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

Bagheri, S.

Chapter 5

SOCU: EGO-based Constrained Optimization

5.1 Outline

Real-world optimization problems are often subject to several constraints which are expensive to evaluate in terms of cost or time. The SACOBRA optimization framework introduced in Ch. 3 and Ch. 4, is an example for a strong surrogate-assisted optimizer, developed to tackle such expensive optimization problems. SACOBRA makes use of radial basis function interpolation as surrogates to save some expensive function evaluations. Another modeling approach which is very popular as surrogate for unconstrained optimization problems, is Kriging aka Gaussian processes modeling. Efficient Global Optimization (EGO), a well-known Kriging-based surrogate-assisted algorithm [90], is an example for such surrogate-assisted solvers. This algorithm was originally proposed to address unconstrained problems but later was modified to solve constrained problems [159].

Although the already existing Kriging-based constrained solvers [159, 51] have some bottlenecks in handling constraints in an efficient manner, the attractive property of the Kriging method, being able to determine an uncertainty level of the model at each point, motivates us to invest on improvement of a constrained EGO-based optimizer [159]. We are interested to know if an EGO-based constrained solver, benefiting from the probabilistic modeling, can compete with SACOBRA in solving challenging COPs. To do so, we try to develop an algorithm which overcomes some common issues of EGO-based algorithms.

The common issues that many EGO-based optimizers suffer from are mainly: (1) early stagnation, (2) problems with multiple active constraints and (3) frequent crashes. In this chapter, a new EGO-based algorithm is introduced which tries to overcome these common issues with Kriging-based optimization. We apply the proposed algorithm on COPs subject to inequality constraints with dimension $d \leq 4$

from the G-function suite [107] and on an airfoil shape optimization example. The analysis in this chapter is based on the work of Bagheri et al. [14].

The rest of this chapter is organized as follows: In Sec. 5.2 we explain our motivation and also we pose two research questions which will be answered in Sec. 5.7. The related work (Sec. 5.3) categorizes different Kriging-based constrained solvers. After briefly describing Kriging surrogate models and EGO in Sec. 5.4, the constraint handling approach is described and then the proposed plugin control algorithm for preserving feasibility is explained in the same section. Sec. 5.5 explains the experimental setup. Results are shown and analyzed in Sec. 5.6. Finally, this chapter is concluded and the regarding research questions are addressed in Sec. 5.7.

5.2 Introduction

A constrained optimization problem (COP) can be defined as the minimization of an objective function (fitness function) f subject to inequality constraint function(s) $g_1, \ldots, g_m$ as described in Eq. (3.4).

Real-world COPs are often very expensive to evaluate which means only a very limited number of function evaluations is allowed in practice. Giannakoglou [67] presents an overview on efficient optimization approaches and shows indicative examples from aerodynamics. Therefore, a proper optimizer for this sort of problems should be able to find optimal or near-optimal solutions with a low number of function evaluations. Many different surrogate-assisted techniques have been developed to tackle expensive COPs [15, 127, 177, 29, 141]. The main idea adopted by nearly all surrogate-assisted optimization algorithms is to use fast mathematical or statistical models for the optimization process and only evaluate a new solution on the expensive function when the model needs to be updated. Among the existing multitude of surrogate modeling approaches with various properties, Kriging [103, 63] appears to be one of the most attractive techniques. Kriging, aka Gaussian Processes, offers a strong modeling tool which provides an estimate of the model uncertainty in addition to the model of the function. Although in recent years many studies have been undertaken with different Kriging-based unconstrained and constrained optimizers, there are still limitations associated with the use of Kriging models. One of the issues that almost all Kriging-based strategies face is that frequent crashes occur during the optimization process. The crashes usually happen when a new solution is located very close to a former one. We find that adding a noise variance is an effective way to handle such stability issues.

Another point is that most of the existing Kriging-based constrained optimization strategies are evaluated only on simple 2-dimensional benchmark functions mostly with only one active constraint [159, 51, 153, 152, 73]. However, real-world COPs are often multi-constrained and are not limited to 2-dimensional problems. This motivates us to develop a new Kriging-based optimization algorithm which avoids crashes and is applicable on challenging COPs like G-problems. We evaluate the proposed algorithm on all benchmarks with dimension $d \leq 4$ taken from the challenging G-function suite [107]. Throughout this chapter we try to answer the following research questions:

Q5.1 Is it possible to modify existing Kriging-based optimization algorithms to handle challenging COPs with multiple *active* constraints?
Good results achieved by Kriging-based constrained optimizers are limited to COPs with one or none active constraints. In this chapter we will propose a modification for the standard Kriging-based constraint optimizer [159] and test whether this can help to tackle COPs with multiple active constraints.

Q5.2 Is it possible to balance the exploration of feasible and infeasible infill points in a proper way?
Balancing the exploration of feasible and infeasible infill points is vital to surrogate-assisted methods. However, Kriging-based constrained optimizers often suffer in this matter. In this chapter we investigate the reasons behind such unbalance and propose a treatment for it.

5.3 Related Work

Most Kriging-based constrained optimization algorithms make use of Kriging's statistical property, the expected improvement function [116, 90], for efficiently solving global COPs. We can categorize such algorithms into three main groups.

(a) The first group of algorithms transform a constrained problem into an unconstrained problem. Schonlau et al. [159] is an example of type-(a) algorithm. This algorithm suggests to maximize the multiplication of the expected improvement of the objective function and the probability of feasibility, which are both statistical measures determined from Kriging models of objective and constraint functions. This algorithm fails if the number of active constraints are large or if the objective function is very steep or flat around the feasibility boundary. Algorithms which tend to maximize the penalized expected improvement [152, 124] also belong to the first group.

(b) Another approach to address COPs is to solve a constrained sub-problem. As an example, we can name a work from Sasena et al. [153] in which the expected improvement is maximized subject to the approximation of the constraint functions. Audet et al. [8] maximize the expected improvement subject to the expected violation of each constraint.

(c) Methods in the third category transform the constrained problems to multi-objective unconstrained problems and then use multi-objective optimizers. These methods often consider the expected improvement of the fitness function as one objective and one or more statistical properties of the constraint functions as other objective(s) [84, 125, 51]. Although Durantin et al. [51] show that the type-(c) algorithms perform better than the existing algorithms from type (a) and (b), we have to consider that solving a multi-objective problem is a complex task. An increase in problem dimension or number of active constraints makes such algorithms very time-consuming.

