Constrained Optimization with a Limited Number of Function Evaluations

Patrick Koch[*], Samineh Bagheri[*], Christophe Foussette[⊕], Peter Krause[⊕], Thomas Bäck[⊕], Wolfgang Konen[*]

[*]Cologne University of Applied Sciences
Steinmüllerallee 1
51643 Gummersbach
E-Mail: {patrick.koch, wolfgang.konen}@fh-koeln.de

[⊕]divis intelligent solutions GmbH
Joseph-von-Fraunhofer-Str. 20
44227 Dortmund
E-Mail: {foussette, krause, baeck}@divis-gmbh.de

Abstract

In real-world optimization often constraints must be respected, restricting the number of feasible solutions. Therefore algorithms and strategies have been proposed to repair constraint-violating solutions or to avoid extensive search in infeasible regions. Such constraint handling methods are well-known from the literature, but most algorithms have the drawback that they require a large number of function evaluations. This can be especially problematic for real-world optimization tasks, which often incorporate expensive simulations. Up to now, only little work has been devoted to *efficient* constraint-based optimization (severely reduced number of function evaluations). A possible solution in that regard is to use *surrogate models* for the objective and constraint functions respectively. While the real function might be expensive to evaluate the surrogate functions are often much faster. Recently, as an example for this approach, the solver *COBRA* was proposed and outperforms most other algorithms in terms of required function evaluations on a large number of benchmark functions. In this paper we propose a new implementation of COBRA and compare it with other constraint-based optimization algorithms. We discuss the internal components of the algorithm and find that by adding new strategies, the algorithm can be significantly improved. We also report on negative results where COBRA still shows a bad behaviour and gives indications for possible improvements.

1 Introduction

Real-world optimization problems are often subject to constraints, restricting the feasible region to a smaller subset of the search space. It is the goal of most optimizers to avoid infeasible solutions and to stay in the feasible region, in order to converge in the optimum. However, the search in constraint black-box optimization can be difficult, since knowledge about the size of the feasible region and the location of the optima is usually unknown. This problem even turns out to be much harder, when only a limited number of function evaluations is allowed for the search. However, this is often the case in real-world optimization, where good solutions are requested in very restricted timeframes. In this paper we present a state-of-the-art solver for constraint-based optimization and discuss advantages and common pifalls of the method.

1.1 Related work

For constrained problems several repair methods have been proposed [1, 2], that aim to repair infeasible solutions. Another common approach is to incorporate static or dynamic penalty terms to stay in the feasible region [3, 4, 5]. Other techniques handle constraints by optimizing objective function and constraint functions separately in a lexical order [6, 7]. [8] is another example of this approach with stochastic ranking. Also multi-objective optimization algorithms have been designed for constraint-based optimization, considering the constraint functions as additional objectives [8, 9]. Beyer and Finck [10] propose an extension of CMA-ES which allows to handle special types of constraints successfully.

In the field of model-assisted optimization algorithms for constrained problems, Support Vector Machines (SVMs) have been used by Poloczek and Kramer [11]. They make use of SVMs as a surrogate of the objective function, but achieve only slight improvements. Powell [12] proposes COBYLA, a direct search method which models the objective and the constraints by linear functions. Recently, Regis [13] developed COBRA, an efficient solver that makes use of Radial Basis Function (RBF) interpolation, and outperforms most algorithms in terms of required function evaluations on a large number of benchmark functions.

In this paper we analyze a new implementation of COBRA in `R`, which allows easy adaptation of certain components of the algorithm. In Sec. 2 we present the problem and the algorithm in more detail. In Sec. 3 we perform a thorough experimental study on analytical test functions and on a real-world benchmark function. The results are discussed with regard to specific parameter settings of the algorithm in Sec. 4 and we give conclusive remarks in Sec. 5.

2 Methods

2.1 Constrained-based optimization

A constrained optimization problem can be defined by the minimization of a real-valued objective function f subject to constraint functions $s_1, \ldots, s_m$:

$$\begin{array}{ll} \text{Minimize} & f(\vec{x}), \vec{x} \in \mathbb{R}^d \\ \text{subject to} & \\ & s_i(\vec{x}) \leq 0, i = 1, 2, \ldots, m \end{array}$$

In this paper we always consider minimization problems. Maximization problems can be transformed to minimization problems without loss of generality.

2.2 Radial Basis Functions

The COBRA algorithm incorporates optimization on auxiliary functions, e.g., a regression model of the search space. Although numerous regression models are available therefore, we employ interpolating RBF [14, 15], since they outperformed other models. In this paper we use the same notation like Regis [16]. The RBF model requires a set of design points (a training set) as input: n points $\vec{u}^{(1)}, \ldots, \vec{u}^{(n)} \in \mathbb{R}^d$ are evaluated on the real function $f(\vec{u}^{(1)}), \ldots, f(\vec{u}^{(n)})$. We use an interpolating radial basis function as approximation:

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

Here, $|| \cdot ||$ is the Euclidean norm, $\lambda_i \in \mathbb{R}$ for $i = 1, \ldots, n$, $p(\vec{x})$ is a linear polynomial in d variables, and φ is of cubic form $\varphi(r) = r^3$. Although other choices for φ are possible and have been tested in related work, cubic RBF have been shown to be superior in [17].

