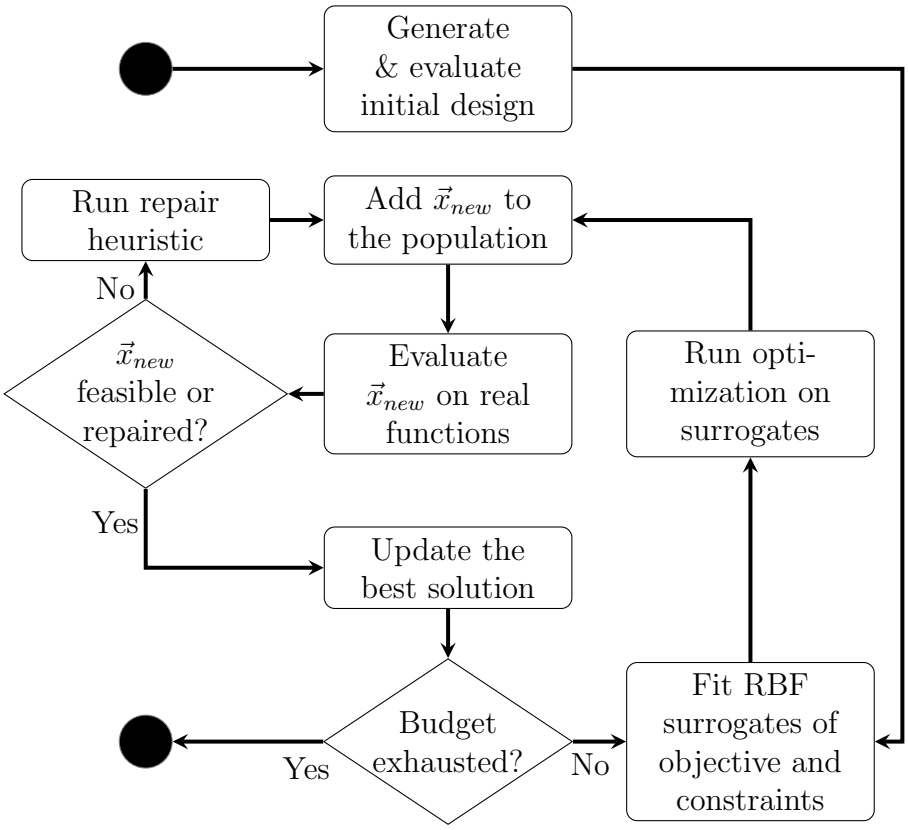

Figure 3.3: COBRA flowchart.

standard constrained optimizer[4] the constrained surrogate subproblem

$$\begin{aligned} \text{Minimize} \quad & s_0^{(n)}(\vec{x}) \\ \text{subject to} \quad & \vec{x} \in [\vec{l}, \vec{u}] \subset \mathbb{R}^d, \\ & s_j^{(n)}(\vec{x}) + \delta^{(n)} \leq 0, \qquad j = 1, 2, \ldots, m \\ & \rho^{(n)} - ||\vec{x} - \vec{x}_p|| \leq 0, \qquad p = 1, \ldots, n. \end{aligned} \tag{3.5}$$

[4]Regis [141] uses MATLAB's FMINCON, an interior-point optimizer, which is not available in the `R` environment. In our COBRA implementation 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.

Compared to the original problem in Eq. (3.4) this subproblem uses surrogates and it contains two new elements $\delta^{(n)}$ and $\rho^{(n)}$ which are explained in the next subsections. Before going into these details let us finish the description of the main loop: The optimizer returns a new solution $\vec{x}_{new} = \vec{x}_{n+1}$. If $\vec{x}_{n+1}$ is not feasible, a repair algorithm RI2, described in [98], 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 it is 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 n incremented by one.

Distance Requirement Cycle

COBRA [141] 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 already visited points. 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 cyclically the next element $\rho_n = \xi^{(i)}$ of Ξ and adds the constraints $||\vec{x} - \vec{x}_j|| \geq \rho_n$, $j = 1, ..., n$, to the set of constraints. This measures the distance between the proposed infill solution and all n previous infill points. The so-called distance requirement cycle (DRC) Ξ is a key idea of COBRA, 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 individual components can be chosen arbitrarily.

Uncertainty of Constraint Predictions

COBRA [141] 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[6] $\delta^{(n)}$ is used to handle wrong predictions of the constraint surrogates. Starting with $\delta_{init} = 0.005 \cdot l$, where l is the length of the smallest side of the search space, a point $\vec{x}$ is said to be *feasible in iteration n* if

$$s_j^{(n)}(\vec{x}) + \delta^{(n)} \leq 0 \quad \forall \quad j = 1, \ldots, m \tag{3.6}$$

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

[6] The parameter used as a margin of uncertainty is originally called $\epsilon^{(n)}$ in [141] but here we use $\delta^{(n)}$ to avoid confusion with equality margin parameter in Ch. 4.

Table 3.1: Adaptive control elements of the SACOBRA algorithm that were in previous work on the COBRA algorithm either manually adjusted for each problem or not present at all. In contrast to this, SACOBRA always starts with the same settings and adjusts the elements either *automatically* (once at the start) to the problem at hand or *adaptively* (specific to the problem and changable during iterations).

Element	[Regis14] [141]	[Koch14,15] [97, 98]	SACOBRA [this work]
Input rescaling	always	never	always
Constraint normalization	manually	manually	automatic (Eq. (3.7))
DRC adjustment	manually	manually	automatic (acc. to $\widehat{FR}$)
Random start probability	never	never	adaptive (acc. to feasibility rate)
Objective transform *plog*	manually	manually	adaptive (Q-value, Eq. (3.8))

holds. That is, we tighten the constraints by adding the factor $\delta^{(n)}$ which is adapted during the search. The $\delta^{(n)}$-adaptation is done by counting the feasible and infeasible infill points C_{feas} and C_{infeas} over the last iterations. When these counters reach the threshold for feasible or infeasible solutions, T_{feas} or T_{infeas}, respectively, we divide or multiply $\delta^{(n)}$ by 2 (up to a given maximum δ_{max}). When $\delta^{(n)}$ 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, $\delta^{(n)}$ is increased in order to keep the points further away from the real constraint boundary.

Repair Heuristic (RI2)

Sometimes the infill points returned by the internal optimizer are infeasible. A repair algorithm is embedded in the COBRA-R optimization framework which intends to repair infill points with a slight infeasibility by guiding them to the feasible region. The repair algorithm RI-2 used in COBRA-R is described and discussed in detail by Koch et al. [98]. It is worthwhile to mention that the repair algorithm is performed on the surrogate models, so no real function evaluations are necessary for this repair.