To the best of our knowledge the current state-of-the-art for solving the G-problems is SACOBRA [15] (self adjusting COBRA). SACOBRA uses RBF surrogate models. SACOBRA is *not* a Kriging-based COP and it does not use the EI approach.

In this chapter a new type-(a) algorithm called SOCU (**S**urrogate-Assisted **O**ptimization encompassing **C**onstraints and **U**ncertainties) is described and compared with SACOBRA.

5.4 Methods

5.4.1 Kriging Surrogate Models

Kriging is a statistical modeling technique based on Gaussian processes. This algorithm approximates the function $f(\vec{x})$ with the surrogate model

$$Y \;=\; \mu + e(\vec{x}), \tag{5.1}$$

where μ is the average of the stochastic process and the error term $e(\vec{x})$ is normally distributed with mean 0 and variance $\sigma^2(.)$. The estimation of μ and σ are the heart of Kriging modeling and described in more detail in the standard Kriging literature [103, 90].

5.4.2 Expected Improvement with Constraints

Efficient global optimization (EGO) is an algorithm developed by Jones et al. [90] for unconstrained optimization based on Kriging. The main idea of the EGO algorithm – originally introduced by Močkus et al. [116] – is to balance between exploration and exploitation by maximizing the expected improvement in Eq. (5.2) during a sequential optimization process:

$$\begin{aligned} EI(x) &= E\left[\max(f_{min} - Y, 0)\right] \\ &= (f_{min} - \mu(\vec{x}))\Phi\Big(\frac{f_{min} - \mu(\vec{x})}{\sigma(\vec{x})}\Big) + \sigma(\vec{x})\varphi\Big(\frac{f_{min} - \mu(\vec{x})}{\sigma(\vec{x})}\Big), \end{aligned} \tag{5.2}$$

where the plugin f_{min} is the fitness value of the best-so-far solution and Φ and φ are the cumulative and probability density function of the standard normal distribution. Schonlau et al. [159] extended the EGO algorithm to handle inequality constraints. Their algorithm maximizes the penalized expected improvement function EI_p shown in Eq. (5.3) which is the product of the expected improvement (now with plugin f_{min} being the best *feasible* fitness value) and the feasibility function $F(x)$

$$EI_p(\vec{x}) = EI(\vec{x}) \cdot F(\vec{x}) = EI(\vec{x}) \cdot \prod_{j=1}^{m} P(g_j(\vec{x}) < 0), \tag{5.3}$$

where $P(g_j(\vec{x}) < 0)$ is the probability of $g_j(\vec{x})$ to be feasible, measured with the help of the Kriging model for the jth constraint:

$$P(g_j(\vec{x}) < 0) = \Phi\Big(\frac{-\mu_j(\vec{x})}{\sigma_j(\vec{x})}\Big). \tag{5.4}$$

This algorithm often faces difficulties in solving COPs with two or more active constraints, because the product of feasibility probabilities approaches zero near the feasibility border where the optimum is located[1]. Therefore, the penalized expected improvement may have very small values in the interesting region. Furthermore, this algorithm is unlikely to sample infeasible solutions. Others have shown that existence of the infeasible solutions in the population can often be helpful [151, 150]. In order to give solutions around the boundary a higher chance to be selected, we modify the feasibility function introduced by Schonlau et al. [159] and formulate a

[1] In presence of a active constraints the feasibility function is $F(\vec{x}^*) = (\frac{1}{2})^a$ at the optimum, which goes rapidly to zero as the number of active constraints grows.

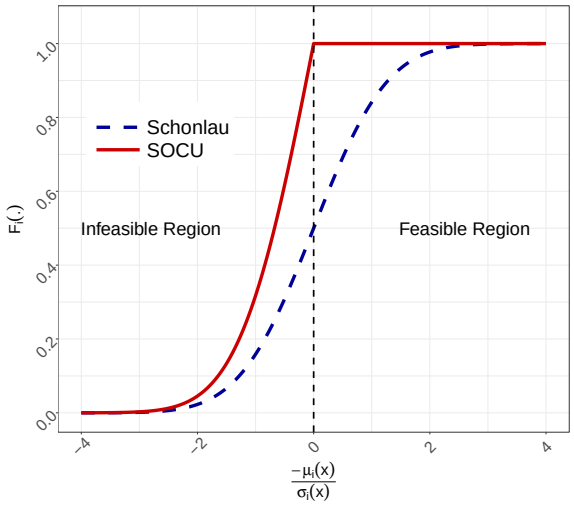

Figure 5.1: Feasibility function for the i-th constraint $F_i(\vec{x})$. The total feasibility function is $F(\vec{x}) = \prod F_i(\vec{x})$

modified expected improvement function EI_{mod} as follows:

$$EI_{mod}(\vec{x}) = EI(\vec{x}) \cdot F(\vec{x}) = EI(\vec{x}) \cdot \prod_{i=1}^{m} \min\left(2\Phi\left(\frac{-\mu_i(\vec{x})}{\sigma_i(\vec{x})}\right), 1\right). \tag{5.5}$$

Fig. 5.1 shows the different feasibility functions used in Schonlau algorithm (blue dashed line) and our proposed algorithm (red line) for one constraint.

The proposed algorithm SOCU, shown in Alg. 5, initially maximizes the feasibility function in order to find at least one feasible solution. As soon as one feasible solution is found, the algorithm proceeds by maximizing the modified expected improvement (Eq. 5.5) in each iteration. Since the EI_{mod} function is highly multimodal, we decided to use a simulated annealing method as the internal optimizer.

