Fachhochschule Köln
Cologne University of Applied Sciences

Efficient Surrogate Assisted Optimization for Constrained Black-Box Problems

Samineh Bagheri

Master Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Engineering

Cologne University of Applied Sciences
Campus Gummersbach
Faculty of Computer Science
and Engineering

In the Course of Studies

Master of Automation & IT

First Supervisor:	Prof. Dr. Wolfgang Konen Cologne University of Applied Sciences
Second Supervisor:	Prof. Dr. Thomas Bäck Leiden University

February 2015

Abstract

Modern real-world optimization problems are often high dimensional and subject to many constraints. These problems are typically expensive in terms of cost and computational time. In order to optimize such problems, conventional constraint-based solvers require a high number of function evaluations which are not affordable in practice. Employment of fast surrogate models to approximate objective and constraint functions is a known approach for efficient optimization. The performance of the RBF interpolation is not dependent on the dimensionality of the optimization tasks. This is why in the area of surrogate-assisted optimization a lot of attention is devoted to RBF modeling. As an example for such a solver, COBRA is a constrained based efficient optimizer that outperforms many other algorithms on a large number of benchmarks. COBRA-R is a variant of COBRA extended with several algorithms such as a different initialization method and a novel repair technique. In this thesis, after investigating the strengths and weaknesses of COBRA-R, we introduced several extensions to enhance the overall performance of COBRA-R. Our investigation showed that the RBF surrogates cannot provide a suitable model for steep functions. Therefore, problems with steep objective or constraint functions have to be modified in order to be optimized with the COBRA-R approach. Additionally, the performance of COBRA-R is highly sensitive to the correct selection of a parameter called DRC. Also, the surrogate models appeared to be wrong for the problems with highly varying input ranges. Moreover, it was observed that sometimes a bad initial design could cause an early stagnation. The extended COBRA-R called self-adaptive COBRA-R intends to overcome these mentioned obstacles by including three extra steps: 1. Rescaling the input space to $[0, 1]^d$ (if it is necessary). 2. Automatic parameter/function(s) adaptation according to the information gained from the initial population. 3. Random start mechanism to avoid occasional bad solutions due to a few unfortunate initial designs. We evaluate our approach by using 11 G-problems and a high dimensional automotive problem (MOPTA08) as benchmark. We also report negative results where SACOBRA-R still shows a bad behavior and gives indications for possible improvements.

Keywords: nonlinear optimization, constrained optimization, expensive function, surrogate models, Radial Basis Functions, efficient optimization.

Acknowledgements

I want to express my deepest gratitude to my supervisor, Prof. Dr. Wolfgang Konen who was abundantly helpful and offered priceless assistance, support and guidance throughout my study period. Special thanks to Dr. Patrick Koch, for his support in the last year.

I would like to thank Prof. Dr. Bäck for accepting to supervise my thesis. The last but not the least, I would like to thank Markus Thill and Robin Eccleston for their great help in revising and editing this thesis.

To my parents and my brother Milad.

Contents

Abstract . . . iii
Acknowledgements . . . iv
List of Tables . . . viii
List of Figures . . . ix

Chapter 1 Introduction **1**
1.1 Related Work . . . 4
1.2 Thesis Outline . . . 5

Chapter 2 Methods **6**
2.1 No Free Lunch Theorems for Optimization . . . 6
2.2 Unconstrained Optimization Techniques . . . 6
2.2.1 Point Based Strategy . . . 7
2.2.2 Population Based Strategies . . . 9
2.3 Constraint Handling Techniques . . . 14
2.3.1 Death Penalty . . . 15
2.3.2 Penalty Functions . . . 16
2.3.3 Stochastic Ranking . . . 16
2.3.4 Multiobjective Optimization . . . 17
2.3.5 Repair Algorithms . . . 17
2.4 Surrogate Assisted Optimization . . . 17
2.4.1 COBYLA (Linear Approximation) . . . 17
2.4.2 COBRA (RBF) . . . 21
2.4.3 COBRA-R (RBF) . . . 24

Chapter 3 Research Questions **29**

Chapter 4 Experimental Analysis **32**
4.1 G-problem Test Suite . . . 32
4.1.1 Challenges of Optimization on the G-problems . . . 35
4.1.2 Performance of COBRA-R with tuned Parameters . . . 38
4.1.3 Performance of Self-Adaptive COBRA-R . . . 49
4.1.4 Comparing the Performance of SACOBRA-R and COBRA-R . 68
4.1.5 Comparison with other Techniques . . . 69

4.2 MOPTA 2008 73
4.2.1 Performance of COBRA-R with different Internal Optimizers . 74
4.2.2 Repairing infeasible for MOPTA08 78

Chapter 5 Conclusion and Future work **84**
5.1 Conclusion 84
5.2 Future Work 88

Bibliography **89**

Chapter A Algorithms **94**

List of Tables

4.1 Characteristics of the G-functions 35
4.2 Summary of the challenges of G-problems and possible solutions in COBRA 39
4.3 COBRA-R optimization error on G-functions with tuned parameters. 40
4.4 COBRA-R optimization result on G-functions with tuned parameters. 43
4.5 COBRA-R parameter settings for G-problems 44
4.6 Comparing the worst results achieved by SACOBRA-R and manually tuned COBRA-R 55
4.7 Self-adaptive COBRA-R optimization error on G-functions. 60
4.8 Self-adaptive COBRA-R optimization result on G functions. 60
4.9 Comparison of the error obtained by COBRA-R & self-adaptive COBRA-R 68
4.10 Result comparison with other techniques 69
4.11 Distance Requirement Cycles 76

List of Figures

2.1 Conceptualization of point based optimization method. 7
2.2 Conceptualization of population based optimization method. 10
2.3 Two dimensional demonstration of reforming a simplex (Nelder-Mead) 14
2.4 Conceptualization of RBF interpolation in $1D$, with Gaussian $\phi(r)$ . . 19
2.5 Conceptualization of RBF interpolation in $1D$, with cubic $\phi(r)$ 19
2.6 Flowchart of the COBRA [34] algorithm. 22
2.7 Flowchart of the COBRA-R algorithm. 25
2.8 randomly produced Latin hypercube in 2 dimensions 26

4.1 G06 problem 37
4.2 G02-2d problem 38
4.3 COBRA-R optimization process with tuned parameters for G-problems(1) 41
4.4 COBRA-R optimization process with tuned parameters for G-problems(2) 42
4.5 COBRA-R optimization process with tuned parameters for G-problems(3) 43
4.6 Impact of the initial population size on the COBRA-R performance for G03 45
4.7 Impact of the internal optimizer selection on the COBRA-R performance for G08 47
4.8 Impact of DRC selection on COBRA-R performance for G02-2d . . . 48
4.9 Self-adaptive COBRA-R algorithm flowchart 50
4.10 Impact of the input space normalization on the approximation error of the objective function 51
4.11 Impact of the logarithmic transformation of the objective function for G03 with a large FR 57
4.12 Self-adaptive COBRA-R optimization process for G-problems(1) . . . 61
4.13 Self-adaptive COBRA-R optimization process for G-problems(2) . . . 62
4.14 Self adapted COBRA-R optimization process for G-problems(3) . . . 63

4.15 Self-adaptive COBRA-R optimization process for the G02 problem . . 65
4.16 Objective function approximation error for G01 to G11. 66
4.17 COBYLA optimization process for MOPTA08 74
4.18 COBRA-R(ISRES) optimization process for MOPTA08. 76
4.19 COBRA-R optimization for MOPTA08 with COBYLA, HJ and NM. 77
4.20 Different optimization processes for MOPTA08 79
4.21 Different optimization processes for MOPTA08 (zoomed) 80
4.22 Repairing performance on reducing the number of violations. 81
4.23 Approximation error and move length in the input space for MOPTA08. 82

5.1 Impact of random start algorithm on G01 87

Chapter 1

Introduction

Nowadays, optimization problems arise in many fields like automotive and semiconductor industry, process and control engineering and many other areas. Real-world optimization problems often have to be tackled by numerical approaches due to the absence of an algebraic model of the problem to be optimized. It is common that complex computer-based models are used to simulate expensive engineering problems. For instance, a car crash simulation eliminates the possible costs imposed due to the need of replacing the damaged parts after doing a real crash test [18]. As the computational power grows, the simulation software becomes more and more accurate and complex. Complex simulations are computationally expensive. As an example, it can take ca. 20 hours to simulate a car crash [18]. Expensive optimization problems are desired to be solved within a severely limited number of evaluations. Another demand to address real-world optimization problems, is due to the limitations imposed by the existing constraints, restricting the valid solutions to a smaller subset of the search space. Therefore, the development of efficient optimization techniques for constrained expensive black-box problems are of great interest for industrial applications.