Fitting of the model can be done by defining a distance matrix $\Phi \in \mathbb{R}^n$: $\Phi_{i,j} = \varphi(||\vec{u}^{(i)} - \vec{u}^{j}||), i, j = 1, \ldots, n$. The cubic RBF model is obtained by solving the linear system of equations:

$$\begin{bmatrix} \Phi & P \\ P^T & 0_{(d+1)\times(d+1)} \end{bmatrix} = \begin{bmatrix} \lambda \\ \vec{c} \end{bmatrix} \times \begin{bmatrix} F \\ 0_{d+1} \end{bmatrix} \tag{2}$$

where $0_{(d+1)\times(d+1)} \in \mathbb{R}^{(d+1)\times(d+1)}$ is a zero matrix, $F = (f(\vec{u}^{(1)}), \dots, f(\vec{u}^{(n)}))$, 0_{d+1} is a vector of zeros, $\lambda = (\lambda_1, \dots, \lambda_n)^T \in \mathbb{R}^n$ and $\vec{c} = (c_1, \dots, c_{d+1})^T \in \mathbb{R}^{d+1}$ are the coefficients of the linear polynomial $p(\vec{x})$. The matrix in Eq. (2) is invertible if it has full rank. This is usually the case, if $d + 1$ linearly independent points are provided. The matrix inversion can be efficiently calculated by using singular value decomposition (SVD) or similar algorithms.

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

2.3 Constrained Optimization by Radial Basis Function Approximation

Constrained Optimization by Radial Basis Function Approximation (COBRA) is an optimization algorithm proposed by Regis [13]. The main idea of this method is to use approximations of both the objective function and constraint functions, in order to save evaluations of the real function and constraints. Internally COBRA uses RBF interpolation for the modeling of the objective and constraints. Each iterate is a result of an optimization on a subproblem, which is defined by the RBF interpolation models of the objective and the constraint functions.

Fig. 1 presents a flowchart of the algorithm. In the beginning an initial population or design is generated to make it possible to create the first RBF model. This can be done by using various strategies which are described in more detail in Sec. 3.3. The resulting RBF models from the initial design are used to find the next iterate to be evaluated on the real function. Therefore, a sequential search is performed on the surrogate functions. The best point obtained from this search is referred to as *infill point* in the remainder of this paper. Note that the infill point is the only point that is also evaluated on the real function. This makes the algorithm efficient in terms of real function evaluations required. If the infill point is better than the current best solution, the best solution is replaced by the infill point. In any case the RBF models are updated using the new information. In the next round again a sequential search is performed until the number of function evaluations exceeds the maximum number of allowed evaluations given by the user (the budget for the optimization).

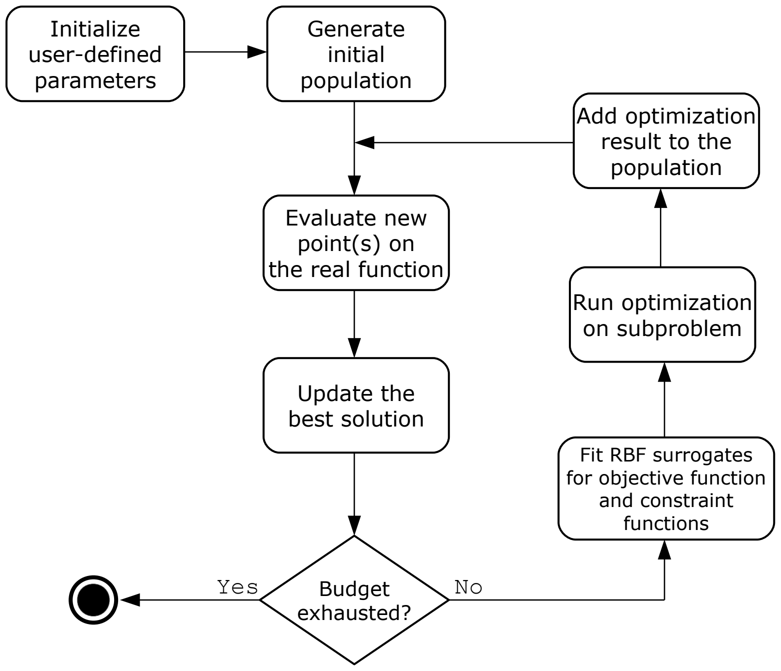
Initialize user-defined parameters
Generate initial population
Add optimization result to the population
Evaluate new point(s) on the real function
Run optimization on subproblem
Update the best solution
Fit RBF surrogates for objective function and constraint functions
Yes
Budget exhausted?
No

Figure 1: Flowchart of the COBRA algorithm.

Model-assisted optimization In each iteration COBRA performs an optimization on the RBF models. This can be done by either using a special constraint solver, or by using a method for unconstrained optimization with a penalty function to avoid infeasible solutions. In this paper we incorporate two constraint solvers, COBYLA [12] and ISRES [8], and also make use of unconstrained optimization methods in form of classical Hooke & Jeeves pattern search [18] and the simplex algorithm by Nelder & Mead [19]. The selection of the internal optimization strategy in COBRA is arbitrary and must be defined by the user. In Sec. 3.4 we compare different optimization strategies on a real-world problem.

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

Uncertainty of constraint predictions COBRA 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 ϵ is used to handle wrong predictions of the constraint surrogates. In the beginning we set $\epsilon_{init} = 0.005 \cdot l$, where l is the diameter of the search space. In each iteration n we only claim the point to be feasible if the following Eq. holds for all constraint surrogates $s_i^{(n)}$ with $i = 1, \dots, m$:

$$s_i^{(n)} + \epsilon_i^{(n)} \leq 0 \tag{3}$$

That is, we tighten the constraints by adding the factor ϵ which is adapted during the search. The ϵ-adaptation is done by counting the feasible and infeasible infill points C_{feas} and C_{infeas} over the last iterations. When the number of these counters reaches the threshold for feasible or infeasible solutions, T_{feas} or T_{infeas}, respectively, we divide or double ϵ by 2 (up to a given maximum). When ϵ is decreased, solutions are allowed to move closer to the 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 a while (T_{infeas}), the ϵ factor is increased in order to keep the points further away from the constraint boundary.