5.4.3 Plugin Control (PC): Preserving Feasibility

In our first experiments with EI_{mod} we observed a strange behavior depicted in Fig. 5.2: Initially, the surface plot of EI_{mod} looks as expected (left plot), but the best feasible solution is still far away from the true solution. When finding better solutions near the true optimum, the EI_{mod} surface would suddenly change (right plot) and the maximum of EI_{mod} shifts far away into infeasible area. Now the EI_{mod} infill

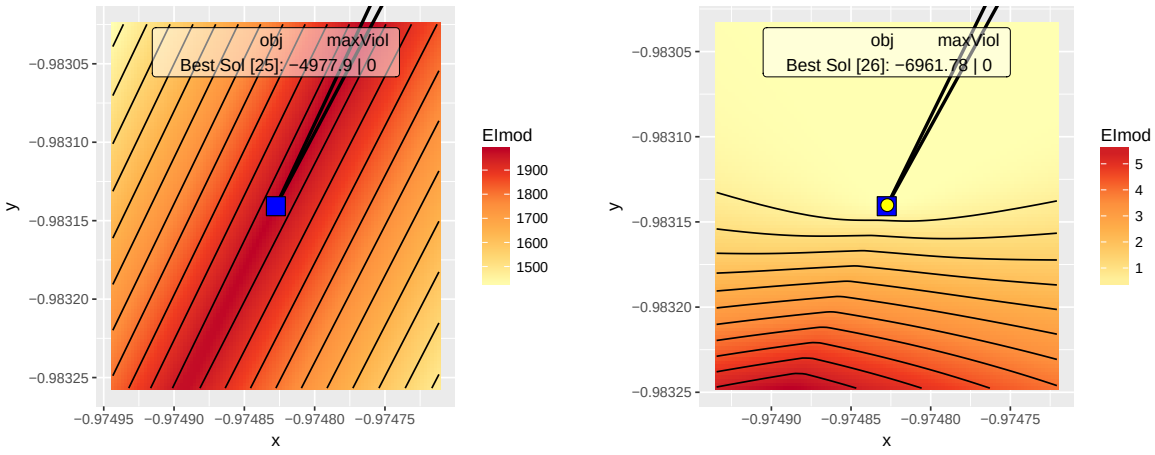

Figure 5.2: EI_{mod} shift towards the infeasible area for a 2-dimensional test problem (G06). The thick lines in form of a pointed triangle show the feasible area, the blue square is the true solution. The yellow circle is the best feasible solution (being outside the plot area in the left plot).

criterion will always suggest infeasible infill points and thus the algorithm stagnates. What is the reason for this behavior? A closer analysis revealed that the plugin f_{min} in Eq. (5.2) is responsible for this. Usually, the plugin f_{min} is taken as the best feasible objective found so far. A new value for f_{min} does not change the maxima locations of EI, but it changes the intercept. This has a large effect on the maxima locations of $EI_{mod}(x) = EI(\vec{x}) \cdot F(\vec{x})$. We explain this with a 1D example.

The 1D-Case

Consider the following simple 1D-model, where $\vec{x} = x$:

$$EI(x) = \max(ax + b, 0), \tag{5.6}$$

$$F(x) = \min\left(2\Phi\big(kx\big), 1\right). \tag{5.7}$$

We assume $k > 0$, so that $x < 0$ is the infeasible area, and $a < 0$, i. e. the (unconstrained) $EI(x)$ has better values towards $x < 0$. This makes the constraint *active*, meaning that the constrained optimum is on the border $x = 0$.

What happens now for EI_{mod} as a function of the intercept b? As Fig. 5.3 depicts, large b have the optimum for EI_{mod} correctly at $x = 0$, but too small values for b lead to a false shift of EI_{mod}'s maximum towards the infeasible area. A short Taylor

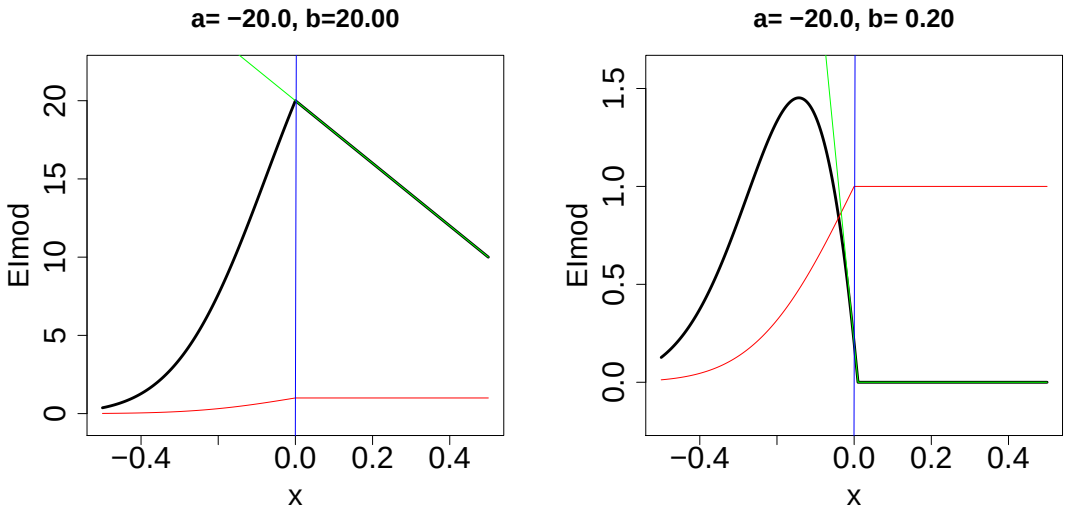

Figure 5.3: EI (green), F (red), and EI_{mod} (black) in the 1D-case. Right: For small intercepts b the optimum of EI_{mod} is shifted towards the infeasible area $x < 0$.

expansion shows that the critical intercept is

$$b_{crit} = \frac{a\sqrt{2\pi}}{2k}. \tag{5.8}$$

Smaller intercepts have the maximum of EI_{mod} shifted to the infeasible area.

To correct this, we simply have to change the plugin in Eq. (5.2):

$$EI(x) = E\left[\max(b_{crit} + f_{min} - Y, 0)\right] \tag{5.9}$$

This ensures that at the active border, where Y is not larger than f_{min}, the value of $EI(x)$ is at least b_{crit}. – If we have multiple constraints, we calculate b_{crit} for each of them.

The 2D- and nD-Case

In the higher-dimensional case ($d > 1$) we have to find the direction $\vec{g}$ of slowest descent of EI_{mod} at the current best feasible point. This is for example in the case of G06 the bisecting line of the two constraints. The slopes a and k for EI and the

Algorithm 5 SOCU algorithm

m: number of constraints
n: number of evaluated points
d: dimension of the problem
$pop^{(n)}$: population of $n = 5d$ initial points generated by LHS
while $n \leq Budget$ **do**
    Build from $pop^{(n)}$ the Kriging models for objective function f: (μ_0, σ_0) and the m constraints g_j: $(\mu_1, \sigma_1), \ldots, (\mu_m, \sigma_m)$
    Obtain $EI(\vec{x})$ from Eq. (5.2) with plugin corrector Eq. (5.9)
    $F(\vec{x}) = \prod_{j=1}^{m} \min\Big(2\Phi\big(-\frac{\mu_j(\vec{x})}{\sigma_j(\vec{x})}\big),\, 1\Big)$
    $EI_{mod}(\vec{x}) = EI(\vec{x}) \cdot F(\vec{x})$
    if a feasible solution has been found **then**
        $\vec{x}_{new} = \arg\max(EI_{mod}(\vec{x}))$ ▷ Use simulated annealing
    else
        $\vec{x}_{new} = \arg\max(F(\vec{x}))$
    end if
    Add $\vec{x}_{new}$ to $pop^{(n)}$ and evaluate it on true f and $g_1, \ldots, g_m$
    $n \leftarrow n + 1$
end while

constraint(s) in Eq. (5.8) have to be replaced by the respective slopes along direction $\vec{g}$.