In order to handle constraints many different strategies are proposed in the scientific literature. It is very common to make use of dynamic or static penalty functions combined with conventional unconstrained solvers to address constrained problems. Some other constraint handlers are working on completely different principles. E.g., repair methods try to generate a feasible solution by modifying the infeasible solutions. Furthermore, several techniques minimize every constraint by considering them as additional objectives to be optimized. The underlying methods consider a constrained-based problem as a multiobjective problem and approach it with multiobjective optimization methods. Evolutionary strategies which are widely studied and used for unconstrained problem domains, can also be used in combination with the mentioned constraint handling techniques. On the other hand, improved stochastic ranking evolutionary strategy proposed in [38] introduces the stochastic ranking to tackle constraints by using the helpful information coming from infeasible points.

Although a lot of research have been devoted to constraint handling approaches, many of the constrained-based solvers demand for a large number of function evaluations. Up to now only few techniques have been proposed regarding *efficient* constraint optimization which can achieve a significant reduction in the number of function evaluations. A possible solution in that regard is to utilize *surrogate assisted* approaches to evaluate models of the objective and/or constraint functions instead of real expensive functions.

As a promising example of surrogate assisted optimization techniques for constrained problems, we can name *COBRA* introduced by Regis [34] which is tested on a large number of benchmarks and outperforms many other algorithms. The COBRA optimization framework utilizes cubic radial basis functions (RBF) to model the objective and constraint functions and performs a constrained optimization procedure on the surrogate functions. An implementation of the COBRA algorithm was developed in `R` under the name of *COBRA-R* optimization framework, the performance of the COBRA-R framework was tested and reported in [20]. COBRA-R is different from COBRA in various aspects. For example, a new repair algorithm called *RI-2* [21] is embedded in COBRA-R as an extension.

The COBRA and COBRA-R optimization frameworks, both use the same benchmark to evaluate the performance of the proposed solvers. A subset of well-known constrained problems, the so called G-problems [26], were used as artificial test-problems. Additionally, a substitute for a real-world car weight minimization subject to several safety requirements, called MOPTA08, was addressed by the mentioned algorithms. MOPTA08 provides a suitable benchmark to evaluate how well an optimizer can cope with demands of engineering problems which are highly constrained in a high dimensional space (68 constraints, 124 dimensions). It is desired to solve MOPTA08 problem within ca. 2000 or less evaluations since it may take about one month to evaluate approx. $15 \cdot d$ design points by a car crash simulation. Investing more computational time is not affordable for such a problem [19].

The better results achieved by COBRA-R [20] for the G-problems in comparison with COBRA [34] was mainly due to the use of manual parameter tuning for COBRA-R. Both techniques require to modify some constraint or objective functions which are difficult for radial basis functions to model. Manual tuning or problem modification is done based on our previous knowledge about the problem, though in black-box optimization such knowledge about behavior of objective and constraint functions is not known.

In this thesis we analyze the COBRA-R algorithm and reveal the strengths and weaknesses. We try to investigate different types of challenges imposed by the G-problems to the COBRA-R framework. After broadening our knowledge about the potential challenges that COBRA-R is dealing with, we introduce new extensions to the current COBRA-R framework. The performance of COBRA-R is sensitive to the parameter selection. Also, the minimization quality is not reasonable when problems with steep objective and constraint functions are addressed. Therefore, we introduced a new extension which is supposed to adapt the parameters and modify the objective and constraint functions of problems automatically (when it is necessary) according to the information driven from the initialization phase. Apart from introducing a problem/parameter adaptation step, we introduced a technique – called random start algorithm – to escape from possible local optima and improve on the worst-case results occurred due to a bad initial design. Furthermore, we investigated whether COBRA-R with the introduced extensions can accomplish good results without the need of manual parameter tuning. The extended COBRA-R optimization framework is named self-adaptive COBRA-R or SACOBRA-R in this thesis. This study focuses on investigating on the performance of the SACOBRA-R framework and comparing the results achieved on minimization of a subset of G-problems and the MOPTA08 problem. The results are compared with several other techniques and the impact of adding problem/parameter adaptation and random start algorithms is discussed in detail.

In general, the overall performance of SACOBRA-R on G-problems is beneficial in comparison with COBRA [34], in terms of better convergence to the real optimum. Also, SACOBRA-R reduces the dependency of the final optimization results on the parameter selection and pre-modification of test problems. SACOBRA-R is capable of improving on the worst case results for a number of G-problems due to the usage of the new extension introduced as the random start algorithm.

In this thesis, we also investigated the performance of COBRA-R with a different parameter setting than what was used in [21] to approach MOPTA08 problem. An improvement is achieved due to selection of a smaller set of distance requirement cycle which is one of the important parameters for the COBRA algorithm and its variants. We discussed the reasons behind the last achieved improvements on MOPTA08 problem in this study.

1.1 Related Work

Up to now various approaches have been proposed to solve constrained optimization tasks efficiently. Surrogate-based optimization is often used to address expensive problems [32]. For example, Wang, Dong, and Aitchison [43] proposed a variant of a response surface method to approximate the objective function by quadratic modeling. Although they addressed constrained problems, they assumed that constraint functions are cheap to evaluate in order to keep the problems easy to tackle. Kramer and Schwefel [22] investigated three methods to handle constraints in Evolution Strategies (ES). Later, Poloczek and Kramer [29] analyzed the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) in combination with a surrogate model based on support vector machines which is used as classifier for feasibility. In their approach it is assumed that only constraint evaluations are expensive and because of that they just concentrated on reducing the constraint function calls by means of support vector classifiers. They obtained slight improvements on analytical test functions, but also report negative results on some functions. Arnold and Hansen [1] give reasons for this decreased performance, which is possibly due to the rotation of the CMA-ES mutation distribution. On the other hand, the contribution made by [29] in terms of reducing the required number of constraint function calls was not significant. Arnold and Hansen [1] recommend an alternative approach which yields better results. Recently, Basudhar et al. [4] coupled the efficient global optimization algorithm with a support vector machine as constraint classifier and a Kriging approach is applied to model the objective function. They report good performances of the method in an experimental study, but use different benchmarks and do not compare their approach with other constraint-handling methods. The test problems used in the mentioned works are mostly low-dimensional problems.

Several optimization approaches such as [35], [13] and [15] utilized radial basis functions as surrogates. Radial basis function approximation is beneficial to other techniques in terms of the small computational time required for modeling, even for high dimensional problems. In [33] a novel surrogate assisted optimization method is proposed for high dimensional expensive black box problems. Later, the same author extends his work and proposes a more sophisticated algorithm called COBRA [34] which also uses RBF models. COBRA [34] obtained promising results on a large range of benchmarks varying in many different aspects. Usage of radial basis functions as surrogate model made it possible for COBRA [34] to approach a very high dimensional problem like MOPTA08 with 124 dimensions and 68 constraints efficiently. Although the results shown for the G-problems approached by COBRA [34] are as good as those by evolutionary based strategies such as [37] and [6], for many of the G-problems a reasonable solution was found after a very limited number of evaluations. Later, [20] and [21] developed COBRA-R which is a variant of the COBRA algorithm in `R`, extended with a repair method and different initialization techniques. Although COBRA-R [20] appeared to be capable of solving the G-problems and MOPTA08 better than COBRA [34], the enhancement was achieved by manual tuning of the parameters. In this study, we review the main differences between COBRA [34] and COBRA-R [20]. Additionally, we introduce two algorithms as extensions to COBRA-R to eliminate the need of manual tuning.

1.2 Thesis Outline

In this thesis several important optimization methods are discussed and explained in Chapter 2. In the same chapter, a few unconstrained techniques which are widely used as the basis of the constrained solvers are discussed (Section 2.2). In Section 2.3, various constraint handling techniques are explained. Section 2.4 addresses surrogate assisted approaches. First, Powell's COBYLA [31] is described and then the Regis' COBRA [34] is discussed in detail followed by an explanation of the COBRA-R optimization framework and its differences from COBRA.

We pose several research questions in Chapter 3. In order to answer the research questions COBRA-R optimization results on G-problems and an engineering problem MOPTA08 are presented and analyzed in Chapter 4. In Section 4.1 SACOBRA-R is introduced and the results obtained for G-problems by means of SACOBRA-R are analyzed and compared with the results from several other techniques. In Section 4.2 the performance of our optimization framework is evaluated on MOPTA08. Finally, in Chapter 5 the outcome of this thesis is summarized and further steps for investigation is listed as the future work.

Chapter 2

Methods

2.1 No Free Lunch Theorems for Optimization

Wolpert and Macready in 1997 [44] introduced the free lunch theorem for optimization. Basically, this theory is telling us that it is impossible to develop a universal optimization technique which can outperform the other optimization approaches on all possible classes of problems [11]. Wolpert and Macready [44] prove that the averaged performance of all algorithms are exactly same if they are performed on all possible types of problems.

Michalewicz also points the similar idea with other words in [26]. No universal optimization algorithm exists for addressing all problems. As Gregory mentions in 1995 in order to successfully optimize a problem we "should try to select an approach that fits the problem we are solving".

Overall, gaining some insight about different optimization algorithms and various possible classes of problems is more important than just comparing the performance of several strategies for a small group of problems. Roughly speaking, it is worthy to illustrate where and why an algorithm fails or succeeds.