3.5.2 SACOBRA

COBRA achieves good results on most of the G-problems and on MOPTA08 as studies from Regis [141] and also studies from Koch et al. [97, 98] have shown. However, it was necessary in all the mentioned works [141, 97, 98] to carefully adjust

Algorithm 1 SACOBRA. **Input:** Objective function f, set of constraint function(s) $\mathbf{g} = (g_1, \ldots, g_m) : [\vec{a}, \vec{b}] \subset \mathbb{R}^d \to \mathbb{R}$ (see Eq. (4.1)), 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.

function SACOBRA($f, \mathbf{g}, \vec{x}_{init}, N_{max}$)
  Rescale the input space to $[-1, 1]^d$
  Generate a random initial population: $P \leftarrow \{\vec{x}_1, \vec{x}_2, \cdots, \vec{x}_{3 \cdot d}\}$
  $(\widehat{FR}, \widehat{GR}_i) \leftarrow$ AnalyseInitialPopulation($P, f, \mathbf{g}$)
  $\mathbf{g} \leftarrow$ AdjustConstraintFunctions($\widehat{GR}_i$, $\mathbf{g}$)
  $\Xi \leftarrow$ AdjustDRC($\widehat{FR}$)
  $Q \leftarrow$ AnalysePlogEffect($f, P, \vec{x}_{init}$)
  $\vec{x}_{best} \leftarrow \vec{x}_{init}$
  while (budget not exhausted, $|P| < N_{max}$) **do**
    $n \leftarrow |P|$
    if ($Q > 1$) **then**
      $f() \leftarrow plog(f())$ ▷ see function *plog* in Eq. (3.3)
    end if
    Build surrogate models $\vec{s}^{(n)}$=$(s_0^{(n)}, s_1^{(n)}, \cdots, s_m^{(n)})$ for $(f, g_1, \cdots, g_m)$
    Select $\rho_n \in \Xi$ and $\delta_i^{(n)}$ according to COBRA base algorithm
    $\vec{x}_{start} \leftarrow$ RandomStart($\vec{x}_{best}, N_{max}$)
    $\vec{x}_{new} \leftarrow$ OptimCOBRA($\vec{x}_{start}, \vec{s}^{(n)}$) ▷ see Eq. (3.5)
    Evaluate $\vec{x}_{new}$ on the real functions $f, \mathbf{g}$
    if ($|P| \mod 10 == 0$) **then** ▷ every 10th iteration
      $Q \leftarrow$ AnalysePlogEffect($f, P, \vec{x}_{new}$)
    end if
    $\vec{x}_{new} \leftarrow$ repairRI2($\vec{x}_{new}$) ▷ see Koch et al. [98] for details on RI2 (repair algo)
    $(P, \vec{x}_{best}) \leftarrow$ updateBest($P, \vec{x}_{new}, \vec{x}_{best}$)
  end while
  return $\vec{x}_{best}$
end function

function updateBest($P, \vec{x}_{new}, \vec{x}_{best}$)
  $P \leftarrow P \cup \{\vec{x}_{new}\}$
  if ($\vec{x}_{new}$ is feasible AND $\vec{x}_{new} < \vec{x}_{best}$) **then**
    return $(P, \vec{x}_{new})$
  end if
  return $(P, \vec{x}_{best})$
end function

Algorithm 2 SACOBRA adjustment functions

function AnalyseInitialPopulation($P, f, \mathbf{g}$)
  $\widehat{FR} \leftarrow \max_P f(P) - \min_P f(P)$ ▷ range of objective function
  $\widehat{GR_i} \leftarrow \max_P g_i(P) - \min_P g_i(P) \quad \forall i = 1, \ldots, m$
end function

function AdjustConstraintFunction($\widehat{GR_i}$,$\mathbf{g}$)
  $g_i() \leftarrow g_i() \cdot \frac{\text{avg}(\widehat{GR_i})}{\widehat{GR_i}} \quad \forall i = 1, \ldots, m$ ▷ see Eq. (3.7)
  return g
end function

function AdjustDRC($\widehat{FR}$)
  if $\widehat{FR} > FR_l$ **then** ▷ Threshold $FR_l = 1000$
    $\Xi \leftarrow \Xi_s \leftarrow \langle 0.001, 0.0 \rangle$
  else
    $\Xi \leftarrow \Xi_l \leftarrow \langle 0.3, 0.05, 0.001, 0.0005, 0.0 \rangle$
  end if
end function

function AnalysePlogEffect($f, P, \vec{x}_{new}$) ▷ $\vec{x}_{new} \notin P$
  $S_f \leftarrow$ surrogate model for $f(.)$ using all points in P
  $S_p \leftarrow$ surrogate model for $plog(f(.))$ using all points in P ▷ see Eq. (3.3)
  $E \leftarrow E \cup \left\{ \frac{|S_f(\vec{x}_{new}) - f(\vec{x}_{new})|}{|plog^{-1}(S_p(\vec{x}_{new})) - f(\vec{x}_{new})|} \right\}$ ▷ E, the set of approximation error ratios, is initially empty
  return $Q = \log_{10}(\text{median}(E))$
end function

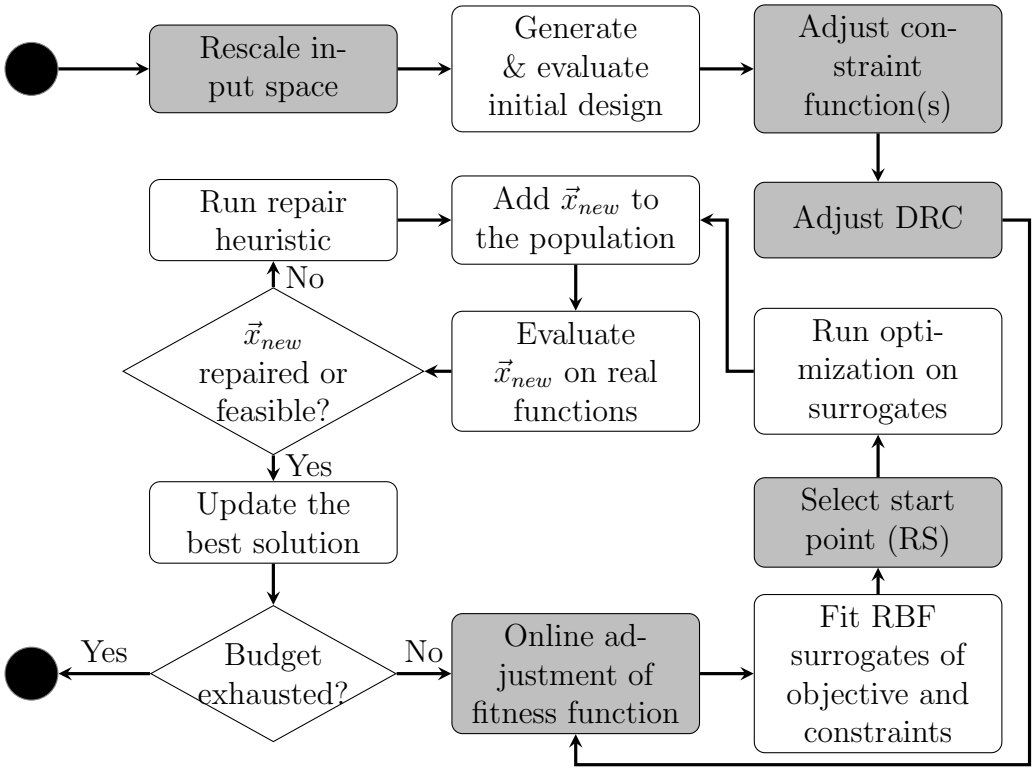

Figure 3.4: SACOBRA flowchart.

the parameters of the algorithm to each problem. Sometimes it was even required to modify the problem (a) by applying a *plog*-transform (Eq. (3.3)) to the objective function or linear transformations to the constraints or (b) 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.

The main contribution of the work of Bagheri et al. [19] which is described in this chapter is to present with SACOBRA an enhanced COBRA algorithm which has no needs for manual adjustment to the problem at hand. Instead, SACOBRA during its execution extracts information about the specific problem (either after initialization or online during iterations) and takes appropriate internal measures to adjust its parameters or to transform functions. Table 3.1 lists the elements which were manually adjusted in the former works [97, 98, 141] and which are now in SACOBRA under adaptive or automatic control. These elements will be described in more detail in the following subsections.