5.5 Experimental Setup

5.5.1 General Setup

Initially we test the proposed algorithm on a toy problem *Sphere4* of steerable difficulty: This problem has a sphere as objective function and 4 linear constraints, 2 of them being active, 2 inactive. The constraints enclose a feasible region with a triangular tip of angle ϕ (see Fig. 5.4).

Next, we apply SOCU to all G-problems from the G-problem suite having 4 or less dimensions (see Tab. 7.2).[2] This is because it is well known that Kriging algorithms

[2] G02 and G03mod are problems scalable in their dimension d. We use here $d = 2$.

Table 5.1: Characteristics of the G-functions: d: dimension, 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	ρ	FR	GR	LI	NI	a
G02	2	99.997%	0.57	2.632	1	1	1
G03mod	2	21.5%	1.99	1.000	0	0	1
G05mod	4	0.0919%	8863.69	1788.74	2	3	3
G06	2	0.0072%	1246828.23	1.010	0	2	2
G08	2	0.8751%	1821.61	2.393	0	2	0
G11mod	2	66.724%	4.99	1.000	0	0	1
G15mod	3	0.0337 %	586.0	1.034	1	1	2
G24	2	0.44250%	6.97	1.82	0	2	2
XFOIL	4	0.1349 %	0.99	1.34	0	3	1

are viable only for not too large dimensions. Equality constraints are translated to inequality constraints in the same way as Ch. 3.[3]

For each algorithm we run 30 independent trials with different $n = 5d$ initial points. Constraint violations smaller than 10^{-5} are tolerated. We consider a fixed budget of 100 function evaluations for all problems. The Kriging models are built by R [137] packages DiceKriging and DiceOptim. In order to optimize EI_{mod} we use Generalized Simulated Annealing (R package GenSA). The time limit for this internal optimizer is set to 10 seconds in each iteration, allowing for several thousand Kriging model evaluations (depending on problem size). Additionally, we run SOCU on an application example from aerodynamics described in the next section.

5.5.2 Aerodynamic Shape Design Problem

The benchmarks related to aerodynamic shape design problems represent an ideal platform to test optimization systems and algorithms that make use of surrogate methods for the evaluation of the objective functions and the constraints [87]. This is for two main reasons, namely that the evaluation of objectives and constraints very often requires a significant computational effort, and that both objectives and constraints may have a degree of non-linearity which is a function of the computational

[3]The inequality sign is chosen in such a way that the constrained optimum stays at the same location.

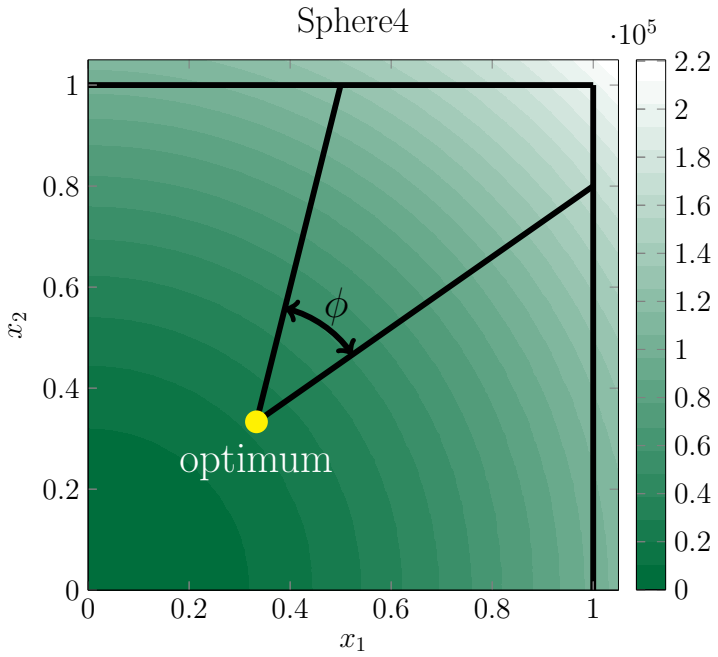

Figure 5.4: *Sphere4* problem. The colored contour levels show the objective function $10^5 \cdot (x_1^2 + x_2^2)$ (2D sphere function). The thick black lines depict 4 linear constraints enclosing the feasible region. The difficulty of this problem is scalable by changing the feasibility angle ϕ.

model used, the operating conditions considered and the shape parametrization chosen. The problem presented here is a simple subsonic airfoil section design exercise derived from [135], but with some special features in terms of ease of implementation, flexibility, reproducibility and availability of the analysis codes that allow its use even in contexts not specialized to aerodynamic design.

Problem Setup

The goal of this optimization problem is the reduction of the aerodynamic drag of a given airfoil changing its shape. A generic airfoil shape is parametrized as linear combination of an initial geometry, defined parametrically by $(x_0(s), y_0(s))$, and a number of functions $y_i(s)$ that may be defined analytically or by point distributions

[80]:

$$y(s) = k\left(y_0(s) + \sum_{i=1}^{n} w_i y_i(s)\right), \qquad x(s) = x_0(s) \tag{5.10}$$

where the airfoil shape is controlled by the design parameters w_i and by the scale factor k. The operating conditions are Mach number equal to 0.0 (incompressible flow) and Reynolds number equal to 3000000. The starting airfoil is the NACA 2412 [1]. The Mach and Reynolds numbers that characterize the specified regime of operation are sufficiently low to allow an extended laminar bucket that can have a beneficial effect on a large part of the flight envelope. The design goals are translated into the following optimization problem:

$$\begin{cases} \min C_D \\ \text{subject to:} \quad C_L = 0.5 \\ \qquad\qquad\quad C_M \geq -0.07 \\ \qquad\qquad\quad C_{Dp} \geq 0 \\ \qquad\qquad\quad t/c = 0.12 \\ \qquad\qquad\quad \ell_R \geq 0.006 \end{cases} \tag{5.11}$$

where C_D, C_{Dp}, C_L and C_M are the drag, pressure drag, lift and pitching moment coefficient of the airfoil; t/c denotes the thickness to chord ratio. The two equality constraints defined in (5.11) are here satisfied by explicitly changing two free problem parameters and therefore they are not considered by the optimization algorithm. In particular, the constraint on t/c is satisfied by changing properly the free parameter k, while the constraint on C_L is satisfied by changing the second free parameter, namely the airfoil angle of attack α.