2.2 Unconstrained Optimization Techniques

Many optimization approaches applied to constrained problems use unconstrained techniques in combination with a suitable constraint handling method. For instance, handling constraints by means of penalty functions transforms a constrained problem into an unconstrained problem. Therefore, a conventional unconstrained approach in combination with a constraint handler can be utilized to locate a global feasible optimum or several local optima. Unconstrained problems are less challenging and studied more often than constrained problems. There are a number of proposed techniques to solve such problems in a satisfactory manner although there is no single method which can cope with all types of problems. Later, we describe COBRA-R framework which is capable of applying any unconstrained optimization technique in order to address constrained problems. However, a promising result is highly dependent on relevant selection of techniques and parameters. Studying various kinds of unconstrained optimization algorithms gives us an insight into the correct choice of algorithms for different problems. In general, iterative unconstrained optimization can be categorized into two separated classes: point based and population based strategies.

An unconstrained optimization problem is defined as the minimization of an objective function f in a search space bounded by a lower bound l_b and an upper bound u_b. Maximization and minimization problems are sharing similar principles. A maximization problem can be converted to a minimization problem by negating the objective function f. An unconstrained optimization problem is formulated as follows:

$$\text{Minimize} \quad f(\vec{x}), \qquad \vec{x} \in \mathbb{R}^d,\ l_b \leq \vec{x} \leq u_b. \tag{2.1}$$

2.2.1 Point Based Strategy

Point based strategies address optimization algorithms which use one point in every iteration to produce only one new iterate. These type of algorithms need one function evaluation per iteration Figure 2.1. The new iterate will be considered as starting point in the next iteration if it produced a better objective value. Otherwise, the old starting point should be modified in a somehow different manner as long as the termination criteria are not met. Two well-known algorithms in this class are: simulated annealing for global optimization and Hooke-Jeeves for local optimization.

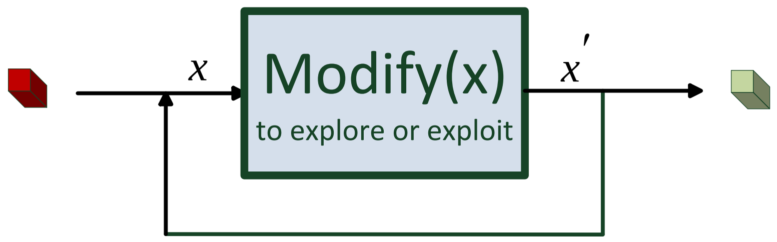


Figure 2.1: Conceptualization of point based optimization method.

Pattern Search (Hooke-Jeeves)

Hooke-Jeeves [14] is a classical pattern search algorithm which is very well suited for numerical black-box unconstrained problems because of its derivative-free manner. The Hooke-Jeeves process is a local direct search; it cannot guarantee to locate the optimum of a multimodal problem unless enough sets of starting points distributed all over the search space are used. In every iteration the next iterate is selected by a local sequential *exploratory move* or *exploitative move*. *Exploitative moves* which are also called pattern moves follow a successful exploratory move. The algorithm can be summarized as follows:

Exploratory move. In the beginning of every iteration the best visited point so far ($\vec{x}^{best}$) is considered as the starting point. In order to obtain knowledge about the correct move direction, exploratory moves are performed. Consider, that the problem $f(x)$ is defined in d dimensions and that we want to explore behavior of $f(x)$ in the neighborhood of the current best point. An exploratory move aims to find $\vec{x}^{new}$ in the neighborhood of $\vec{x}^{best}$ with a better objective function value. In every iteration we walk from $\vec{x}^{best}$ along every dimension with the step size ϵ. First we perform an incremental move and if no progress could be observed then we move to the opposite direction. As soon as a better point is obtained the vector $\vec{x}^{new}$ is updated. If exploratory moves have been considered in all the directions and resulted in a reduction of the objective function value $f(\vec{x}^{new}) < f(\vec{x}^{best})$ then the pattern move is called. Exploration moves can fail if the search is stuck in a local optima. On the other hand, in the case of unimodal problems as we approach the real optimum we require smaller step sizes. Therefore, if the exploration fails then the step size ϵ should be reduced.

Pattern move. Only if the exploration in one or more directions led to a better iterate, a pattern move or exploitative move is carried out. Pattern moves use the information gained by exploration to speed up the optimization process by assuming that the direction of the last move $\vec{x}^{new} - \vec{x}^{best}$ is the correct direction. Thus, it moves the new point found by exploration $\vec{x}^{new}$ slightly into the same direction, although there is no guarantee that the pattern move yields in a progress. Hence, we assign the pattern move's result in a temporary variable $\vec{x}^{temp}$:

$$\vec{x}^{temp} = \vec{x}^{new} + \alpha(\vec{x}^{new} - \vec{x}^{best}), \tag{2.2}$$

where, α is the acceleration factor.

After executing the pattern move it is time to update the $\vec{x}^{best}$. If the pattern move was successful then $\vec{x}^{best}$ will be replaced by $\vec{x}^{temp}$ otherwise, the current best point is the output of the exploratory move before the exploitative move $\vec{x}^{new}$. The process will be repeated as long as the termination criteria is not met. The termination criteria can be defined in various ways. In efficient optimization we have limited budget of function evaluations. Therefore, one of the termination criteria could be the number of function evaluations.

2.2.2 Population Based Strategies

Population based strategies are iterative optimization processes which evolve solutions in every iteration by an interaction of a population of solutions Figure 2.2. Many of such heuristics are inspired by nature, which are either on the basis of Darwin's law of evolution or a behavioral imitation of social animals. In principle, they can be classified into categories of Evolutionary Algorithms (EAs) and Swarm Intelligence (SI). Such approaches appeared to perform strong in black box optimization and attracted the interest of many researchers. Numerous variants of nature based methods have been developed during the last years. The performance of one algorithm in comparison to the other is dependent on the test problems and parameter settings and there is no algorithm which can outperform all others in black box optimization. On the other hand, the progress of nature-inspired heuristics in the field of efficient constrained black box optimization is not yet empirically observable. As population the based approach usually requires a large number of function evaluations to find the optimum, it is not the optimal approach for problems with a limited budget [28]. Although there are many attempts in the direction of parameter adjustment in order to reduce the number of function evaluations imposed by such techniques, still the improvement is not satisfactory. The challenge becomes even more pronounced for constrained problems because many constraint handling tricks impose even more evaluations of true functions. Therefore, using surrogate models can be an efficient solution for use of this class of population based optimization problems, which will be discussed in this study. Evolutionary Strategies [39] and Genetic Algorithms [10] are two very well-known branches of Evolutionary Algorithms.

Evolutionary Strategies

There are many variants of evolutionary strategies. At this point we provide an explanation of a basic multimembered ES. This variant is typically a better imitation of natural evolutionary processes in comparison with a two membered ES or (1+1)-ES.

The main concept of evolutionary strategies is the survival of the fittest members of a population, therefore, a higher chance for good genes to appear in the future generations. Assume, that the number of parents in every generation is μ and the

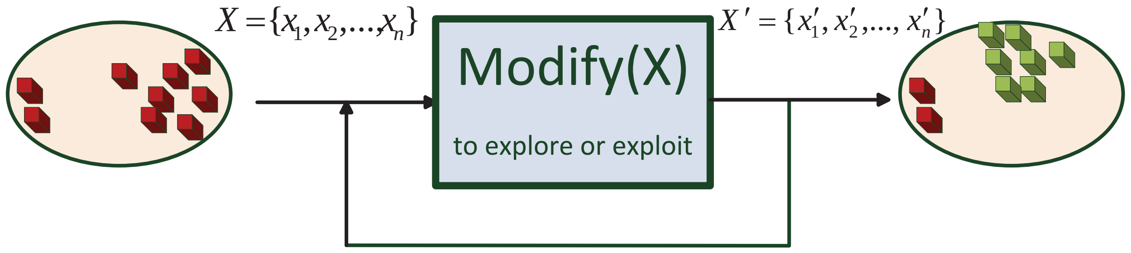


Figure 2.2: Conceptualization of population based optimization method.

number of offspring in the same generation is λ. Then, in a two membered evolutionary strategy both of these parameters have a value of one. Thus the parent of the next generation is selected from a population of the size $\mu + \lambda = 1 + 1$ according to the fitness value of current parent and offspring. Nevertheless, this model of evolutionary strategy is far from the natural evolution. Consider, $\lambda > 1$ and $\mu > 1$, then the selection can be done either among the parents and the offspring $(\mu + \lambda)$ or just among the offspring λ (in this case the offspring population should be larger than the parents population $\lambda > \mu$). In the first multimembered approach $(\mu + \lambda)ES$, a good parent can survive for very long time over generations. This behavior is inconsistent with with reality. Any member of a population has a limited lifetime. Therefore, the second mentioned approach $(\mu, \lambda)ES$ appears to be more realistic.

In practice, every ES includes three main steps after the generation of the initial population:

Step 0: **(Initialization)**
In the beginning, a parent population should be initialized with μ individuals. Initialization can be a tricky task for constrained problems but for unconstrained problems we can often use random initialization in the search space, or biased initialization in the neighborhood of a promising starting point which is known before any computation starts.