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( x_best)
2:     if (|P_feas|/|P| < 0.05) then      ▷ if less than 5% of the population are feasible
3:         p ← p_2
4:     else
5:         p ← p_1
6:     end if
7:     ε ← a random value ∈ [0, 1]
8:     if (ε < p) then
9:         x_start ← a random point in search space
10:    else
11:        x_start ← x_best
12:    end if
13:    return (x_start)
14: end function
```

Fig. 3.4 shows the flowchart of SACOBRA where the five new elements compared to COBRA are highlighted as gray boxes. The complete SACOBRA algorithm is presented in detail in Algorithm 1 – 3. In the following we describe the five new elements in the order of their appearance:

Rescaling the Input Space

The search space $[\vec{l}, \vec{u}]$ in Eq. (3.4) is rescaled to $[-1, +1]^d$. That is, the function $f(\vec{x})$ is replaced with function $f(\vec{k}(\vec{x}))$ where $\vec{k}(\vec{x})$ rescales from $\vec{x} \in [-1, +1]^d$ to $[\vec{l}, \vec{u}]$. The same rescaling occurs for constraint functions $g_j(\vec{x})$. This rescaling is done before the initialization phase. It avoids numerical instabilities caused by high values of $\vec{x}$ and ill-conditioning as shown in Sec. 3.4.1.

Adjusting Constraint Functions (aCF)

The function AdjustConstraintFunction in Algorithm 2 aims to normalize the constraint functions in such a way that they have equal importance for the optimizing algorithm. Firstly, each constraint is divided by $\widehat{GR}_i$. The range $\widehat{GR}_i$ for the ith constraint is estimated from the initial population in Algorithm 2. This transforms

the range of each constraint approximately to an interval of length one around the zero point, of course without shifting the zero point, since this defines the boundary between feasible and infeasible region. Secondly, every constraint is multiplied by the average constraint range,

$$\text{avg}\left(\widehat{GR}_i\right) = \frac{1}{m}\sum_{i=1}^{m}\widehat{GR}_i, \tag{3.7}$$

in order to keep the balance between objective and constraint functions. To understand this second point, consider the following example: Assume an objective function with $FR = 1000$ and two constraints with the ranges $[-1000, 1000]$ and $[-800, 800]$. After the first normalization step both constraints are in the range $[-0.5, 0.5]$. The optimizer is in danger to pay little attention to the constraints since their values are much smaller than the objective value. Multiplying by $\text{avg}\left(\widehat{GR}_i\right)$ brings both constraints to the range $[-900, 900]$ and thus reconstitutes approximately the relative balance between constraints and objective.

Adjusting DRC Parameter (aDRC)

DRC adjustment (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 larger move in the input space yields a very large change in the output space. As shown in Sec. 3.4.2 already, the combination of points with large change in output space and points with small change may result in oscillating behavior of the RBF model. This leads 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 AdjustDRC 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.

Random Start Algorithm (RS)

Normally, COBRA starts its internal optimization from the current best point. With RS (Algorithm 3), the optimization starts with a certain constant probability p_1 from a random point in the search space. If the rate of feasible individuals in the population P drops below 5%, 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.

Online Adjustment of Fitness Function (aFF)

The analysis in Sec. 3.4.2 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, the tests have shown that the *plog*-transform is harmful for other problems. Therefore, a careful decision whether to use *plog* or not should be made. The idea of the 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 19 of Algorithm 2). We do this in every kth iteration (usually $k = 10$) and collect these ratios in a set E. If

$$Q = \log_{10}(\text{median}(E)) \tag{3.8}$$

is above 0, then the RBF for $plog(f)$ is better in the majority of the cases. Otherwise, the RBF on f is better.[7] Step 11 of Algorithm 1 decides on the basis of this criterion Q which function f is used as RBF surrogate in the optimization step. Note that the decision for 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 the SACOBRA algorithm.

3.6 Experimental Evaluation

3.6.1 Experimental Setup

We evaluate SACOBRA by using a subset of a well-studied test suite of G-problems described in [60, 114]. The diversity of the G-problem characteristics makes them a very challenging benchmark for optimization techniques (see Fig. A.1 in Appendix. A). In Tab. 3.2 we show and explain features of these problems. The features ρ, FR and GR (defined in Tab. 3.2) are measured by Monte Carlo sampling with 10^6 points in the search space of each G-problem.