3 Experimental analysis

3.1 Benchmark functions

For evaluation we use popular benchmark functions, e.g., the G functions described in [20]. In Tab. 1 we present characteristics of these functions. It can be seen from the table that the functions differ in dimension, objective and number and type of constraints. Since these functions are often used in the scientific literature for analyzing constrained-based solvers, they provide a good analytical testbed for our algorithm.

Additionally we evaluate the algorithm on a high-dimensional real-world problem from the automotive industry: the MOPTA 2008 benchmark by

Table 1: Characteristics of G functions: d: dimension, ρ: percent feasible, R: range of objective function, LI: the number of linear inequalities, NI: number of nonlinear inequalities, NE: the number of nonlinear equalities

Fct.	d	type	ρ	R	LI	NI	NE
G01	13	quadratic	0.0003%	293.87	9	0	0
G02	20	nonlinear	99.9973%	0.69	1	1	0
G03	10	nonlinear	0.0026%	1	0	0	1
G04	5	quadratic	27.0079%	9725.83	0	6	0
G05	4	nonlinear	0.0000%	8850.43	2	0	3
G06	2	nonlinear	0.0057%	1247439.40	0	2	0
G07	10	quadratic	0.0001%	5660.62	3	5	0
G08	2	nonlinear	0.8581%	1691.26	0	2	0

Jones [21] is a 124-dimensional problem with 68 constraints. All input parameters have been normalized to $[0, 1]$. The constraint values are meaningfully scaled, e.g., if a constraint value s_i of 0.05 is returned, this means that the constraint boundary is violated by a percentage of 5%. The problem should be solved within $1860 = 15 \cdot d$ function evaluations which in reality refers to one month of computation time on a high-performance computer.

3.2 Results on benchmark functions

We compare our COBRA implementation in `R`[1] with the results from other research articles. Tab. 2 shows the results of 30 independent runs on the G functions introduced in Sec. 3.1. For comparison we present the results from COBRA by Regis [13], the stochastic ranking evolution strategy (ISRES) by Runarsson and Yao [8] and the Repair Genetic Algorithm (RGA) by Chootinan and Chen [2]. The results from COBRA by Regis [13], ISRES [8] and RGA [2] have been taken from the original articles of the authors, for COBYLA we ran experiments using the nloptr package in `R`.[2]

The results of our COBRA implementation can achieve accuracies similar or close to the results of the evolutionary algorithms (EA) ISRES and RGA. This already can be seen as a success, since it was not clear, if the surrogate models in COBRA are able to find very good approximations of the real function and constraints. However, with exception of function G02, where our COBRA implementation performed poorly, the results are

[1]`http://cran.r-project.org/`

[2] `http://cran.r-project.org/web/packages/nloptr/nloptr.pdf`

Table 2: Best (b), median (m) and worst (w) results and their standard deviation (sd) determined in 30 independent runs with different approaches.

Fct.	Optimum		COBRA-R	COBRA [13]	ISRES [8]	RGA 10% [2]	COBYLA [12]
G01	-15.00	b	-15.00	NA	-15.0	-15.0	-15.0
		m	-15.00	NA	-15.0	-15.0	-13.83
		w	-13.00	NA	-15.0	-15.0	-0.27
		sd	0.58	NA	5.8e-14	0.0	1.30
G02	-0.80355	b	-0.409403	NA	-0.803619	-0.801119	-0.272
		m	-0.346592	NA	-0.793082	-0.7857	-0.199
		w	-0.281917	NA	-0.723591	-0.745329	-0.164
		sd	0.028	NA	2.2e-02	-0.0137	0.023
G03	-1.0	b	-0.9899	-0.8965	-1.001	-0.9999	-1.0
		m	-0.9753	0.00	-1.001	-0.9999	-0.2289
		w	0.000	0.00	-1.001	-0.9997	0.0
		sd	0.19	NA	0.0	0.0	0.45
G04	-30665.539	b	-30665.5386	-30665.49	-30665.539	-30665.5386	-30665.539
		m	-30665.5386	-30665.15	-30665.539	-30665.5386	-30665.539
		w	-30665.5386	-30664.58	-30665.539	-30665.5386	-30665.539
		sd	7.5e-05	0.04	1.1e-11	0.0	8.3e-09
G05	5126.498	b	5126.4981	5126.5	5126.497	5126.498	5126.498
		m	5126.4981	5126.51	5126.497	5126.498	5126.498
		w	5126.4989	5126.53	5126.497	5126.498	5126.498
		sd	2.4e-04	0.0	7.2e-13	0.0	0.0
G06	-6961.8138	b	-6961.8134	-6944.54	-6961.81	-6961.81	-6961.81
		m	-6961.8116	-6795.6	-6961.81	-6961.81	-6961.81
		w	-6961.8044	-6460.53	-6961.81	-6961.81	91.05
		sd	1.5e-2	24.6	1.9e-12	0.0	1782.09
G07	24.306	b	24.306	24.48	24.306	24.329	24.306
		m	24.306	24.306	24.306	24.472	24.306
		w	24.309	29.33	24.306	24.835	1440.87
		sd	6.6e-04	0.15	6.3e-05	0.13	343.91
G08	-0.0958250	b	-0.0958250	-0.10	-0.0958	-0.0958	-0.0958
		m	-0.0957808	-0.09	-0.0958	-0.0958	-0.0272
		w	-0.0945741	-0.06	-0.0958	-0.0958	0.0
		sd	2e-04	0.0	2.7e-17	0.0	0.02

in line with the EA and also outperform the original COBRA algorithm in Matlab published by Regis [13] on some functions. While in general the implementations are similar to each other, the following differences can be made responsible for this:

- internal optimizer in COBRA (Regis uses `fmincon` in Matlab, we use COBYLA, ISRES, NMKB or HJKB in `R`)
- size of the initial design. Regis always uses $d+1$ points, whereas we also set higher values up to $3 \cdot d + 1$
- initial design type: Regis proposes random initial design, we allow LHS, optimized and biased designs (Sec. 3.3)
- usage of *repair infeasible* method (Sec. 3.5), for repairing slightly infeasible solutions

As Tab. 2 only reports the objective values, we want to indicate that COBRA has been designed for complex real-world optimization, and one of its main advantages is that optimization can be performed spending only very few function evaluations. Because COBRA makes use of internal surrogate models, it usually requires only a fraction of the function evaluations needed by other strategies such as the EA. In Tab. 3 we present the number of real function evaluations required by the methods to achieve the objective values in Tab. 2. It is clearly visible that both our COBRA implementation in `R` and the original COBRA implementation in Matlab need only very few function evaluations, while algorithms like ISRES or RGA sometimes require 1000 times more evaluations to yield similar results.

Comparing COBRA and COBYLA, it can be seen from the result tables that COBRA requires less function evaluations. One reason therefore might be that COBYLA does not incorporate any additional heuristics as the distance requirement and only uses linear models which might be worse for nonlinear functions. Another disadvantage of COBYLA can be the dependency of the delivered starting point. While COBRA uses a set of points, COBYLA is prone to early convergence in local optima when the problem is multimodal and the starting point is far from the optimum. As a consequence the final solutions of COBYLA are not as precise as the solutions delivered by COBRA and COBYLA needs substantially more function evaluations until convergence (cf. Tab. 3).

Table 3: Average number of function evaluations used to reach results shown in Tab. 2 for different algorithms.

	COBRA-R	COBRA [13]	ISRES	RGA 10%	COBYLA
G01	59	NA	350 000	95 512	986.4
G02	500	NA	350 000	331 972	5 000.0
G03	500	100	350 000	399 804	3 402.2
G04	100	100	350 000	26 981	441.3
G05	100	100	350 000	39 459	745.0
G06	100	100	350 000	13 577	276.6
G07	150	100	350 000	428 314	2 740.0
G08	150	100	350 000	6 217	430.0

3.3 Initial design

The initial design strategy in COBRA (Sec. 2.3) can be responsible for obtaining good or bad results. Especially for multimodal problems a good selection of the initial design can lead to meaniningful better results. In the R implementation of COBRA the user can select between three different initialization strategies:

LHS: Latin Hypercube Sampling of size n

Biased: The points are randomly sampled around the provided starting solution with given standard deviation sd.

Optimized: An initial optimization is performed without model-assisted optimization. In this paper we used Hooke & Jeeves [18] with a static penalty function for this.

In Fig. 2 we provide the results of 20 runs with COBRA on the G07 test function, first with a LHS random initialization and second with an optimized initialization. As can be seen from the plot the random LHS initialization leads to a premature convergence with no improvement after some early iterations. Instead the optimized initialization, where a Hooke & Jeeves search is performed using a simple penalty function, leads to much better progress and no stagnating behaviour. As a consequence, the selection of the initial design has a direct effect on the performance. A very bad selection of initial design points seems to completely deteriorate the algorithm's progress.

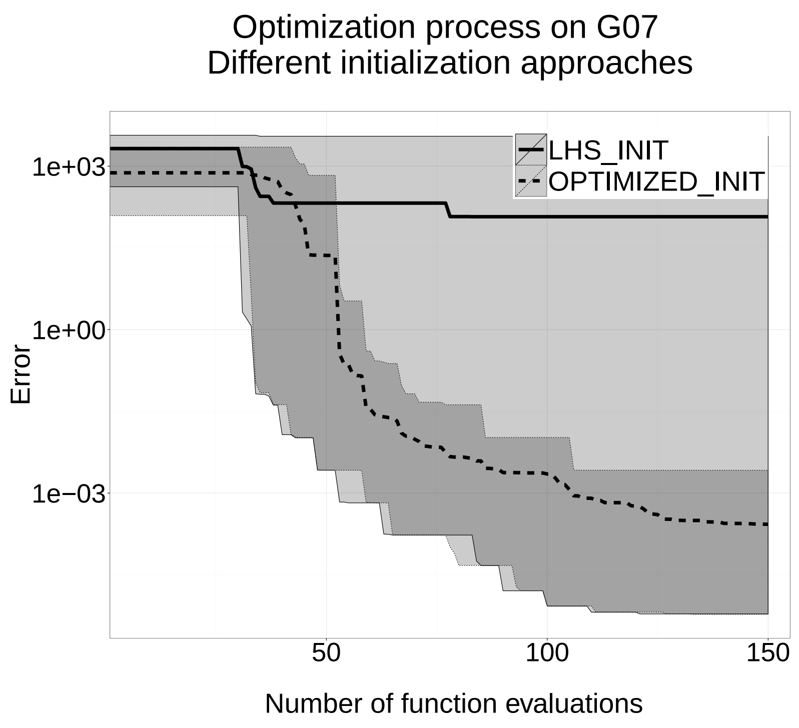

Figure 2: The plot shows the mean results of 30 independent runs on G07 for the LHS and optimized initialization strategies. The upper and lower ribbons depict best and worst solutions of the 30 runs.