Step 1: **(Mutation)**
Parents are responsible for reproducing a population of offspring with the size of λ by mutation. This is done by modifying the parents genes and creating new individuals. Genes are variables of the problem, so an n dimensional problem has n genes. Consider, $\vec{x}_k^{(g+1)} = \{x_{k,1}^{(g+1)}, x_{k,2}^{(g+1)}, \ldots, x_{k,n}^{(g+1)}\}$ is the k-th parent

in the $g+1$ generation with n genes. Therefore, in this scenario mutating the individuals means moving the parents slightly in the input space. On the other hand, it should be considered that every parent should generate λ/μ offspring in average. Thus, the offspring population of $\vec{y}_l^{(g+1)}$ for $l \in \{1, \ldots, \lambda\}$ is produced as follows:

$$\vec{y}_l^{(g+1)} = \vec{x}_k^{(g+1)} + \vec{z}^{(g\lambda+l)}, \tag{2.3}$$

where $\vec{y}_l^{(g+1)}$ is the vector of l-th offspring, $l \in \{1, \ldots, \lambda\}$ and $k \in \{1, \ldots, \mu\}$.

e.g., $k = \begin{cases} \mu & \text{if } l = pl, \, p \text{ is an integer} \\ l \ mod \ \mu & \text{otherwise.} \end{cases}$

$\vec{z}^{(g\lambda+l)}$ is a normally distributed random vector $N(0, \sigma_i^2)$. Hence, in every generation a random vector z is generated λ times to move each parent about λ/μ times with an averaged step size of σ in the input space in order to generate the offspring.

Step 2: **(Ranking & Selection)**
After the mutation step, all the newly generated individuals will be evaluated by an objective function and they will be ranked with respect to their fitness value. Consider, the optimization problem as described in Equation 2.1. Therefore, the individuals which have a smaller objective value are considered as the fitter and as a result they are ranked higher. A sorting algorithm should be used to put the offspring individuals in the desired order. After sorting the offspring we have

$$f(\vec{y}_1^{(g+1)}) < f(\vec{y}_2^{(g+1)}) < \ldots < f(\vec{y}_\lambda^{(g+1)}). \tag{2.4}$$

The very first μ individuals among the offspring population will be selected to be the parents in the next generation. Therefore,

$$\vec{x}_s^{(g+2)} = \vec{y}_s^{(g+1)}, \text{ for } s \in \{1, \ldots, \mu\}. \tag{2.5}$$

Step 3: **(Check Termination Criteria)**
Termination criteria are often checked after the evaluation and ranking step, in the case of fulfilling any termination criterion the process will be stopped. Otherwise, the process will be continued from step 1.

Selection and control of the step size σ_i has influence on the performance of the optimization. According to Darwin's evolutionary model, mutation is not fully random. Therefore, this idea that the result of reproduction of every parent can be in a hyperellipsoid with semi-axes σ_i for $i \in \{1, \ldots, n\}$ around the parent, is clever. But there is no unique step size vector which is valid over all generations. So, this vector can be controlled based on the success rate. Basically, the step size should be increased in the case of a high success rate and should be decreased if the success rate is lower than desired.

One other approach to control the step size, is considering σ_i as a strategy parameter and initialize it with the initiative population for each single individual. Then, in every generation recombine and mutate individuals and strategy parameters [2].

Genetic Algorithms

Genetic algorithm is similar to ES in many aspects. Both techniques are biological-inspired population based strategies which can address nonlinear black-box unconstrained problems by means of recombination and mutation of the superior individuals in the population [12].

Although, GA and ES are sharing many conceptual properties, they are pretty different in the details regarding the operators. One of the main differences between GAs and EAs is about the representation of the individuals which is considered as binary coding for GA whereas, ES uses real values representation. In [12] a more detailed list of differences between these two well-known evoluitonary algorithms is provided.

Nelder-Mead Optimization Algorithm

We already discussed a pattern search technique. Nelder-Mead represents a local direct-search optimizer. This unconstrained optimization method was introduced in [27] for the first time but it is used as the basis of several other sophisticated algorithms which can handle constraints in an efficient manner [24]. Therefore, we bring an abstracted explanation of it here.

Classical Nelder-Mead can be applied to solve high dimensional unconstrained problems. In this approach, for an n-dimensional problem in every iteration a simplex with $n + 1$ vertices is utilized. Every vertex is ranked regarding its objective value and the worst vertex will be replaced by a better point in the search space. The simplex approaches the optimum by adaptively reshaping over iterations.

In order to replace the worst vertex, we first need to find a better point. There are several means of building a new simplex which are used in the specific situations during the search. A new simplex can be created by one of the four schemes of *reflection*, *expansion*, *contraction* and *shrink*.

Assume, $\{\vec{P}_1, \vec{P}_2, \ldots, \vec{P}_{n+1}\}$ are $n+1$ points in an n-dimensional space, and they represent the vertices of the current simplex. The point with the highest objective value for every simplex is then considered as the worst point of the iteration $\vec{P}_w$ and the point with the lowest objective value is the best point $\vec{P}_b$. We define a hyperplane $\mathbf{V}$ which is $\sum \vec{P}_i$ for $i \neq w$. Centroid of the V is represented as $\bar{P}$.

Reflection
Consider a linear approximation in the trust region specified by the current simplex. By walking from the worst point $\vec{P}_w$ to any point in the $\mathbf{V}$ hyperplane, the objective value decreases. Therefore, the first guess for finding a better point could be the reflection of the worst point over the centroid point $\bar{P}$. Reflection is done in the beginning of every search cycle and can be awarded by expansion in the case of success or it can be followed by contraction if it fails. A moderate solution by reflection will simply be replaced by the worst solution and next search cycle starts again with reflection.

- Success : $f(\vec{P}_r) < f(\vec{P}_b)$
- Fail : $f(\vec{P}_w) < f(\vec{P}_r)$
- Moderate : $f(\vec{P}_b) < f(\vec{P}_r) < f(\vec{P}_w)$

Expansion The expansion in Nelder-Mead algorithm is performed to speed up the search. If in the reflection phase the move was towards the correct direction, then we move slightly further into the same direction.

Contraction A failure for the reflection can be due to a large move or a non valid direction addressed by the linear approximation. In this case, we select two points on the hyperline which connects the worst point $\vec{P}_w$ to the centroid point $\bar{P}$ equally distanced from the centroid.

$$\begin{aligned} P_{c_o} &= \bar{P} + \gamma(\bar{P} - \vec{P}_w) \\ P_{c_i} &= \bar{P} - \gamma(\bar{P} - \vec{P}_w) \text{ where } \gamma < 1 \end{aligned} \tag{2.6}$$

Shrink The simplex must be shrunk towards the best point $\vec{P}_b$, if contraction fails. Shrinking is done by by replacing all $\vec{P}_i$ by $\frac{\vec{P}_i + \vec{P}_b}{2}$.

These schemes are visualized for two dimensional in the Figure 2.3. Moreover, the general algorithm for any arbitrary n is listed in Algorithm 3.

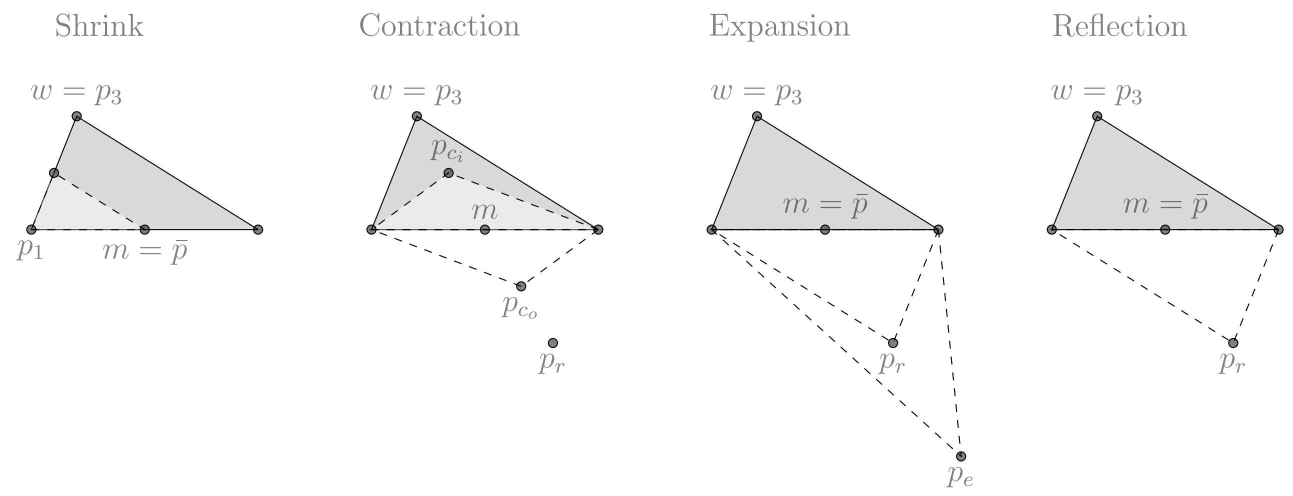


Figure 2.3: Two dimensional demonstration of four means of building a new simplex in the Nelder-Mead optimization algorithm: reflection, expansion, contraction and shrink.