For problems with equality constraints including G03, G05 and G11 the equality constraints are transformed to inequality constraints by replacing each equality operator with an inequality operator of the appropriate direction [141]. The appro-

[7] Our experimental analysis on the G-problem test suite will show (Sec. 3.6.4) 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.

Table 3.2: Characteristics of the G-functions with inequality constraints: d: dimension, type: type of fitness function, ρ: feasibility rate (%), FR: range of the fitness values, GR: ratio of largest to smallest constraint range, LI: number of linear inequalities, NI: number of nonlinear inequalities, a: number of constraints active at the optimum.

Fct.	d	type	ρ	FR	GR	LI	NI	a
G01	13	quadratic	0.0003%	298.14	1.969	9	0	6
G02	10	nonlinear	99.997%	0.57	2.632	1	1	1
G03mod	20	nonlinear	2.46e-6%	92684985979.23	1.000	0	1	1
G04	5	quadratic	26.9217%	9832.45	2.161	0	6	2
G05mod	4	nonlinear	0.0919%	8863.69	1788.74	2	3	3
G06	2	nonlinear	0.0072%	1246828.23	1.010	0	2	2
G07	10	quadratic	0.0000%	5928.19	12.671	3	5	6
G08	2	nonlinear	0.8751%	1821.61	2.393	0	2	0
G09	7	nonlinear	0.5207%	10013016.18	25.05	0	4	2
G10	8	linear	0.0008%	27610.89	3842702	3	3	6
G11mod	2	linear	66.7240%	4.99	1.000	0	1	1

priate direction is that direction which makes this side of the hypersurface infeasible that contains the unconstrained optimum. (The hypersurface is the set of all points where the constraint value is zero.) For suitable objective functions this forces the constrained optimum to be exactly on the hypersurface – the same as it would be for the equality constraint.[8] The modified problems G03mod, G05mod and G11mod are described in Appendix. A. In Ch. 4 we introduce an equality handling approach for SACOBRA.

The MOPTA08 benchmark by Jones [92] 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 about 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.

[8]This approach (which differs from the approach taken in the CEC 2006 competition [107]) will not work for every objective function. It will fail for objective functions with minima on both sides of the constraint hypersurface. See Sec. 3.7.2 for further discussion on this. But the approach works for the G-problems considered here.

Table 3.3: 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: problem dimension) are taken from [141].

parameter	value	
	COBRA	SACOBRA
δ_{init}	$0.005 \cdot l$	$0.005 \cdot l$
δ_{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

The COBRA-R optimization framework allows the user to choose between several initialization approaches: Latin hypercube sampling (LHS), Biased and Optimized [97]. 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 Regis' COBRA [141] 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 [83]. 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.

Tab. 3.3 shows the parameter settings used for COBRA and SACOBRA in the experiments reported here. All G-problems were optimized in SACOBRA with exactly the same initial parameter settings. In contrast to that, the COBRA results in Regis [141] and our previous work [97, 98] were obtained by manually activating *plog* for some G-problems and by manually adjusting constraint factors and other parameters. – We note in passing that SACOBRA has additionally about[9] 15 fixed parameters, some of which are shown in Tab. 3.3, and some additional parameters

[9]It depends a bit how these parameters are counted, e. g. whether the 0.05 in Algorithm 3 is counted as a fixed parameter or not.

like n_{init}, p_1 and p_2 are mentioned in the text. However, these parameters are kept constant for all experiments shown below.

3.6.2 Convergence curves

Figures 3.5 – 3.7 show the SACOBRA convergence plots for all G-problems. The red square is the result reported by Regis [141] after 100 iterations. If no red square is shown, this function was not covered in [141]. The blue horizontal lines show two different success thresholds. The solid blue line shows the success threshold $\tau = 0.05$ considered here and the dashed blue line shows the success threshold $\tau_{CEC} = 0.0001$ suggested in CEC 2006 [107]. 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 (G03mod, G05mod, 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 [141].

3.6.3 Performance profiles

The main result is shown in Fig. 3.8 which analyzes the impact of different elements of SACOBRA on the G-problems. It shows the data profiles for different SACOBRA variants in comparison with the data profile for COBRA-R. COBRA-R is the COBRA implementation from [97], i. e. SACOBRA with all 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). COBRA-R was run with a fixed parameter set for all G-problems.[10] We note in passing that many fixed parameter settings were tested for COBRA from which the one with the overall best results is reported. Other fixed parameter settings were perhaps better on some of the runs but inevitably worse on other runs. In the end a similar or slightly worse data profile for COBRA would emerge. We cannot be absolutely sure that there might be another parameter setting with better results, but the probability for such an event is from our experience pretty low. SACOBRA increases significantly the success rate on the G-problem benchmark suite.