3.4 Internal optimization strategy

We ran COBRA on the MOPTA benchmark problem with different optimizers for the optimization on the surrogate functions. E.g., in our COBRA `R` implementation we can select between the following optimization strategies:

- Hooke & Jeeves (HJKB) search [18]
- Nelder & Mead simplex algorithm (NMKB) [19],
- Constrained-based optimization by linear approximation (COBYLA) by Powell [12]

We assumed to get the best results with COBYLA, since it builds an internal model of the objective and constraints and usually outperforms the classical direct search strategies which are sometimes even not designed for high-dimensional problems (e.g., Nelder & Mead tends to get lost in large search spaces). In the left plot of Fig. 3 the mean out of ten runs of the best feasible solution visited over the optimization loop so far is depicted. In the initialization one feasible starting solution was given by

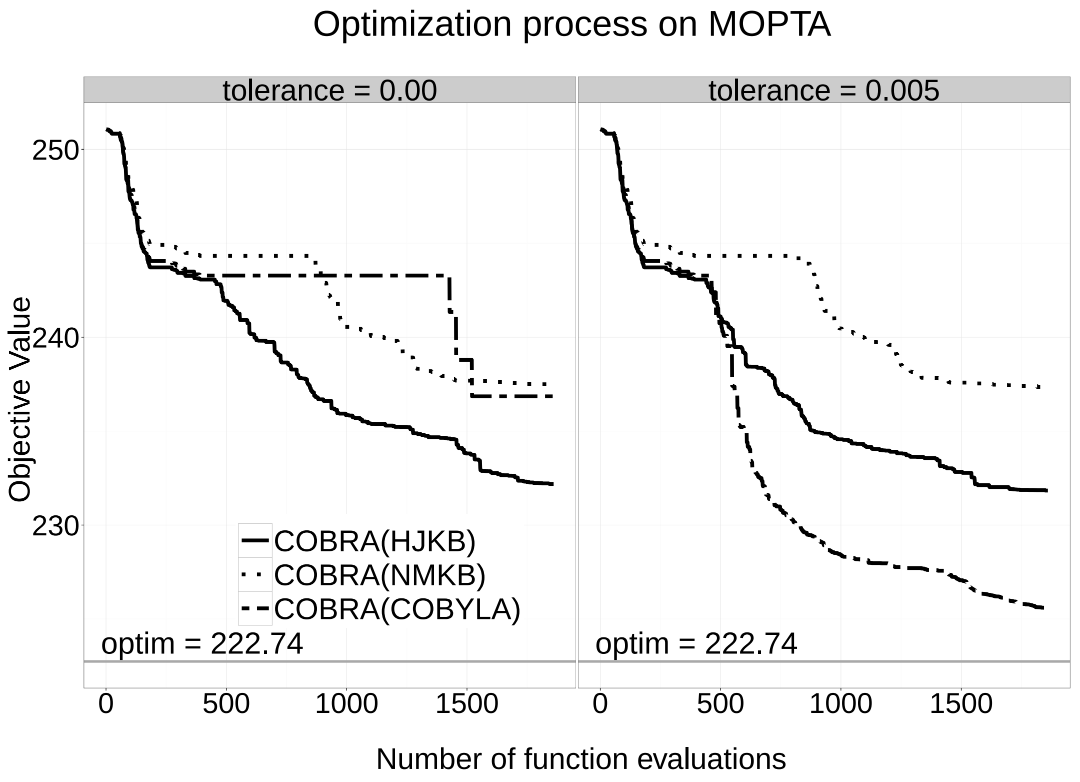

Figure 3: Average optimization progress for COBRA-R on the MOPTA 2008 benchmark. Left: best feasible solution up to the current iteration, right: best solution with 5% constraint violation (tolerance) allowed. The known optimum is shown as a straight grey line at 222.74.

Jones [21]. The rest of the 249 initial design points was generated by the *optimized* initialization strategy (cf. Sec. 3.3) using a size of 249 design points. It can be seen from the plot that surprisingly HJKB outperforms NMKB and COBYLA.

Interestingly, the situation changes if we allow for small constraint violations of up to 5%. Although infeasible solutions can be generated by this, it is possible that small constraint violations can be resolved afterwards, e.g., by applying a method similar to the *repair infeasible* algorithm we are presenting in Sec. 3.5.

In the right plot of Fig. 3 we again started the algorithms HJKB, NMKB and COBYLA on the MOPTA benchmark, but now allowing constraint violations of up to 5%. As can be seen from the plot the result of the COBYLA algorithm has been improved significantly. While COBYLA performed rather poor before with the hard constraint, it can now reach the optimum of the MOPTA benchmark in several runs. While the classical HJKB and NMKB did not yield any benefits from this small change, COBYLA could clearly improve its result and is close to the desired optimum.

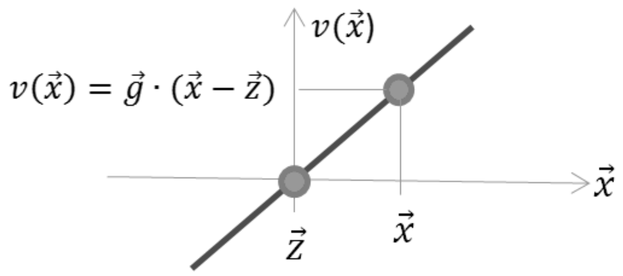

Figure 4: Repairing infeasible solutions with a gradient step. A slightly infeasible solution $\vec{x}$ is repaired with the help of the *surrogate function* $v(\vec{x})$ of a constraint (i. e. we do not need any expensive evaluations of the true function): We estimate the gradient $g(\vec{x})$ of the constraint surrogate. If the linear approximation near $\vec{x}$ holds, an appropriate step from $\vec{x}$ to a feasible solution $\vec{z}$ can be easily calculated.