2.3 Constraint Handling Techniques

Engineering optimization problems in real-world are more complicated than what is described by Equation 2.1 which basically represents the minimization of an objective function. Most of the underlying problems belong to the class of constrained optimization problems (COPs) [16]. Valid solutions for COPs are located in a so-called feasible region restricted by several constraints and this makes such class of problems very demanding.

Generally, the constrained minimization problem can be formulated as follows: the task is to find a feasible point $\vec{x} = \{x_1, x_2, \ldots, x_d\} \in \mathtt{R}^d$ which minimizes the objective function function $f(x)$. A solution is considered as feasible when it satisfies all equality and inequality constraints, h_j and g_i, respectively.

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

The existence of equality constraints $h_j,\ j = 1, 2, \ldots, m$ makes the COPs even more demanding. Many algorithms cannot handle problems with equality constraints directly and they change the equality constraint to the inequality constraint. Therefore, we reformulate the COP as follows:

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

Conventional unconstrained techniques are not sufficient to address COPs. Typically, a combination of unconstrained optimization algorithms with a constraint handling extension are applied to COPs [23]. Constraint handling is done in different fashions. Several approaches concentrate on the feasible region and assume that any feasible solution is better than the infeasible ones [23] [9] unlike some others which try to approach the feasible region by allowing some infeasible points in the population [37]. Additionally, there exists several approaches which benefit from the existence of infeasible solutions by repairing and guiding them to the feasible area [6] [25] [21]. Various constraint handling techniques can be classified as following:

- *Death penalty* rejects the infeasible individuals and re-sample as long a feasible solution is found. This method is used with simulated annealing [40] and ES for simple problems [3].
- *Penalty functions* are added to the objective function to increase the fitness value proportional with the distance to the feasibility or the number of violated constraints.
- *Stochastic ranking* is utilized for ES. With a fixed probability P_f feasible solutions are ranked according their objective value. Otherwise, high ranks are assigned to the infeasible solutions according to the sum of the constraint values [37].
- *Repair algorithms* attempt to find a solution by modifying the infeasible solutions.
- *Multiobjective optimization*, where the objective and constraint functions are both minimized [16].

2.3.1 Death Penalty

Death penalty is a simple algorithm but not very practical especially for problems with very small feasible region. It imposes many extra evaluations to locate a feasible solution. By rejecting all the infeasible points we are losing the useful information about the distance to the feasible area.

2.3.2 Penalty Functions

Since the penalty principle is simple and easy to implement, many constrained optimization algorithms are working based on the usage of the different variants of penalty functions [23]. Another advantage of the penalty functions is that they can be used in combination with most of the unconstrained optimization algorithms and transform COPs to conventional unconstrained problems. The main idea is to add a penalty value proportional to the values of the violated constraints to the objective function (see Equation 2.7).

$$\tilde{f} = f(x) + \alpha \cdot G(x). \tag{2.7}$$

Where, α is the penalty factor and can be assigned statically or dynamically.

Selecting a suitable penalty factor requires extra tuning and a correct penalty factor is not the same for various problems. Although several self-adaptive penalty approaches [8] were proposed, there exists no unique algorithm for determining the correct penalty factor during the optimization procedure.

In [37], the two terms of *underpenalization* and *overpenalization* are introduced to describe the circumstances of having a wrong penalty factor. If the penalty term is too small (underpenalization) then basically the constraints are not taken into account and the minimization is done only regarding the objective function. Therefore, the optimization procedure can have a problem locating any feasible solutions. Furthermore, a large choice for the penalty factor results in ignoring the impact of the objective function. This yields in finding feasible solutions but not finding a correct direction to minimize the objective value.

2.3.3 Stochastic Ranking

In ES for each generation the new parents are selected by ranking the offspring. For unconstrained problems the ranking is done regarding the objective value. For COPs, the ranking can be done by comparing the penalized objective function $\tilde{f}$ Equation 2.7. The problem with this approach is the challenge to find a proper way to select correct penalty factor. In order to avoid underpenalization and overpenalization stochastic ranking was introduced [37]. The main idea is to take advantage of both feasible and infeasible points by allowing several infeasible points in the parents populations. It is shown in [37] that this method is successful in optimizing the G-problems.

2.3.4 Multiobjective Optimization

[40], [7] and [16] introduce techniques in which constraints are handled by being considered as another objective function. Multiobjective optimization can be sometimes more expensive and complicated than the conventional COPs [42]. It is not efficient to use multiobjective optimization especially when the number of constraint functions are relatively high.

2.3.5 Repair Algorithms

The constraint handling approaches which attempt to generate feasible points by modifying an infeasible point are called repair algorithms. Except from gene-repair which is a nature-inspired strategy and can be applied to genetic algorithms, there are also general repair approaches which can be combined with various optimization techniques. [21] and [6] introduced a gradient based repair algorithm. Both methods have similar principles. They both use the gradient information driven from constraints. The differences of these two approaches are discussed in detail in [21].

2.4 Surrogate Assisted Optimization

2.4.1 COBYLA (Linear Approximation)

COBYLA [31] was introduced in 1994 by Powell. This technique is a more sophisticated and efficient version of Nelder-Mead described in Section 2.2.2 and also it is designed to solve constrained problems. This method utilizes linear approximation to model the objective and constraint functions. In every iteration a new simplex is formed and the linear models are fitted through the vertices of the simplex. The procedure to select a new vertex is similar to Nelder-Mead explained in 2.2.2, with the difference that COBYLA selects a better point (vertex) according to an approximated merit function $\hat{\Phi}$ (see Equation 2.8). The constraints are handled by use of a penalty function which is always the greatest violated constraint according to the linear approximation of the constraint functions $\hat{c}_i$ multiplied by a penalty factor α.

$$\hat{\Phi} = \hat{f}(x) + \alpha \cdot [\max\{\hat{c}_i : i = 1, 2, \cdots, m\}]_+ \,, \; x \in \mathtt{R}^d, \tag{2.8}$$

where, $\hat{}$ is representing the linear approximation instead of the true function. Therefore, $\hat{f}(x)$ is the linear approximation of the objective function, $\hat{c}_i$ is the linear approximation of the i-th constraint function and $\hat{\Phi}$ is the function which is supposed to be minimized in every iteration of COBYLA.

COBYLA uses a dynamic penalty function. The penalty factor α is initially set to a small positive value and is adjusted internally. Let us assume that x^* is the optimum of the problem and in every iteration the best point found so far is x^b. Therefore, it is not logical if the value of the merit function in the optimum is larger than the value of the merit function in the best found point, and if this happens it provides an evident that the penalty factor is not large enough. Hence, consider $\bar{\alpha}$ is the smallest possible value which fulfills the following condition $\hat{\Phi}(x^*) < \hat{\Phi}(x^b)$, then the penalty factor α is left as it is if $\alpha > \frac{3}{2} \cdot \bar{\alpha}$; otherwise, α is increased to $2 \cdot \bar{\alpha}$. According to [31] the adjustment factors are determined by numerical calculations. Although, COBYLA appears to be successful for minimizing several well-known COPs, there is no guarantee that the described dynamic penalty approach is sufficient for all the problems.

Radial Basis Function

In surrogate assisted optimization, finding a suitable approximation technique to build the surrogate model is a challenge. Although, there are multitude approaches with various characteristics, there are not many which can compete with radial basis function interpolation when dealing with high dimensional problems [5]. It is worth to mention that RBF interpolation is also suitable for problems in low dimensions. In the other words, performance of RBF interpolation is not highly dependent on the dimensionality of the problem comparing with other multivariate approximation methods.

Any function which is only dependent on the distance from a specific point in the space belongs to the group of radial basis function.

$$\phi(x) = \phi(r) = \phi(||x - c||), \ x, c \in \mathbb{R}^d \tag{2.9}$$

The distance r is often determined based on the Euclidean norm but this is not the only approach. In addition, for defining ϕ a wide range of choices exist like Gaussian, thin plate spline and polyharmonic spline.

Gaussian RBF: $\phi(r) = e^{-\epsilon r^2}$, where ϵ can have any positive value.

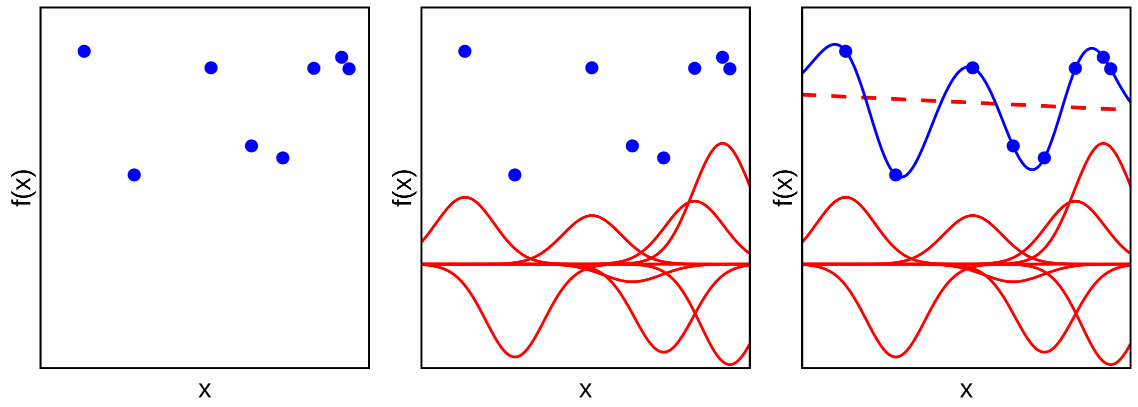


Figure 2.4: Conceptualization of RBF interpolation in $1D$, with Gaussian $\phi(r)$. The goal is to approximate a curve according to the information from the blue points. The red curves are weighted Gaussian radial basis functions with centers of blue points. The red dashed line is the polynomial $p(x)$. Summation of all the red curves and the dashed line is the blue curve which interpolates all points and fits a smooth curve through them.