In addition, in Fig. 3.8 the effect of the five elements of SACOBRA is analyzed: The data profiles with a „\“present the SACOBRA results when one specific of the five SACOBRA elements is switched off. We see that the strongest effects occur

[10] In the previous work [97, 98] good results with COBRA are reported, but this was with varying parameters and with tedious parameter tuning on each specific G-problem.

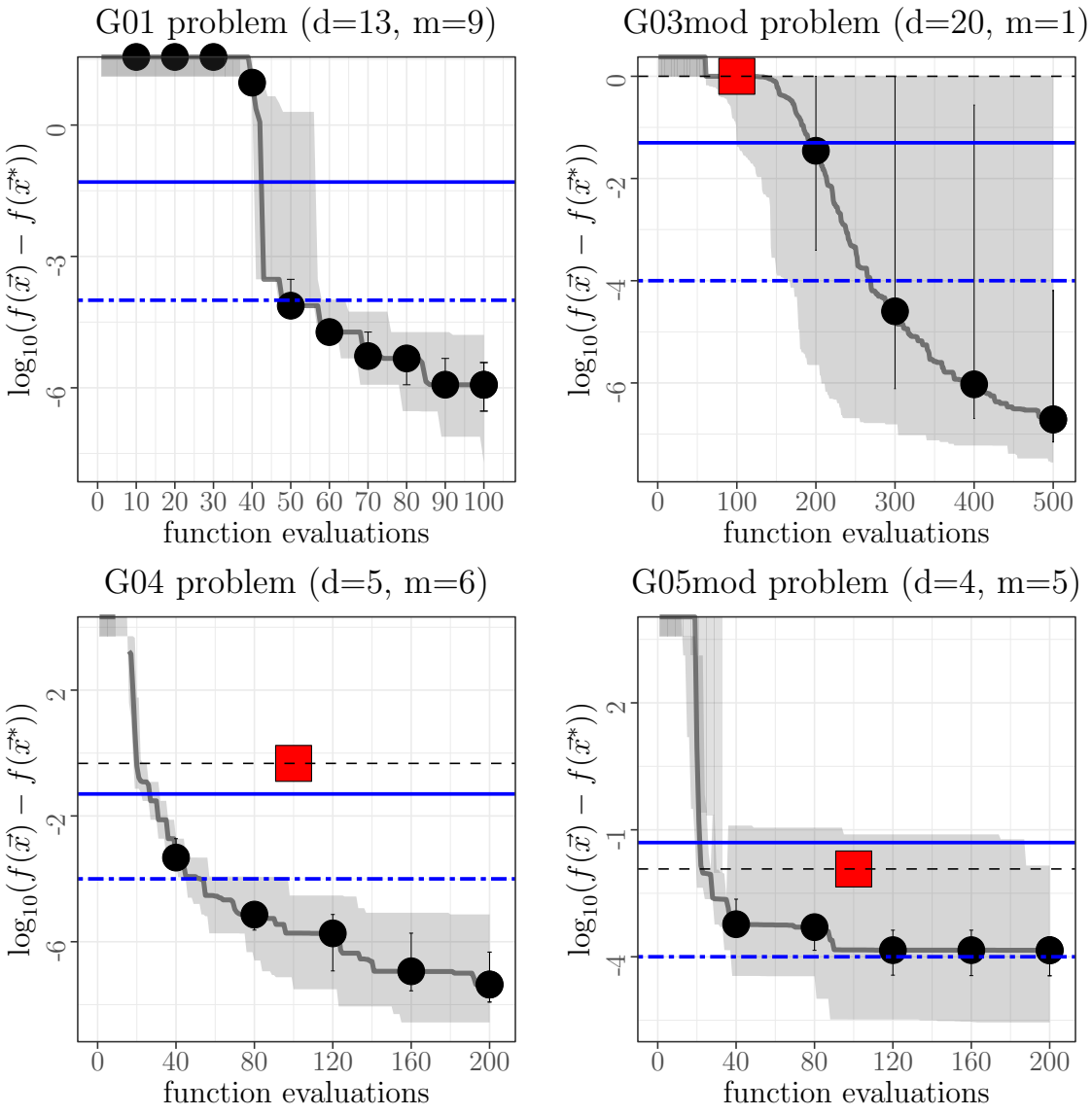

Figure 3.5: SACOBRA optimization process for G01, G03mod, G04, and G05mod. The gray curve shows the median of the error for 30 independent trials. The error is calculated with respect to the true minimum $f(\vec{x}^*)$. The gray shade around the median is showing the worst and the best error. The error bars mark the 25% and 75% quartile. See text for explanation of red square and blue horizontal lines.

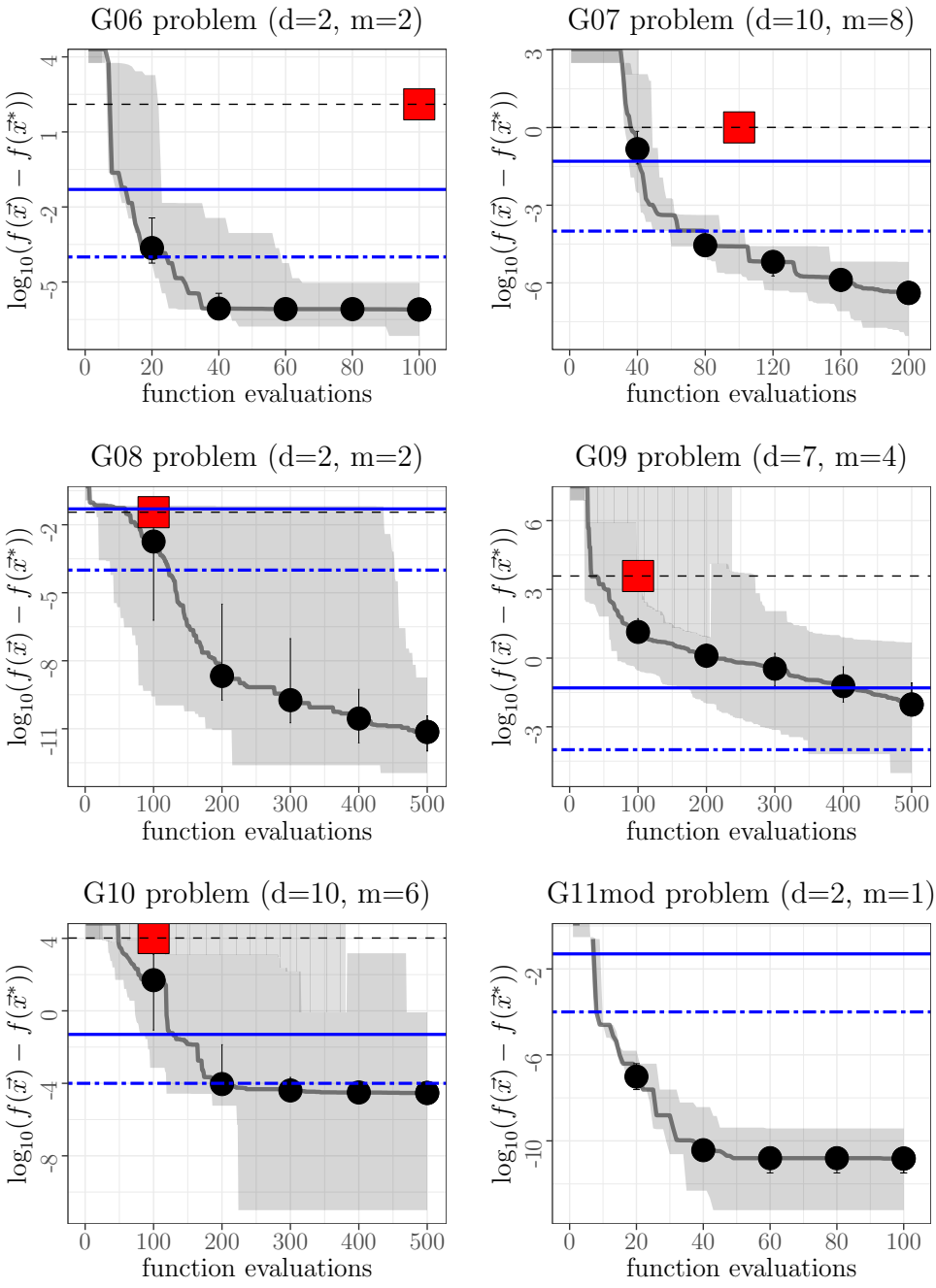

Figure 3.6: SACOBRA optimization process for G06–G11mod. The gray curve shows the median of the error for 30 independent trials. The error is calculated with respect to the true minimum $f(\vec{x}^*)$. The gray shade around the median is showing the worst and the best error. The error bars mark the 25% and 75% quartile. See text for explanation of red square and blue horizontal lines.

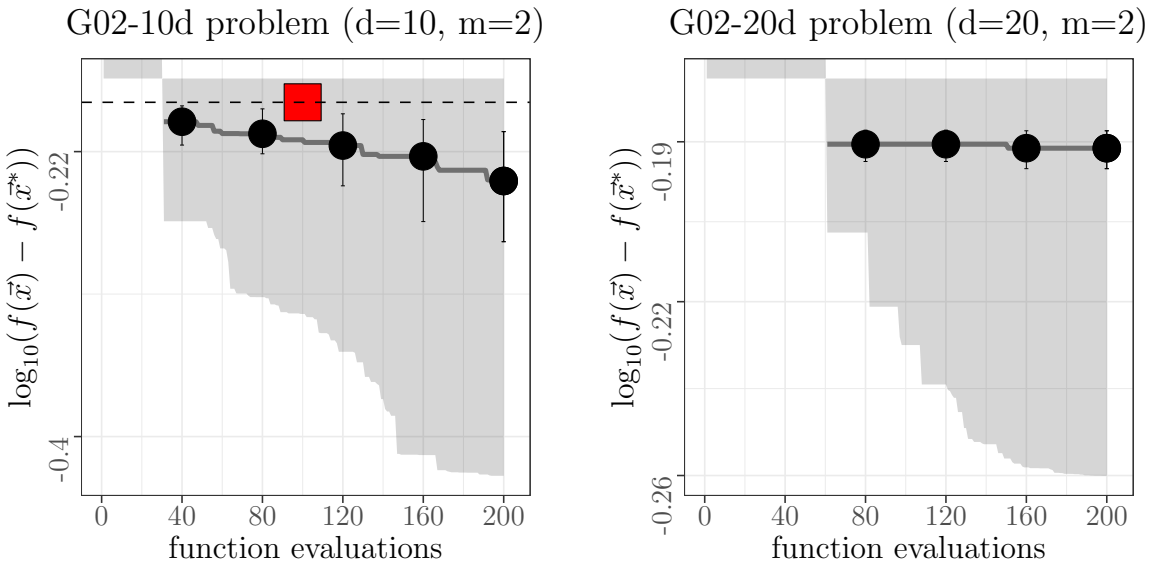

Figure 3.7: SACOBRA optimization process for G02 in 10 and 20 dimensions. The gray curve shows the median of the error for 30 independent trials. The error is calculated with respect to the true minimum $f(\vec{x}^*)$. The gray shade around the median is showing the worst and the best error. The error bars mark the 25% and 75% quartile. See text for explanation of red square.

when rescaling is switched off (early iterations) or when aFF is switched off (later iterations).

Fig. 3.9 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 on 30 runs. Full SACOBRA is significantly better than each reduced SACOBRA or COBRA-variant at least for some G-problems (each column has a dark cell). In addition, each G-problem benefits from one or more SACOBRA extensions (each row has a dark cell). The only exception to this rule is G11mod, but for a simple reason: G11mod is an easy problem which is solved by *all* SACOBRA variants in each run, so none is significantly better than the others.

3.6.4 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

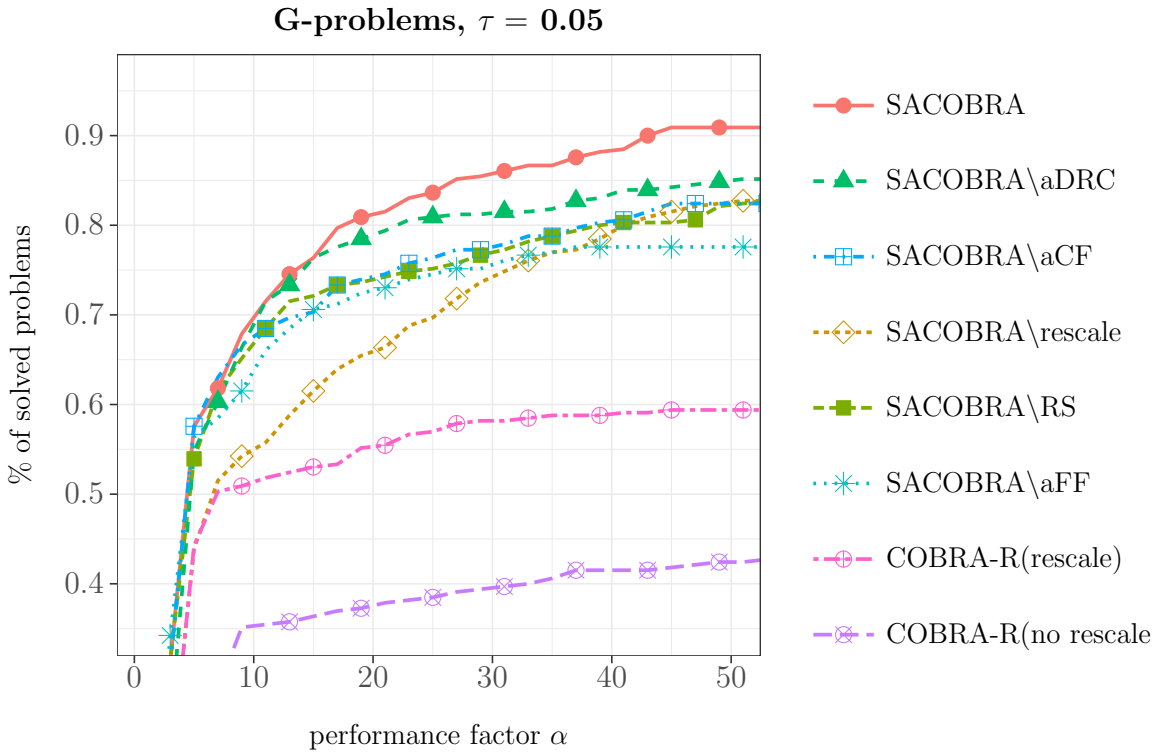

Figure 3.8: Data profile of SACOBRA, SACOBRA\rescale (SACOBRA without rescaling the input space), and other „\"-algorithms are with a similar meaning. The performance factor α is the budget divided by $d+1$ where d is the individual dimension of each test problem (see Sec. 2.5 and Tab. 3.2).

beneficial for two other problems, and with negligible effect on the other problems. Therefore, a careful selection should be done. Although we demonstrated in Sec. 3.4.2 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. 3.5.2 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. 3.10 shows the Q-value for all G-problems. The G-problems are ranked on the horizontal

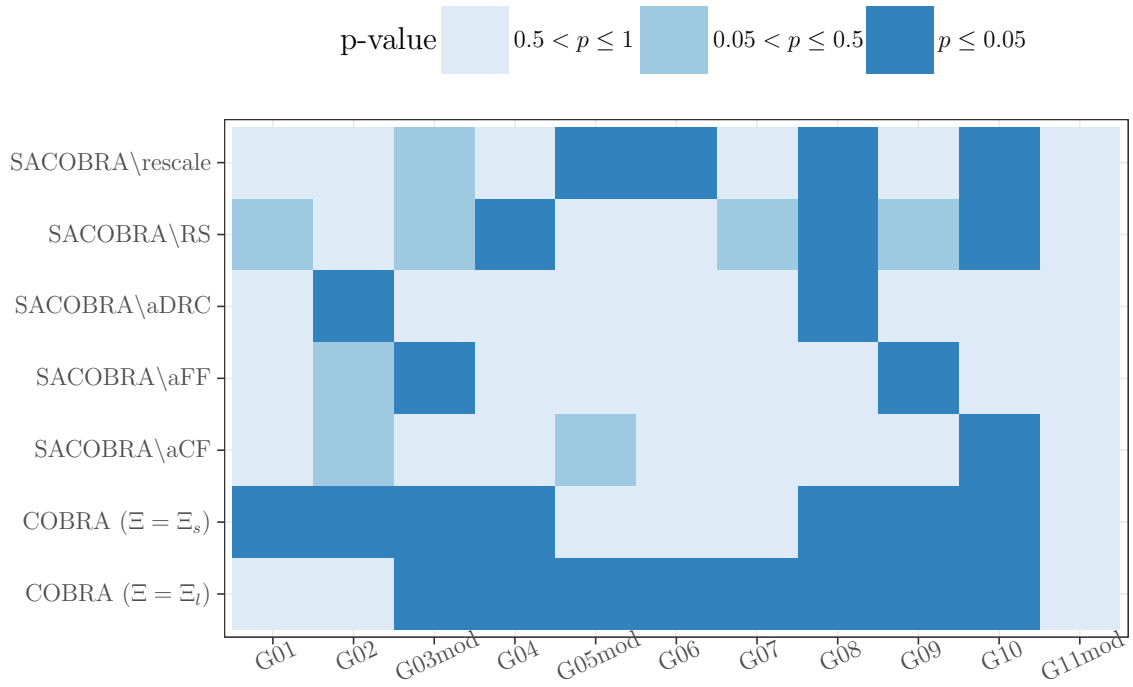

Figure 3.9: Wilcoxon rank sum test, paired, one sided, significance level 5%. Shown is the p-value for the hypothesis that for a specific G-problem the full SACOBRA method at the final iteration is better than one of the other solvers shown along the y-axis. See Fig. 3.8 and Sec. 3.5.2 for our naming conventions 'SACOBRA\rescale' and similar. Significant improvements ($p \leq 5\%$) are marked as cells with dark blue color.

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 G03mod. 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

$$R = \frac{\text{median}(E_{opt})}{\text{median}(E_{opt}^{(plog)})}. \tag{3.9}$$

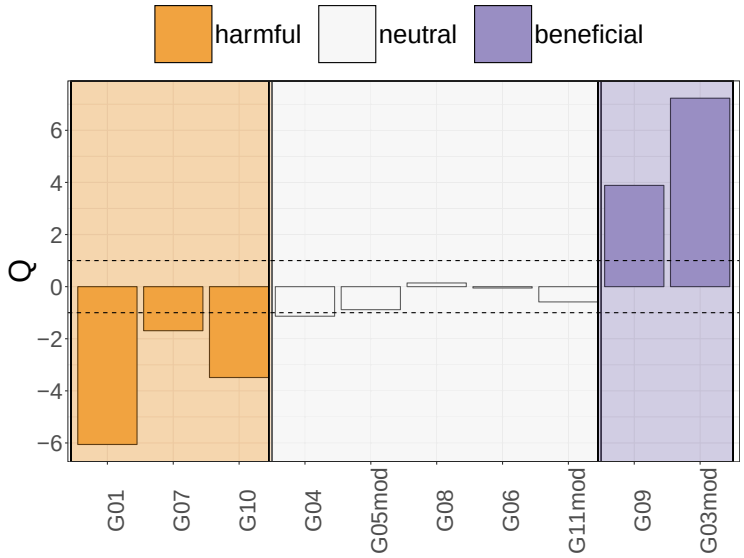

Figure 3.10: Q-value (Eq. (3.8)) at the end of optimization for all G-problems. The G-problems are ordered along the x-axis according to the R-value defined in Eq. (3.9) which measures the impact of *plog* on the optimization performance. Any threshold for Q in $[-1, 1]$ will clearly separate the harmful from the beneficial problems. This figure shows that Q, which is available online, is a good predictor of the impact of *plog* on the overall optimization performance.

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. 3.10 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. 3.10.