The aerodynamic analysis code here selected to evaluate the airfoil performance is Drela's XFOIL code [47]. This code is based on a second order panel method interactively coupled to a boundary layer integral module. Laminar-to-turbulent flow transition is predicted using the method described in [48].

5.6 Results

5.6.1 Demonstration on *Sphere4*

The *Sphere4* problem illustrated in Fig. 5.4 is a constrained optimization problem of steerable difficulty. It is used here to demonstrate the difference between the Kriging-based algorithms. Fig. 5.5 shows the results on *Sphere4*. As expected, the

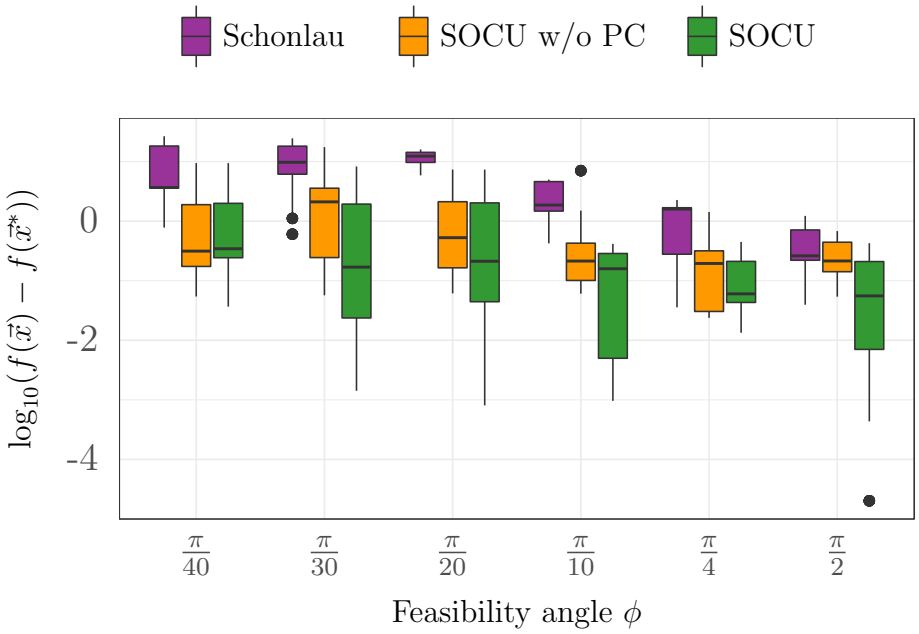

Figure 5.5: Comparing the final optimization error determined by different algorithms for optimizing *Sphere4* problems with different feasibility angles. The results are taken from 30 independent runs and 100 function evaluations.

problem gets harder for all algorithms as the feasibility angle ϕ decreases. This is because the feasible region becomes smaller and the optimum is surrounded by a neighborhood containing more and more infeasible points. – Interestingly, also the gap between Schonlau and SOCU gets larger as ϕ decreases. This is understandable as well: For SOCU the probability of feasibility is 1 along both active constraint lines for all ϕ. For Schonlau the probability of feasibility is 0.25 at the optimum (2 active constraints, both 0.5). If ϕ is large, the probability quickly rises to 0.5 as we move along one of the active constraint lines, because the distance to the other line increases. But if ϕ is small and we move along one constraint line, the probability according to Schonlau stays longer near 0.25, because the second constraint line is not far away. Thus the solution found by Schonlau will be farther from the optimum since the maximum of the product $EI \cdot F$ moves farther into the feasible region. This is exactly what we see in Fig. 5.5.