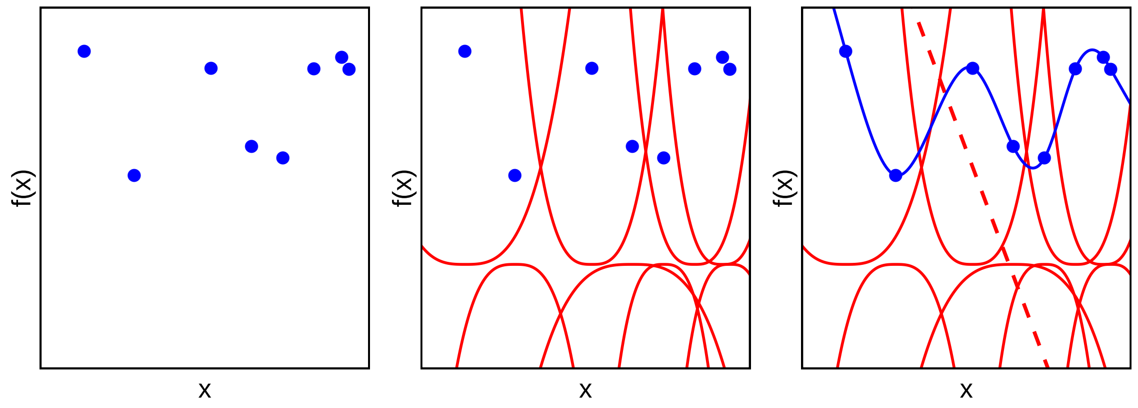


Figure 2.5: Conceptualization of RBF interpolation in $1D$, with cubic $\phi(r)$. The goal is to approximate a curve according to the information from the blue points. The red curves are weighted cubic radial basis functions with centers of blue points. The red dashed line is the polynomial $p(x)$. Summation of all the red curves and the dashed line is the blue curve which interpolates all points and fits a smooth curve through them.

$$\text{polyharmonic spline: } \phi(r) = \begin{cases} r^k & \text{where } k = 1, 3, 5, \dots \\ r^k ln(r) & \text{where } k = 2, 4, 6, \dots \end{cases}$$

$$\text{thin plate spline: } \phi(r) = r^2 log(r)$$

pseudo-cubic: $\phi(r) = r^3$

RBF interpolation fits a model on n points $x = \{x_1, x_2, \dots, x_n\}$ in d dimensional space with the known real function values of $U = \{u(x_1), u(x_2), \dots, u(x_n)\}$. This is done by summation of n radial basis functions defined with x_i centers and with different weights λ_i.

$$s_n = \sum_{i=1}^{n} \lambda_i \phi(||x - x_i||) + p(x), \tag{2.10}$$

where $p(x)$ is a polynomial tail added to the summation of RBFs in order to reduce degrees of freedom. Hence, weights of the radial basis functions and the polynomial can always be obtained uniquely by solving the following system of equations.

$$\begin{pmatrix} \Phi & P \\ P^T & 0_{(d+1)\times(d+1)} \end{pmatrix} \begin{pmatrix} \lambda \\ e \end{pmatrix} = \begin{pmatrix} U \\ 0_{d+1} \end{pmatrix}, \tag{2.11}$$

where Φ is the distance matrix – a symmetric matrix with zero diagonal – and $\Phi_{ij} = \phi(||x_i - xj||)$, i and j both are from 1 to n. In addition, $0_{(d+1)\times(d+1)}$ and 0_{d+1} are zero matrices in $\mathbb{R}^{(d+1)\times(d+1)}$ and $\mathbb{R}^{d+1}$ space, resp. Moreover, P is defined as follows,

$$P = \begin{bmatrix} 1 & x_1^T \\ 1 & \vdots \\ 1 & x_n^T \end{bmatrix}_{n\times(d+1)}.$$

We obtain the weights of RBFs $\lambda = \{\lambda_1, \dots, \lambda_n\}^T$ and coefficients of the polynomial tail $e = \{e_1, \dots, e_d\}^T$ as following,

$$\begin{pmatrix} \lambda \\ e \end{pmatrix} = \begin{pmatrix} \Phi & P \\ P^T & 0_{(d+1)\times(d+1)} \end{pmatrix}^{-1} \begin{pmatrix} U \\ 0_{d+1} \end{pmatrix} \tag{2.12}$$

It is important to stress this point that the matrix inversion can be done if and only if $rank(P) = d+1$ [5], [30] which means we need $d+1$ affinely independent points among n points. Therefore, we can say for RBF interpolation in the d dimensional space at least $d+1$ points must be used.

RBF models are easy to implement and computationally efficient even for high dimensional problems. Also, they can train models with reasonable accuracy and only a very few points. Regis in [34] uses a cubic RBF for COBRA and Extended ConstrLMSRBF algorithms. The same author in [33] and [35], illustrates the superiority of the cubic RBF in comparison with other kinds of RBFs like thin plate spline for black box optimization applications. As a result, we also utilize the same form of the RBF for COBRA-R optimization framework. But it is very easy to modify this feature in COBRA-R framework and use other types of RBF or even other kind of surrogates.

Figure 2.4 is a $1D$ interpolation example by means of Gaussian radial basis function. The goal is to approximate a curve according to the information from the blue points. Red curves are weighted Gaussian radial basis functions with centers of blue points. The red dashed line is the polynomial $p(x)$. Summation of all the red curves and the dashed line is the blue curve which interpolates all points and fits a smooth curve through them.

2.4.2 COBRA (RBF)

Constrained Optimization by Radial Basis Function Approximation (COBRA) is a surrogate assisted optimization algorithm proposed by Regis [34]. 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. COBRA uses RBF interpolation to model the objective and constraints. Each iterate is a result of an optimization on a subproblem, which is a constrained problem defined by the RBF interpolation models of the objective and the constraint functions.

Figure 2.6 illustrates a flowchart of the algorithm. In the fist step an initial population is produced to make it possible to create the first RBF model. The initialization phase in COBRA [34] is done by locating $d + 1$ random individuals in the search space. The resulting RBF models from the initial design are used to find the next iterate to be evaluated on the real function. The COBRA process has two main phases. If there exists no feasible point in the initialization step phase I is performed and as soon as a feasible point is found then the phase II is called; otherwise, phase II is directly started. During the both phases an internal optimizer minimizes the subproblem defined in each phase. Basically, the subproblems are optimization problems on the surrogate.

Note that the solution returned by the optimization on the surrogates (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 internal optimizer returns a better point 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. This procedure will be performed in a sequence until the number of function evaluations exceeds the maximum number of allowed evaluations given by the user (the budget for the optimization).

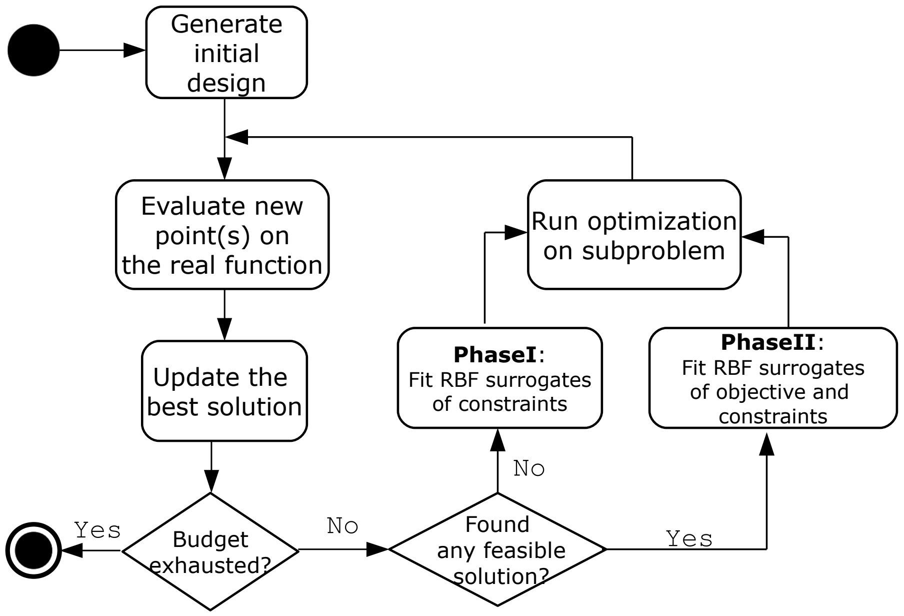

Generate initial design
Evaluate new point(s) on the real function
Update the best solution
Budget exhausted?
Yes
No
Found any feasible solution?
No
Yes
PhaseI: Fit RBF surrogates of constraints
PhaseII: Fit RBF surrogates of objective and constraints
Run optimization on subproblem

Figure 2.6: Flowchart of the COBRA [34] algorithm.