[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.

The G-problems for which *plog* is beneficial are G03mod and G09: These are according to Tab.3.2 the two problems with the largest fitness function range FR, thus strengthening our hypothesis from Sec. 3.4.2: 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 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 3.2) *and* having no mixed quadratic terms. Those functions can be fitted perfectly by the polynomial tail (Eq. (2.9)) 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.

3.6.5 Comparison with Other Optimizers

Tab. 4.2 shows the comparison with different state-of-the-art optimizers on the G-problem suite. While ISRES (Improved Stochastic Ranking [151]) and DE (Differential Evolution [31]) are the best optimizers in terms of solution quality, they are cited in the relevant papers with a high number of function evaluations.[13] 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 4.2).

G02 is marked in grey cell color in Tab. 4.2 because it is not solved to the same level of accuracy by most of the optimizers. ISRES and RGA (Repair GA [38]) 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.

The results in column SACOBRA, DE and COBYLA are from our own calculation in `R`. The results in column COBRA, ISRES, RGA and CMA-ES were taken from the papers cited. In two cases (red italic numbers in Table 4.2) the reported solution is better than the true optimum, possibly due to a slight infeasibility. This is explicitly stated in the case of ISRES [150, p. 288], because the equality con-

[12] The analytical form is available in the appendices of [150] or [151].

[13] Strictly speaking we do not know what the results of ISRES or other optimizers after 500 iterations would be, since such ‚early' results are not given in the papers. It is however well-known that those algorithm usually need a larger number of iterations to get high-quality results. For two algorithms, DE and COBYLA, we make a comparison with limited budgets in Fig. 3.11 below and find this hypothesis confirmed.

Table 3.4: 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 [this work]	COBRA [141]	ISRES [151, 150]	RGA 10% [38]	COBYLA [131] (infeas)	DE [31, 187]	CMAES [6]
G01	-15.0	m	**-15.0**	NA	**-15.0**	**-15.0**	-13.83	**-15.0**	NA
		fe	100	NA	350000	95512	12743	59129	NA
G02	-0.8036	m	-0.3466	NA	-0.7931	-0.7857	-0.197 (5)	**-0.8036**	NA
		fe	400	NA	349600	331972	97391	226994	NA
G03mod	-1.0	m	**-1.0**	-0.09	***-1.001***	-0.9999	**-1.0** (3)	-0.9999	NA
		fe	300	100	349200	399804	31069	211966	NA
G04	-30665.539	m	**-30665.539**	-30665.15	**-30665.539**	**-30665.539**	**-30665.539**	**-30665.539**	NA
		fe	200	100	192000	26981	418	33963	NA
G05mod	5126.497	m	**5126.498**	5126.51	**5126.497**	**5126.498**	**5126.498** (7)	**5126.498**	NA
		fe	200	100	195600	39459	194	13375	NA
G06	-6961.81	m	**-6961.81**	-6834.48	**-6961.81**	**-6961.81**	**-6961.81** (3)	**-6961.81**	**-6961.81**
		fe	100	100	168800	13577	134	2857	1060
G07	24.306	m	**24.306**	25.32	**24.306**	24.471	**24.306** (6)	**24.306**	**24.306**
		fe	200	100	350000	428314	13072	94313	11283
G08	-0.0958	m	**-0.0958**	***-0.1***	**-0.0958**	**-0.0958**	-0.0282	**-0.0958**	NA
		fe	200	100	160000	6217	553	990	NA
G09	680.630	m	680.761	3953.97	**680.630**	680.638	**680.630** (2)	**680.630**	**680.630**
		fe	300	100	271200	388453	8973	34836	4106
G10	7049.248	m	7049.253	18031.74	**7049.248**	7049.566	7064.8 (22)	**7049.248**	**7049.248**
		fe	300	100	348800	572629	270840	74875	18781
G11mod	0.75	m	**0.75**	NA	**0.75**	**0.75**	**0.75**	**0.75**	NA
		fe	100	NA	137200	7215	11788	2190	NA
	average fe		218	100	261127	210012	40652	68681	8807

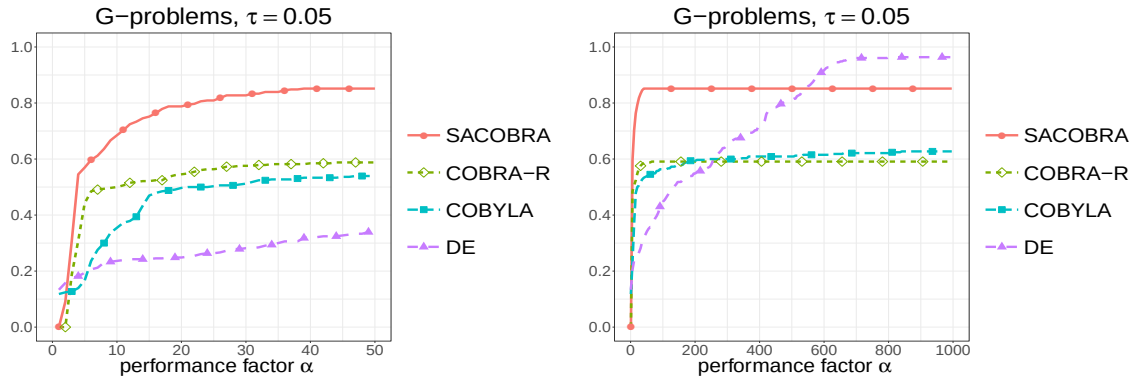

Figure 3.11: Comparing the performance of the algorithms SACOBRA, COBRA (with rescale), Differential Evolution (DE), and COBYLA on optimizing all G-problems G01-G11mod (30 runs with different initial random populations).

straint $h(x) = 0$ of G03 is transformed into an approximate inequality $|h(x)| \leq \epsilon$ with $\epsilon = 0.0001$.

COBRA [141] comes close to SACOBRA in terms of efficiency (function evaluations), but it has to be noted that [141] does not present results for all G-problems (G01 and G11mod are missing and G02 results are for 10 dimensions, but the commonly studied version of G02 has 20 dimensions). Furthermore, for many G-problems (G03mod, G06, G07, G09, G10) a manual transformation of the original fitness function or the constraint functions was done in [141] 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.