3.5 Repairing infeasible solutions

Sometimes very small constraint violations occur for infill points, because the internal optimization method could not determine a feasible solution, or the solution returned by the optimizer was assumed to be feasible on the surrogates, but turns out to be infeasible after evaluation on the real function. As a consequence this can lead to a unwanted discarding of good but slightly infeasible solutions.

The *repair infeasible* method has been especially designed for internal optimization strategies inside COBRA like COBYLA. In most of our runs with COBYLA, it turned out that the infill points often have very small constraint violations. For COBYLA, it took a relatively long time to resolve such small constraint violations. To circumvent this issue we integrated a very simple gradient descent strategy for resolving small constraint violations. In Fig. 4 we describe the general sketch of the *repair infeasible* algorithm. The repair infeasible technique in our approach and [2] are both based on utilizing the gradient information from constraints. In [2], the repairing operation is embedded into a Genetic Algorithm and the infeasible results are repaired with a defined probability by deriving the gradient of the real constraint functions to direct the infeasible point towards the feasible region. Our approach only relies on constraint surrogates and does not impose any extra real function evaluations.

In Fig. 5 we show the performance of COBRA on function G06 with and without *repair infeasible*. It can be seen from the plot that after a first phase with similar progress of both variants the variant with activated repair infeasible can approximate the optimum much better. The progress is very fast with neat improvements during iterations 40 to 50. In contrast the variant without *repair infeasible* stagnates at an inferior level.

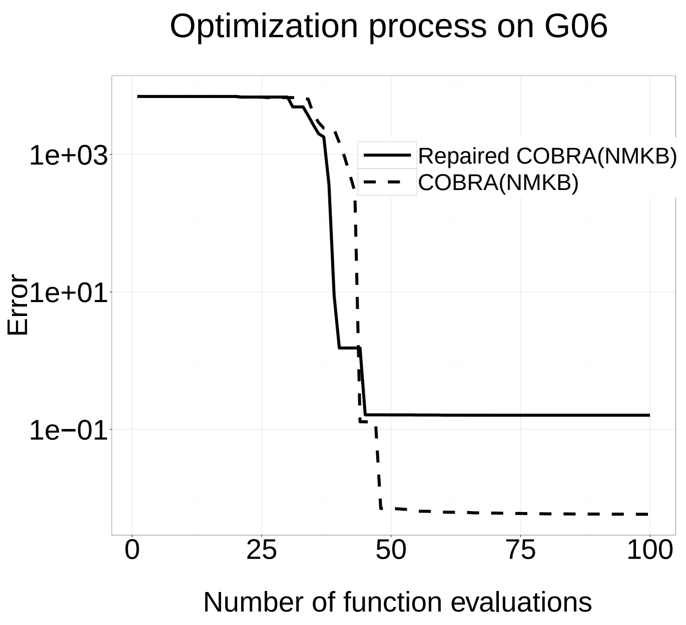

Figure 5: Impact of using the repair infeasible technique within COBRA(NMKB) on the G06 problem.

4 Discussion

In this section we discuss the components of the COBRA algorithm and want to draw the attention on common pitfalls and drawbacks of the method.

4.1 Initial design

The RBF models used internally in COBRA need at least $d+1$ points (where d is the dimension of the search space) to fit an interpolating model. In the COBRA algorithm an initial population is created as a first step (see Fig. 1). The method used for this can be designed by the user, but usually methods from statistics are chosen therefore. We have integrated the strategies described in Sec. 3.3, but depending on the problem also other designs are possible. Although with $d+1$ a lower bound exists for the initial design points and can be used by the algorithm, it can be advantageous to increase this number and to not rely on the minimum number of initial design points.

In our experimental study it was crucial to find a good strategy for the initial design generation and number of design points. We found these settings to be important for the later performance of the whole optimization run. Both settings can be very problem-dependent, but we showed several examples, where the selection of the right strategy makes a difference. A general rule-of-thumb for the number of initial design points can be given with $3d$, where d is the dimension of the problem.

4.2 Internal optimization strategy

The selection of the internal optimization strategy in COBRA can be crucial to find good solutions for some problems. E.g., in highly multimodal landscapes, local optimization methods such as COBYLA, HJKB or NMKB are not well-suited. In fact these methods are designed for local optimization and will probably fail on multimodal landscapes. A possible solution to this problem can be to implement a restarting strategy, which from time to time makes random restarts and does not spent the whole budget for the optimization of the starting solution. Other solutions can be to select global optimization strategies like ISRES, which are also available in our `R` implementation of COBRA.

4.3 Distance requirement

In the literature a lot of work has been devoted to balancing exploration and exploitation of the search space. In COBRA, the user controls exploration and exploitation by setting the distance requirement parameter Ξ. Large values in Ξ lead to explorative steps, while smaller values enable closer approximations of the optima. However, too many large values can lead to uncontrolled jumping through the search space, whereas too many small values can lead to a stagnating behaviour in suboptimal regions of the search space. For this reason good settings of Ξ are necessary and must be defined anew for each test case. As a rule of thumb, Regis [13] proposes $\Xi_{local} = \langle 0.01, 0.001, 0.0005\rangle$ for a locally-biased distance requirement, and $\Xi_{global} = \langle 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005\rangle$ for a more globally biased distance requirement. Note that these settings are only suggestions, specific properties of other problems might require to define other settings.