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 neighborhood of the actual best solution. The distance requirement can be passed by the user as an external parameter vector $\Theta = \langle \theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(\kappa)} \rangle$ in phase I and $\Xi = \langle \xi^{(1)}, \xi^{(2)}, \ldots, \xi^{(\kappa)} \rangle$ in phase II, where $\xi^{(i)},\ \theta^{(i)} \in \mathbb{R}^{\geq 0}$. 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 smallest side of the bounds box. In each iteration n we only claim the point to be feasible if the following Equation holds for all constraint surrogates $s_i^{(n)}$ with $i = 1, \ldots, m$:

$$s_i^{(n)} + \epsilon_i^{(n)} \leq 0 \tag{2.13}$$

The constraints are tightened 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 adjust ϵ by dividing or doubling it 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.

Subproblem optimization

In each iteration COBRA performs an optimization on the RBF models. This is done by the Fmincon function in `MATLAB`. This function utilizes an interior point algorithm and can address constrained problems.

In phase I, the goal is to find a feasible point. Therefore, the subproblem in phase I only gives attention to the minimization of the summation of the violated constraints according to the surrogate models of the constraint functions. However, in phase II the objective RBF model is supposed to be minimized subject to the surrogates of constraint functions. In this work the solution returned by the internal optimization on the surrogates is called infill point. The subproblem in phase I is formulated as follows:

$$\begin{aligned}
&\text{Minimize} && \textstyle\sum_{i=1}^{m}[\max\{s_n^{(i)}(\vec{x}), 0\}]^2, && \vec{x} \in \mathbb{R}^d,\ a \leq \vec{x} \leq b && (2.14)\\
&\text{subject to} && s_n^{(i)}(\vec{x}) + \epsilon_n^{(i)} \leq 0, && i = 1, 2, \ldots, m && (2.15)\\
& && \rho_n - ||x - x_j|| \leq 0, && j = 1, \ldots, n && (2.16)
\end{aligned}$$

where $s_n^{(i)}$ is the surrogate model for the i-th constraint function. Also, $\epsilon_n^{(i)}$ is the margin used to take the uncertainty of the constraint prediction into account. Furthermore, ρ_n is the required distance of the new iterate from all the other points in the population in the n-th iteration.

The subproblem in phase II has a different objective in comparison with the subproblem in phase I. The subproblem during phase II is described as following:

$$\begin{aligned}
\text{Minimize} \quad & s_n^{(0)}(\vec{x}), && \vec{x} \in \mathbb{R}^d,\ a \leq \vec{x} \leq b && (2.17)\\
\text{subject to} \quad & s_n^{(i)}(\vec{x}) + \epsilon_n^{(i)} \leq 0, && i = 1, 2, \ldots, m && (2.18)\\
& \rho_n - ||x - x_j|| \leq 0, && j = 1, \ldots, n && (2.19)
\end{aligned}$$

where $s_n^{(0)}(\vec{x})$ is the surrogate model for the objective function.

2.4.3 COBRA-R (RBF)

The promising results achieved by COBRA [34] for high dimensional highly constrained optimization problem (MOPTA08) motivated us to re-implement the COBRA algorithm. COBRA-R [20] is an implementation of COBRA [34] in `R`.

Although, COBRA and COBRA-R are sharing many common principles, there are several differences which can lead to different results for the same problems. The main fundamental differences between COBRA [34] and COBRA-R [20] are listed as follows:

- *Implementation environment*: COBRA-R is implemented in `R` while COBRA is implemented in `MATLAB`
- *Initialization*
- *Skipping phase I*: COBRA-R has an option to skip the phase I and directly proceed with phase II even if no feasible solution is found in the initialization phase.
- *Internal optimizer*
- *Repair infeasible*: an extension to COBRA embedded in COBRA-R

Initialization

The COBRA-R optimization framework gives the users the possibility of choosing between several initialization approaches like LHS, Biased and Optimized. Except

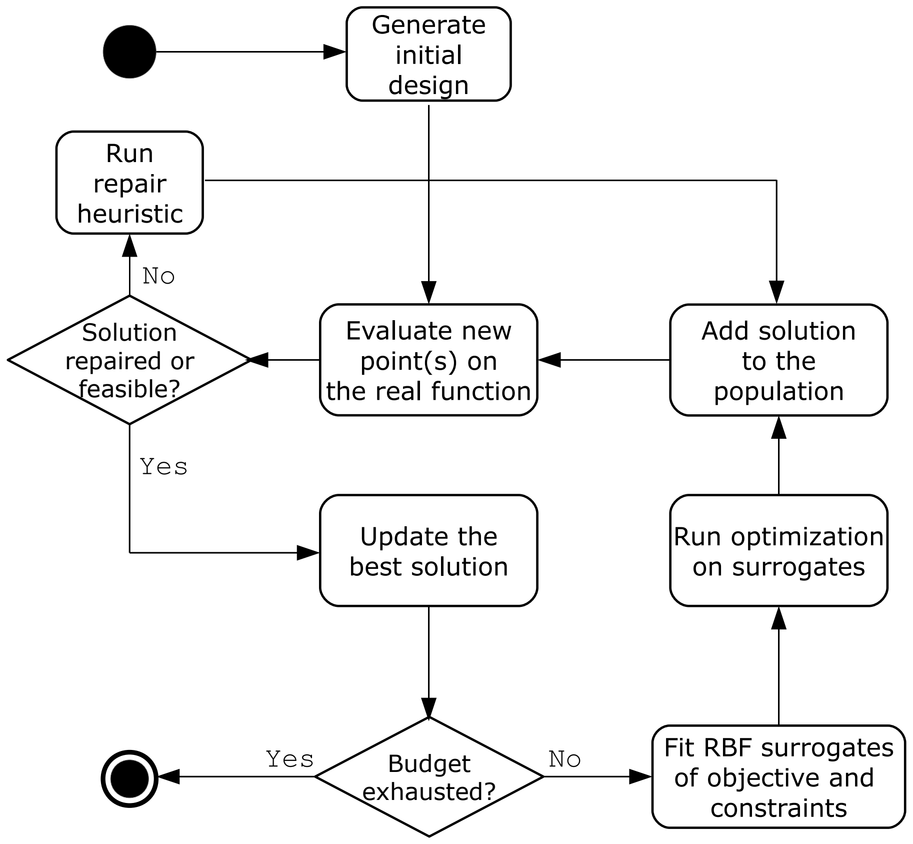

Generate initial design
Run repair heuristic
No
Solution repaired or feasible?
Evaluate new point(s) on the real function
Add solution to the population
Yes
Update the best solution
Run optimization on surrogates
Yes
Budget exhausted?
No
Fit RBF surrogates of objective and constraints

Figure 2.7: Flowchart of the COBRA-R algorithm.

from the LHS initialization the other algorithms are only practical if a feasible starting point is provided. In COBRA [34] the initialization is usually done randomly by means of Latin hypercube sampling.

LHS or Latin hypercube sampling is one of the options in the COBRA-R framework to create the initial population. The underlying algorithm generates random points in a way that the search space is well covered.

Assume, we require to generate n individuals in a two-dimensional input space. Every variable (x_1, x_2) should be stratified into n quasi-equal intervals. The formed square grid is called a Latin square if and only if in each row and column only one point exists (see Figure 2.8). In fact, the Latin hypercube design concept is on

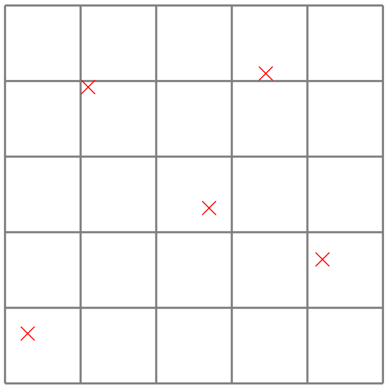


Figure 2.8: A randomly produced Latin hypercube for two variables and 5 individuals (samples).

the basis of the generalization of the two-dimensional Latin square to any arbitrary dimensions [41].

The "lhs" package in `R` provides an implementation of Latin hypercube sampling. The COBRA-R optimization framework utilizes this package to generate the random initial population.

Biased initialization is usually recommended to be performed if a feasible starting point is provided for the problem with a small feasible region. The main idea is to sample points close to the known starting point instead of sampling within the whole search space. Although this approach can be beneficial in some cases, if the starting point is located in a region far from the real optimum then the model built on these points after the initialization phase is not very informative.