CMA-ES [6] has only results for the subset of 4 unimodal objective functions within the set of 11 G-problems. On this subset it shows the best results of all non-COBRA optimizers in terms of function evaluations, although COBRA and SACOBRA are even better. It has to be noted that the algorithm of [6] has the freedom to take a different number of objective and constraint function evaluations. Table 4.2 shows the bigger of those values (the number of constraint function evaluations). The number of objective function evaluations is smaller by a factor of 3–5. In addition, the results in [6] were obtained under stricter *„solved“*-thresholds $\tau' \in [10^{-7}, 10^{-4}]$ while SACOBRA used $\tau = 0.05$ (see Sec. 2.5).

Table 3.5: 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	G05mod
SACOBRA\ RS	13	G03mod, G05mod, G07,G09,G10
SACOBRA\ aDRC	0	–
SACOBRA\ aFF	1	G10
SACOBRA\ aCF	0	–
COBRA (no rescale)	37	G03mod,G05mod,G07,G09,G10
COBRA (rescale)	23	G05mod,G07,G09,G10
COBYLA	48	G02,G03mod,G05mod,G06,G07,G09,G10
DE	0	–

Fig. 3.11 shows the comparison of SACOBRA and COBRA with other well-known constraint optimization solvers available in R, namely DE[14] and COBYLA.[15] The right plot in Fig. 3.11 shows that DE achieves very good results after many function evaluations ($\alpha > 800$), in accordance with Table 4.2. But the left plot in Fig. 3.11 shows that DE is not really competitive if very tight bounds on the budget are set. This result proves only that DE has inferior results to SACOBRA on the G-problem suite for low budgets, but we believe that similar results would also emerge for ISRES, the other high-quality optimizer.

Table 3.5 shows that SACOBRA significantly reduces the number of infeasible runs as compared to COBRA. Most of the SACOBRA variants have less than 2% infeasible runs whereas COBRA has 7-11%. The full SACOBRA method has no infeasible runs at all.

3.6.6 MOPTA08

Fig. 3.12 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

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

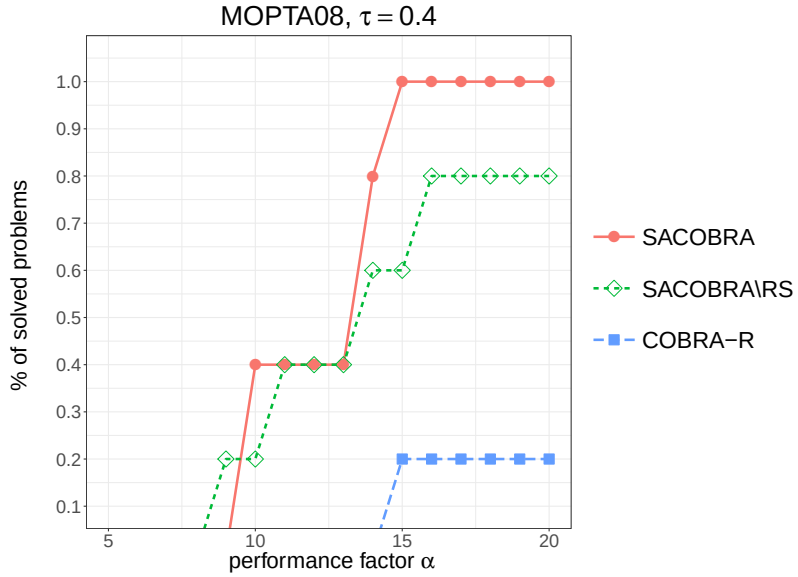

Figure 3.12: Data profile for MOPTA08: Same as Fig. 3.8 but with 10 runs on MOPTA08 with different initial designs. The curves for SACOBRA without rescale, aDRC, aFF, or aCF are identical to full SACOBRA, since in the case of MOPTA08 the objective function and the constraints are already normalized.

profile of Fig. 3.12 if it is not more than $\tau = 0.4$ away from the best value obtained in all runs by all algorithms.

Table 3.6 shows the results after 1000 iterations for Regis' recent trust-region based approach TRB [142] and our algorithms. We can improve the already good mean best feasible results of 227.3 and 226.4 obtained with COBRA [98] and TRB [142], resp., to 223.3 with SACOBRA. The reason that SACOBRA\RS is slightly better than COBRA [98] is that SACOBRA uses an improved DRC.

3.7 Discussion

3.7.1 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

Table 3.6: Comparing different algorithms on optimizing MOPTA08 after 1000 function evaluations.

Algorithm	best	median	mean	worst
COBRA [98]	226.3	227.0	227.3	229.5
TRB [142]	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**

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 of optimization). Exclusion of either one of these two elements led to the largest performance drop in Fig. 3.8 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. 3.4 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 G03mod with a very large fitness range FR (Table 3.2) 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.

3.7.2 Limitations of SACOBRA

Highly Multimodal Functions

Surrogate models like RBF are a great technique for efficient optimization and often (not always) it is only with their help possible to solve constrained optimization problems in less than, say, 1000 iterations. But a current limitation for surrogate modeling are highly multimodal functions. G02 is such a function, it has a large number of local minima. Those functions have usually many 'ups and downs' (informally speaking). If a surrogate model interpolates isolated points of such a function, it tends either to overshoot in other parts of the function (if the isolated points are close to each other) or it puts a smooth but inaccurate surface through the points (if the isolated points are sparsely scattered over the search volume). 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.

G02 is one of the few problems in the G-problem suite which is scalable. Our algorithm has severe difficulty to solve the 20-dimensional G02 (which is the standard version). We were curious to see if SACOBRA can handle G02 in lower dimensions where the complexity of the fitness function is dramatically reduced. That's why we applied our algorithm on the 10-dimensional G02 as well (Fig. 3.7). But Fig. 3.7 shows that there is only very small improvement when scaling down to 10 dimensions.

Non-smooth or Noisy Functions