In our experiments we sometimes discovered, that a large element (i.e. $\xi_i = 0.03$) or a very small element ($\xi_i = 0.0$) in the set can have a beneficial effect. The large element can support the algorithm to make larger steps in the search space, which can be advantageous for highly multimodal functions. The small element instead can help to find better approximations of the target. Fig. 6 shows the infill solutions obtained in a run on function G06 with a small ξ_i of 0.0 included. Stepwise the points move closer to the

Table 4: Challenges of G-problems and MOPTA (MO) and their possible solutions in COBRA

	Challenge(s)	**Solution(s)**
G01	Small feasible region. Often solutions with slightly violated constraints.	Add 0.3 to DRC (exploration). Use repairInfeasible.
G02	Multimodal: Many local optima, especially in 20d	none!
G03	Nonlinear and non-separable objective $\rightarrow$ surrogate model not accurate. High dimension and large range R.	Logarithmic transform
G04	Fitness function and constraints with mixed terms $x_1x_2 \rightarrow$ difficult for constraint surrogates.	Use optimizer COBYLA (others fail: NMKB, ISRES)
G05	Extremely thin feasible region. Three nonlinear active constraints. Highly varying input ranges.	Add 0.0 to DRC, use optimizer COBYLA. Rescale inputs to $[0,1]^d$.
G06	Very thin feasible region, optimum at „tip of needle“. Steep objective function, large range R.	Add 0.0 to DRC (avoid „blocking“ the optimum)
G07	Constraints with mixed terms $x_1x_2 \rightarrow$ difficult for constraint surrogates. Very small ρ = 0.0001%.	Use optimizer COBYLA (others fail: NMKB, ISRES)
G08	Shallow optimum in feasible region is masked by high $(+/-)$ infeasible peaks	Use optimizer ISRES (others fail: NMKB, COBYLA)
MO	Very high d=124 and m=68. Often solutions with slightly violated constraints.	

DRC: distance requirement cycle

R: min-max spread of objective function over search space (see Table 1)

ρ: percentage of feasible volume in search space volume (see Table 1)

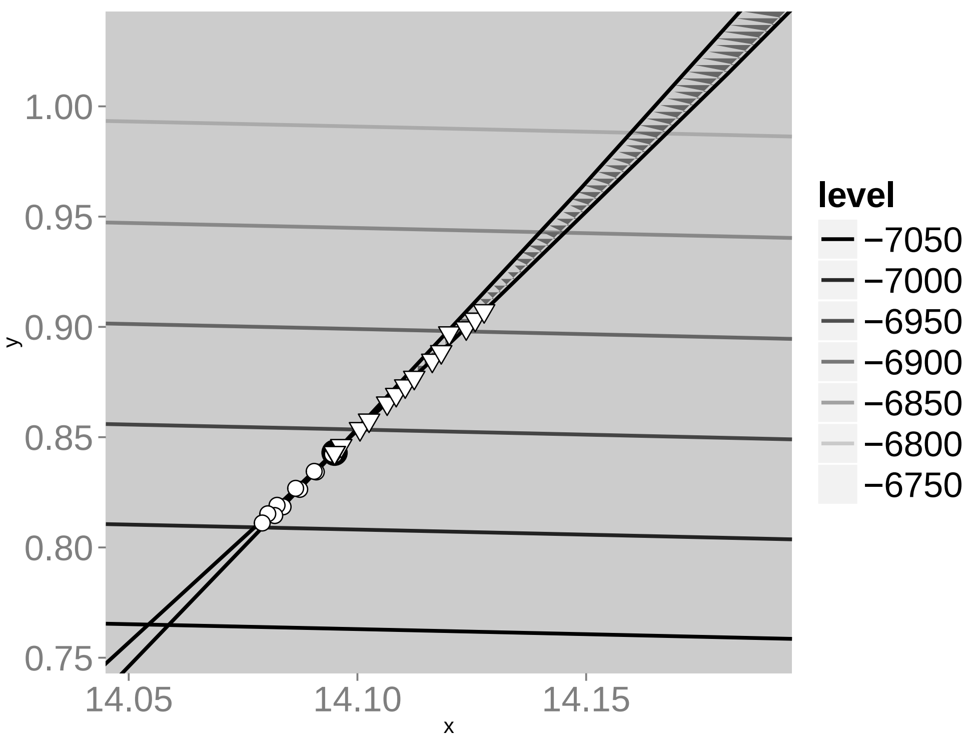

Figure 6: Optimization with COBRA-R on function G06 **with** $\xi_i = 0.0$ in the set. Triangles and circles are representing feasible and infeasible points. The big black circle is the true optimum, resp. The hatched area is the approximation of feasible region.