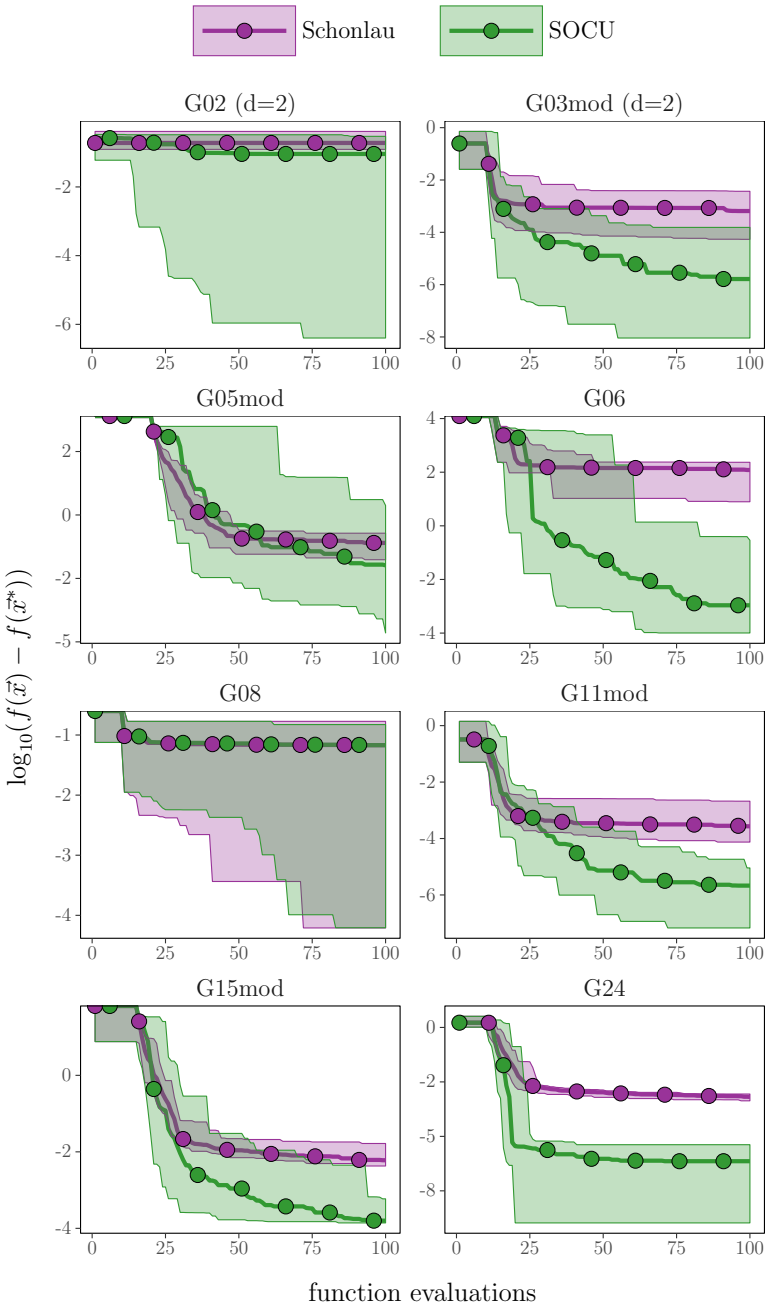

Figure 5.6: Comparing the performance of SOCU and Schonlau [159] on G-problems from Tab. 7.2. The solid curves show the median of the error for 30 independent trials. The error is calculated with respect to the true minimum $f(\vec{x}^*)$. The colored shade around the median is showing the worst and the best error.

5.6.2 Noise Variance

In our first experiments we experienced frequent crashes of the Kriging modeling software due to numeric instabilities. This is a well-known but cumbersome observation about Kriging shared by many researchers, especially if the modeling points are unevenly spaced, as it is inevitably the case in optimization tasks. To avoid too strong oscillations of the Kriging model due to nearby points, it is a common cure to switch from interpolating to approximating Kriging models, either with the so-called nugget-effect or with a noise variance parameter, which assumes a certain noise or uncertainty related with every modeling point. Since the nugget effect leads to a complicated structure for the variance $\sigma^2(x)$, it turned out to be not well-suited for our case.

The noise variance, on the other hand, turned out to be very effective: As Tab. 5.2 shows, the interpolating Kriging models had frequent crashes. By adding the noise variance $\nu = 0.01$ to the model, we could completely avoid any crashes in our experiments.

Table 5.2: Effect of noise variance.

	SOCU w/o noise variance		SOCU	
problem	crashed (%)	iteration	crashed (%)	iteration
G06	100	19	0	–
G02	96	40	0	–

5.6.3 Performance on G-Problems

In Fig. 5.6 we compare the two different variants of Kriging-based EGO, namely the original version of Schonlau et al. [159] and our SOCU algorithm [14], which we applied to all G-problems in Tab. 7.2. SOCU reaches lower optimization errors in most cases. In the case of G08, both algorithms have the same median curve. This is perfectly understandable, since G08 is the only problem without any active constraints. Absence of an active constraint is a convincing reason for similar performance of both algorithms, since the different feasibility functions (Fig. 5.1) have no effect. For problems with active constraints, the high value of SOCU's $F_i(x)$ at the border of the feasible region helps to find better solutions. In Fig. 5.7 we show additionally the effect of switching off the plugin control (Sec. 5.4.3) in SOCU. It can be seen that the plugin control is beneficial for G02, G03mod, G06 and G24.

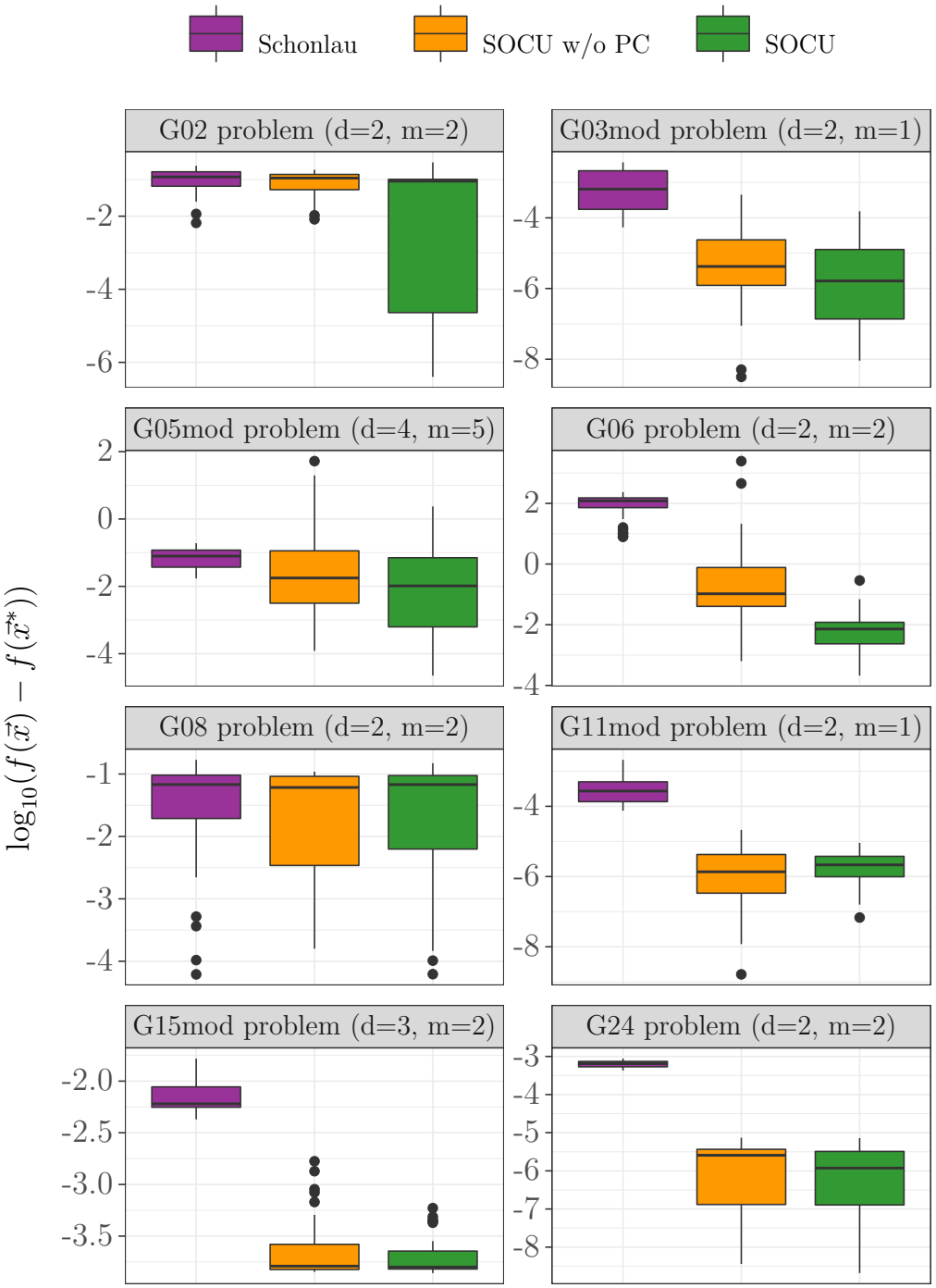

Figure 5.7: Comparing the final optimization error from 30 independent runs determined with different algorithms.