The G-problem test suite – although challenging due to the very diverse characteristics of the problems – is idealizing or too simplicistic with respect to one feature: The functions in the problems are all relatively smooth and noise-free. Real-world problems are often a) non-smooth (locally garbled) or b) noisy or c) the sample points can only be set with a certain uncertainty. Any of these three factors will contribute to a common difficulty for RBF surrogate models: If functions in such problems are sampled only very sparsely with few sample points, it becomes very hard to build reliable models.

It is a subject for ongoing research to quantify whether the ideas of SACOBRA developed here carry on for the case of non-smooth functions as well. It is well known that RBF models are ideal candidates for (smooth) local polynomial models. If the

functions are non-smooth or noisy, it is likely that RBF models due to their interpolating behavior degrade rapidly (due to overfitting). We believe that the same would apply to Kriging surrogate models or other interpolating models [16]. An interesting approach for further research would be to replace the interpolating surrogate models by approximating (non-interpolating) surrogate models to avoid overfitting. In the area of computer graphics there have been good results for approximating RBF models build from noisy data of a 3D-scanner [33]. A challenge for optimization under restricted budgets will be to find the right degree of approximation (smoothing factor) from only relatively few samples.

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. 3.5.1. A reformulation of an equality constraint $h(x) = 0$ as inequality $|h(x)| \leq 0.0001$ as in [107, 150] is not well-suited for COBRA and for RBF modeling.[17] We used in this work the same approach as Regis [141] and replaced each equality operator with an inequality operator of the appropriate direction (see Sec. 3.6.1). The equality-to-inequality transformation of constraints severely changes the nature of the optimization problem since it fundamentally changes the feasible volume. Therefore, we renamed the modified problems by adding a mod as suffix to clearly distinguish between them. It has to be noted that such a modification is not viable for problems with more complicated objective functions having minima on both sides of an equality constraint hypersurface. An equality handling technique for SACOBRA was developed to solve this problem in [16] and will be described in detail in Ch. 4.

3.7.3 Comparison of Solution Qualities

A final cautionary remark is necessary here: The term *„solved“* is defined quite differently in different works and this makes it sometimes difficult to compare results from different papers. While we have for example chosen the threshold $\tau = 0.05$, the CEC 2006 competition had a stricter threshold $\tau = 0.0001$. A fair comparison would either test at same τ-levels (we could do this here only for DE and COBYLA, see

[16] In Kriging metamodels, which follow a kernel-based approach similar to RBF, but are motivated by statistical assumptions, recently some noise handling mechanisms have been addressed, such as the nugget effect [99].

[17] The reason is that RBF modeling is not very accurate for modeling functions with sharp ridges, as in the case of $|h(x)| \leq 0.0001$.

Fig. 3.11) or use other measures avoiding the *solved* criterion, e. g. mean or median error at different iterations.

3.8 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 lead to 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 **Q3.1** 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. 3.5.2. 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. 3.8).[18] In this chapter the main contribution is 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. Therefore, we can conclude a result to **Q3.3**: SACOBRA requires **less than 500 function evaluations** to solve 10 out

[18]This is for the case of threshold $\tau = 0.05$ used in our definition of *solved problems*. If we use the stricter threshold $\tau_{CEC} = 0.0001$ used in the CEC 2006 competition, we can solve 76% of all tested problems.

of 11 G-problems (exception: G02) with similar accuracy as other state-of-the-art algorithms. Those other algorithms often need more function evaluations by a factor between 300 and 1000. The *solved*-condition was defined slightly different here than in the CEC 2006 competition (see Sec. 3.7.3). Under this condition it could be shown that SACOBRA can be used to solve a wide range of constrained optimization problems with nonlinear constraints **Q3.2**.

Future research in this area may be devoted to overcome the current limitations of SACOBRA mentioned in Sec. 3.7.2. As we show in the next chapter the equality constraint limitation is now mainly solved [16], therefore the challenging remaining limitations are: (a) highly multimodal functions like G02 and (b) non-smooth or noisy functions or functions with a certain level of uncertainty.