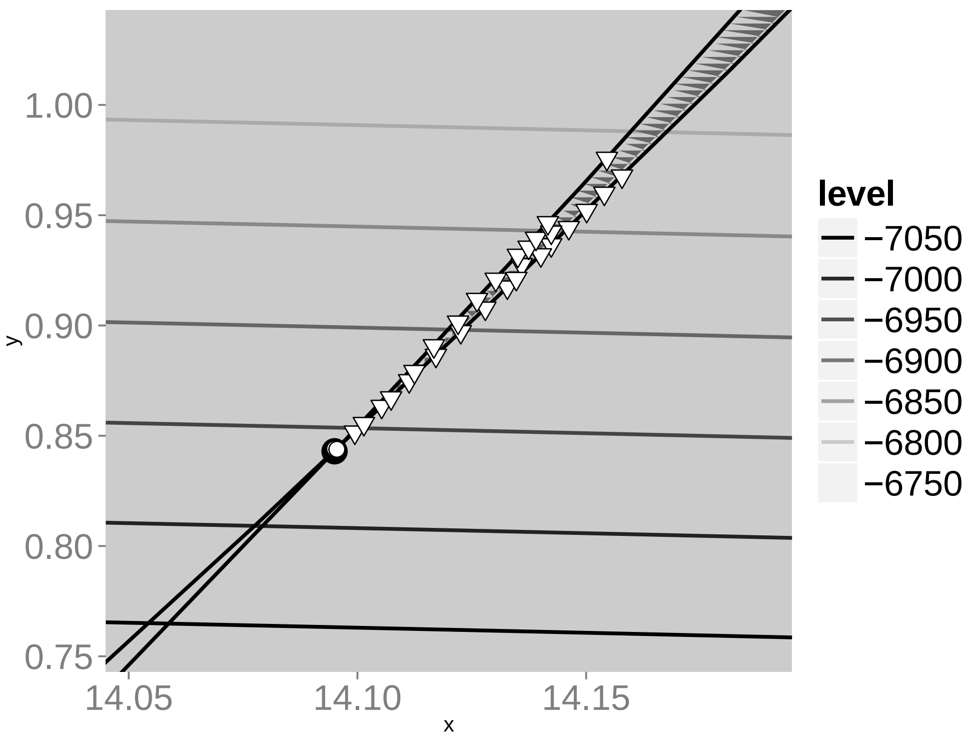

Figure 7: Optimization with COBRA-R on function G06 **without** $\xi_i = 0.0$ in the set.

optimum along the shaded area, which is located at the needle point of this area. Outside the shaded area the points are infeasible which makes this problem very hard for any optimizer, because the feasible region around the optimum becomes very small. With larger values for the Ξ parameter, it would not be possible to exploit solutions in the optimal region. By switching off the distance requirement method from time to time, the algorithm allows solutions in the direct neighbourhood of the known points and can move closer to the real optimum.

In Table 4 we summarize this discussion by showing the quite different challenges posed by each optimization problem and indicate possible solutions found in the framework of COBRA-R to cope with this challenges.

5 Conclusion and outlook

In this paper we presented a comparative study on constrained optimization problems under limited budgets. Therefore we developed a new implementation of the model-assisted algorithm COBRA in R. With our new implementation of COBRA we give users full flexibility for changing or adapting single components of the algorithm. In an experimental study on common benchmark functions we give evidence that our COBRA implementation can reach very good accuracies on most of the functions discussed in this paper (G01, G04, G05, G06, G07 and G08). The experiments showed, that the obtained results of COBRA-R are similar or close to the results of other constrained optimization algorithms including ISRES, RGA and COBYLA. We want to emphasize that the other strategies require a much higher number of function evaluations. Thus, as one of the main contributions of this paper, COBRA can achieve much faster convergence rates with almost similar accuracies on most functions. As only drawback we found negative results on the functions G02 and G03, presumably due to the dimensionality and complexity of these functions ($20d$ and multimodality in the case of G02, highly nonlinear in the case of G03).

Few questions remain open, e.g., the generation of a good initial design, the optimal settings for the distance requirement, or the choice of the best performing optimization strategy on the surrogate functions. In future work we want to elaborate on that and strengthen the advantages of model-assisted optimization under heavily restricted budgets. We are looking forward to improve our promising initial results on the MOPTA real-world problem by providing a deeper analysis of the hyperparameters of the algorithm.

References

[1] Oyman, A.; Deb, K.; Beyer, H.-G.: An alternative constraint handling method for evolution strategies. In: *Proceedings of Congress on Evolutionary Computation (CEC)*, S. 612–619. 1999.

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

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

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

[5] Coello Coello, C.: Constraint-handling techniques used with evolutionary algorithms. In: *Proceedings of the 14th International Conference on Genetic and Evolutionary Computation Conference (GECCO)*, S. 849–872. ACM. 2012.

[6] Takahama, T.; Sakai, S.: Constrained optimization by applying the alpha-constrained method to the nonlinear simplex method with mutations. *IEEE Trans. Evolutionary Computation* 9 (2005) 5, S. 437–451.

[7] Takahama, T.; Sakai, S.: Learning fuzzy control rules by alpha-constrained Simplex method. *Systems and Computers in Japan* 34 (2003) 6, S. 80–90.

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

[9] Aguirre, A. H.; Rionda, S. B.; Coello Coello, C. A.; Lizárraga, G. L.; Montes, E. M.: Handling constraints using multiobjective optimization concepts. *International Journal for Numerical Methods in Engineering* 59 (2004) 15, S. 1989–2017.

[10] Beyer, H.-G.; Finck, S.: On the design of constraint covariance matrix self-adaptation evolution strategies including a cardinality constraint. *IEEE Transactions on Evolutionary Computation* 16 (2012) 4, S. 578–596.

[11] Poloczek, J.; Kramer, O.: Local SVM Constraint Surrogate Models for Self-adaptive Evolution Strategies. In: *KI 2013: Advances in Artificial Intelligence*, S. 164–175. Springer. 2013.

[12] Powell, M.: A Direct Search Optimization Method That Models The Objective And Constraint Functions By Linear Interpolation. In: *Advances In Optimization And Numerical Analysis*, S. 51–67. Springer. 1994.

[13] Regis, R.: Constrained optimization by radial basis function interpolation for high-dimensional expensive black-box problems with infeasible initial points. *Engineering Optimization* 46 (2013) 2, S. 218–243.

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

[15] Powell, M.: The Theory of Radial Basis Function Approximation in 1990. *Advances In Numerical Analysis* 2 (1992), S. 105–210.

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

[17] Wild, S.; Shoemaker, C.: Global Convergence Of Radial Basis Function Trust-Region Derivative-Free Algorithms. *SIAM Journal on Optimization* 21 (2011) 3, S. 761–781.

[18] Hooke, R.; Jeeves, T.: “Direct Search” Solution of Numerical and Statistical Problems. *Journal of the ACM (JACM)* 8 (1961) 2, S. 212–229.

[19] Nelder, J.; Mead, R.: A simplex method for function minimization. *The Computer Journal* 7 (1965) 4, S. 308–313.

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

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