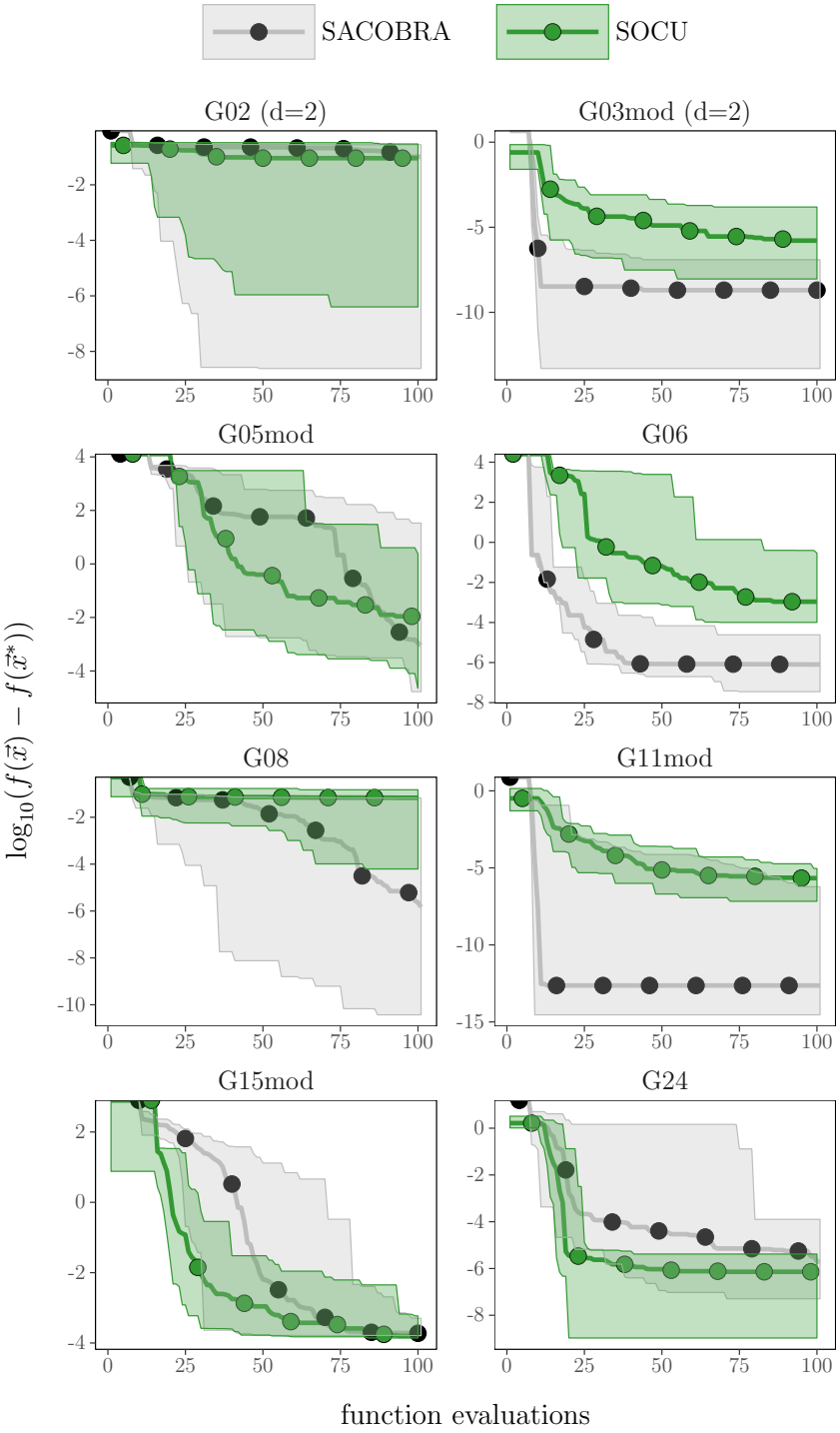

Figure 5.8: Comparing the performance of SOCU and SACOBRA on G-problems from Tab. 7.2. The solid curves show the median f the error for 30 independent trials. The error is calculated with respect to the true minimum $f(\vec{x}^*)$. The colored shade around the median is showing the worst and the best error.

We compare in Fig. 5.8 the results of SOCU with SACOBRA [15], to the best of our knowledge the current state-of-the-art for solving the G-problems. It is evident that SACOBRA is slightly better in most cases, except for the case of G24, where SACOBRA converges more slowly than SOCU. Additionally, three problems (G05mod, G15mod, G24) show a better performance of SOCU in the early stages (between iteration 25 and 75), although SACOBRA catches up at iteration 100.

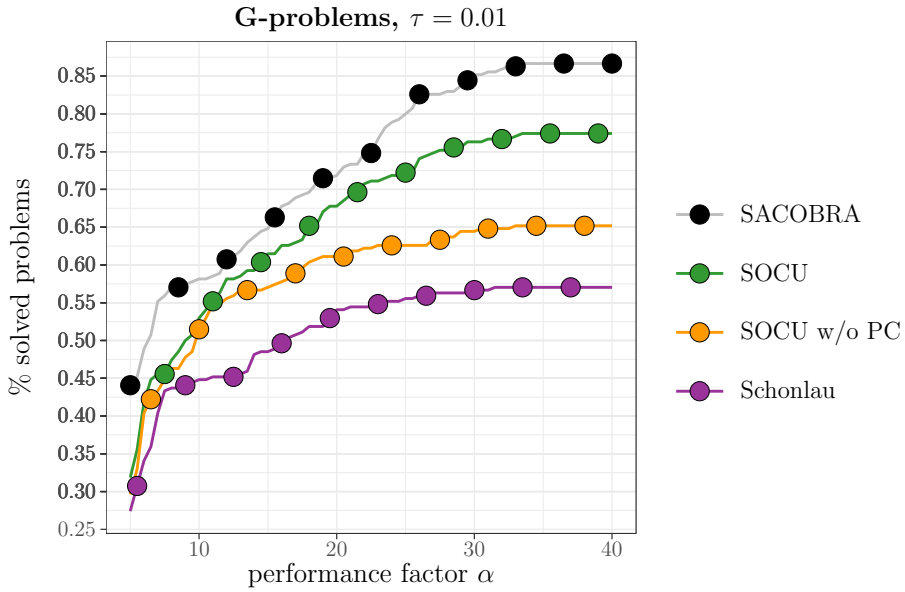

Figure 5.9: Data profile of SACOBRA, Schonlau, SOCU and SOCU w/o PC on G-problems and *Sphere4* The performance factor α is the budget divided by $d+1$ where d is the individual dimension of each test problem.

Finally Fig. 5.9 shows the overall performance comparison for all algorithms on all test problems in form of a data profile described in 2.5. The larger the data profile (ratio of solved problems), the better the relevant algorithm. The performance factor α on the x-axis is the number of iterations divided by $d + 1$.

5.6.4 Performance on XFOIL

Fig. 5.10 shows our results for the XFOIL case. SOCU is slightly better than Schonlau in the median, but the difference is not statistically significant. The similarity is understandable, since problem XFOIL has only one active constraint. In such a case we do not expect the differences between Schonlau and SOCU to be very large. SOCU and SACOBRA produce nearly identical results.

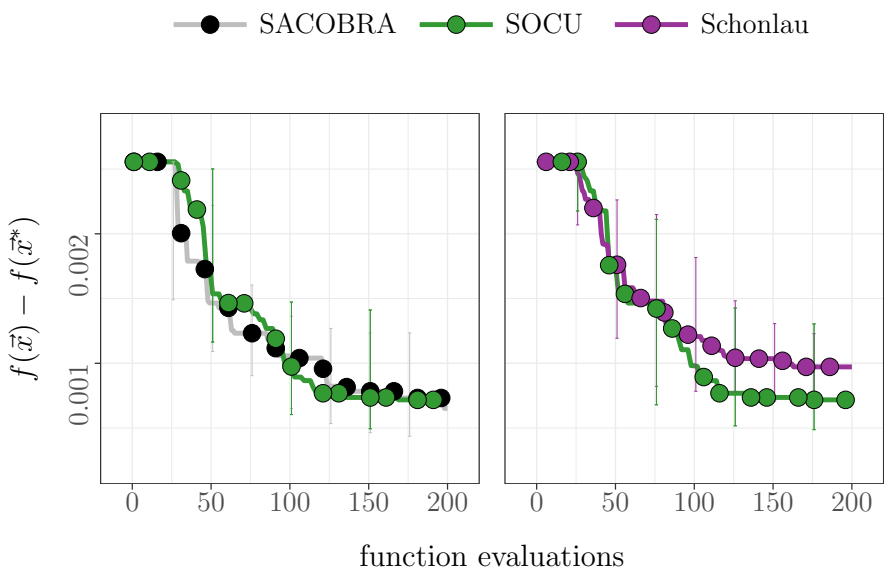

Figure 5.10: Comparing the performance of SOCU, Schonlau [159], and SACOBRA [15] on the XFOIL real world optimization problem (30 independent runs).

5.7 Conclusion

In order to answer the first research question **Q5.1**, the developed Kriging-based constrained optimizer was evaluated on a set of problems listed in Tab. 7.2. As shown in Fig. 5.9, SOCU is able to solve more than 75% of the problems with a limited number of function evaluations of 100. The problems have $d = 2 \dots 4$ and the number of active constraints vary from 0 to 3. Therefore, we can give a positive answer to **Q5.1**. The introduced algorithm could overcome the challenges associated with multiple active constraints in low dimension. However, we are aware of SOCU's limitation in handling COPs in higher dimensions.

The modified expected improvement suggested in Eq. (5.5) does not always direct the search towards the feasible region or even close to the feasible border. An example of EI_{mod} leading the search toward the infeasible region was showcased in Fig. 5.2. Plugin control is the proposed cure for balancing the exploration of feasible and infeasible infill points and redirecting the search toward the feasibility border. The 10% boost contributed by using the plugin control shown in Fig. 5.9 can be considered as a positive answer to **Q5.2**.

We applied Kriging surrogate models to constrained optimization. We could show that a small number of evaluations is sufficient to obtain good optimization results. This work is not the first to do so, but three important conclusions for Kriging-based optimization could be drawn in this chapter:

1. Interpolating Kriging models often suffer from numerical instabilities and subsequent crashes, especially when the population points are unevenly distributed in the search space. This will be nearly always the case when applying Kriging for optimization. We have shown that these crashes can be completely avoided (at least in our test cases) when we add a small noise variance to the Kriging models. This leads to approximating Kriging models and to a variance always larger than zero. Both effects are beneficial for numeric stability of the Kriging models.

2. Many Kriging-based COP-solvers use in one form or the other a product of expected improvement EI and probability of feasibility F. We could show that if the additive plugin in EI gets very small (which is no problem for the optima of EI themselves), this can have adversarial effects for the optima of $EI \cdot F$. A method called *plugin control* was proposed, which successfully counteracts such adversarial effects.

3. Having this plugin control in effect, we could show that a probability curve of SOCU being 1 at the border of feasibility is clearly superior to an approach where the probability is 0.5 at the border [159]. The benefits are - as expected - more clearly seen for problems with two or more active constraints (G05mod, G06, G15mod, G24).

SOCU was tested on a variety of benchmark problems with dimension of $d = 4$ and below. We could perform better than the Kriging-based algorithm of Schonlau et al. [159], but were in most cases worse than the non-Kriging-based SACOBRA algorithm [15]. In some cases (G05mod, G15mod, G24) SOCU showed a better performance (median, best and worst case) in early iterations (< 75) than SACOBRA. We suppose that this is due to the better exploration of the EGO approach. A drawback for Kriging-based models is however that they become very slow and ineffective for larger dimensions. In the future, a closer investigation into the causes of the observed performance differences would be desirable.