In practice, the Biased initialization in the COBRA-R framework generates random points normally distributed around the suggested starting point $N(\vec{x}_{start}, \sigma^2)$, the standard deviation σ can be passed by the user through the parameter called "*initBias*". The *initBias* should be kept in a small range. If the user does not set this parameter, then 0.005 is assigned by default.

Optimized initialization is also recommended to be used for unimodal problems with a feasible starting point. In this approach, unlike the two former ones the starting population is not generated randomly.

As it is expected from the name of the Optimized initialization approach, the initial set of points are created by performing an optimization technique on the real functions. We use Hooke & Jeeves search for a limited number of evaluations to provide enough individuals required in the initialization step. Moreover, the constraints are handled by penalty function addressing the maximum violated constraint.

Internal optimizer

Different optimizers are used by the COBRA-R and COBRA frameworks to solve the subproblem defined by the surrogate models. We call the optimizers used in the inner loop as the internal optimizer. The internal optimizer used in COBRA [34] is Fmincon function provided by the optimization tool in `MATLAB`. It is mentioned in [34] that this function uses an interior point algorithm. The COBRA-R optimization framework gives the users the possibility to choose between COBYLA, ISRES, HJKB and NMKB. All the mentioned solvers are already implemented in `R`. The latter optimization techniques are described in Section 2.2, 2.3 and 2.4.1. In general, any unconstrained or constrained technique can be embedded to the COBRA-R optimization framework.

We use two constrained (COBYLA [31], ISRES [38]) and two unconstrained (NMKB, HJKB) solvers for the internal optimizer. NMKB and HJKB are the implementation of Nelder-Mead and Hooke&Jeeves techniques in `R`, respectively. As it is mentioned before, the subproblem in phase I and phase II both are COP. Therefore, an unconstrained solver requires a trick to handle the constraints.

When an unconstrained technique such as HJKB and NMKB is utilized as the internal optimizer in the COBRA-R framework, the constraints are handled by means of a static penalty approach. In the COBRA-R we have the possibility to skip the phase I and directly start with the phase II, here we discuss how we handle the constraints of the subproblem in phase II. For subproblem of phase II we have two types of the constraints to handle: 1. the constraint imposed by the required distance from the other points in the population. 2: surrogate models of the constraint functions. Usually the COPs have more than one constraint function. In order to handle the second type of the constraints, we always take the summation of the violated constraints on the surrogate into account. Therefore, the function passed to the unconstrained internal optimizers is as following:

$$\text{Minimize} \quad s_n^{(0)}(\vec{x}) + \alpha_2 \cdot (\alpha_1 \cdot penalty1 + penalty2), \tag{2.20}$$

$$\vec{x} \in \mathbb{R}^d, \ a \leq \vec{x} \leq b, \tag{2.21}$$

where, *penalty*1 and *penalty*2 are defined as following:

$$penalty1 = \sum_{i=1}^{m} \max\{s_n^{(i)}(\vec{x}) + \epsilon_n^{(i)}, 0\}, \qquad i = 1, 2, \ldots, m \tag{2.22}$$

$$penalty2 = \sum_{j=1}^{n} \max\{\rho_n - ||x - x_j||, 0\}, \qquad j = 1, \ldots, n. \tag{2.23}$$

The penalty factors α_1 and α_2 are constant during the internal optimization on the surrogates. But, they get adjusted in every iteration, if it is necessary. In every iteration if the infill point does not fulfill the constraint related to the distance requirement cycle, then α_1 is increased. Also, α_2 is increased in every iteration up to a maximum value which can be assigned by the users. The penalty functions described in Equation 2.23 are simply used for the unconstrained internal optimizers NMKB and HJKB. For the constrained internal optimizers (ISRES and COBYLA), the constraints of the subproblem defined in Equation 2.16 and 2.19 are directly passed to the solver.

Repair algorithm

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 the infill points with a slight infeasibility by guiding them to the feasible region. The repair algorithm used in COBRA-R is called ”RI-2”. This algorithm is described and discussed in detail in [21]. It is worthwhile to mention that, since the repair algorithm is performed on the surrogate models no real function evaluations is used and only after the repair algorithm returns a point the real function is called for the purpose of evaluation of the new point.

Chapter 3

Research Questions

In this work we mainly investigate the performance of the COBRA-R optimization framework, which is a variant of COBRA introduced by Regis in [34] for solving black-box constrained optimization problems. First, we present results accomplished by COBRA-R with the manually tuned parameter settings differing from problem to problem. These results are partly presented in an earlier study in [20]. Additionally, we study the benefits of extending the COBRA-R framework with an automatic parameter adaption mechanism. Moreover, we introduce an approach to select the starting point of the internal optimizer to investigate whether an occasional random start has the potential to improve the results in minimization. In this Chapter we state several research questions. In order to answer the research questions, the optimization algorithms are evaluated with 12 benchmarks including MOPTA08 and the well-known G-problems suite.

Q.1: Which challenges are imposed to COBRA-R for optimizing the G-problems?

G-problems with different dimensions, different types of fitness and constraint functions impose various kinds of challenges to the COBRA-R optimization algorithm. This question is addressed in detail in Section 4.1 and answered in the same section.

Q.2: Can COBRA-R efficiently optimize the G-functions?

To our best knowledge, several techniques already exist which can solve G-problems and approach the optimum of many G-problems with a great accuracy, but most of the algorithms require a large number of function evaluations. One important question is whether the COBRA-R optimization algorithm is capable of reducing the amount of function evaluations required for minimizing G-problems?

Q.3: Can COBRA-R achieve better results than COBRA [34] in solving G-problems? If so, what are the reasons for improvement?

Regis in [34] reports the minimization results for G-problems achieved by COBRA algorithm initialized with one set of parameter settings for all problems. Although the results are obtained after a very limited number of function evaluations, the final results for many of tested problems are not as accurate as those determined by evolutionary algorithms with many function evaluations. COBRA-R is an implementation of COBRA in `R` with several differences described in Section 2.4.3. We show that the performance of COBRA-R is very parameter sensitive. Therefore, it is interesting to investigate if COBRA-R initialized with manually tuned parameters for each problem shows any benefits in terms of finding a better solution for G-problems in comparison with COBRA.

Q.4: Are there strategies to improve upon the worst-case results in minimization? E.g. to avoid occasional bad solutions due to an unfortunate initial design?

In order to evaluate the COBRA-R framework all tests for G-problems are repeated 30 times. There are some problems which can be solved efficiently with COBRA-R but few trials out of 30 never improve due to an unfortunate initial design. We introduced a random start strategy to prevent a seldom early stagnation. The performance of the introduced technique is tested in this study.

Q.5: Is it possible to achieve reasonably good results for all G-problems, with COBRA-R initialized with only one set of parameters?

COBRA and COBRA-R need to modify some problems before starting the optimization process in order to assure RBF interpolation can provide an appropriate model for the objective or constraint functions. On the other hand, the performance of COBRA is sensitive to the selection of several parameters like the DRC parameter, initialization approach and size of initial population. We investigate whether extending COBRA-R with a parameter adaptation algorithm provides a framework that can solve G-problems needless of manual tuning and function modification.

Q.6: Does COBRA-R show any benefits on high dimensional problems such as MOPTA08 in comparison with COBRA [34]?

In addition to G-problems we study the performance of our optimization framework on MOPTA08 test problem which is a substitute for a real world problem with high dimension(124) and high number of constraints(68). We compare our results with the promising results presented in [34] and try to answer the question above.

Chapter 4

Experimental Analysis

In this chapter, we investigate the performance of the COBRA-R optimization framework by evaluating it on the G-problem test suite and MOPTA 2008 benchmarks. We compare our results with the original COBRA algorithm [34] and some other optimization techniques.

After describing the G-problem test suit in Section 4.1, different challenges imposed by these problems to the COBRA-R framework are discussed in Section 4.1.1. Analyzing COBRA-R's behavior in approaching different G-problems gave us indications to find suitable parameter configurations for each problem. In Section 4.1.2, we present the optimization results for the G-problems, using the manually tuned COBRA-R optimization. We introduced several algorithms to enhance the overall performance of COBRA-R. We call the extended COBRA-R algorithm as Self-Adaptive COBRA-R (SACOBRA-R) and we discuss the proposed extensions in Section 4.1.3. We evaluate the SACOBRA-R on the G-problems and bring a comparison with COBRA-R and several other techniques, in Section 4.1.4 and 4.1.5, respectively.

Section 4.2 describes a high dimensional real-world optimization problem (MOPTA08). First, we show how COBRA-R with different internal optimizers can approach the MOPTA08 problem in Section 4.2.1. In Section 4.2.2, we show successful results accomplished for optimizing MOPTA08 by means of COBRA-R algorithm with assistance of the RI-2 repair technique. We also report how a careful selection of DRC parameter can improve the performance of the COBRA-R optimization on MOPTA08.

4.1 G-problem Test Suite

The G-problem test suite [26] is very often studied in the scientific literature for analyzing constrained-based solvers. Therefore, the G-functions provide a good analytical testbed for our optimization framework.

The underlying functions differ in dimension ($d = 2, \dots, 20$), type of objective function (linear, quadratic, nonlinear), and number and type of constraints ($m = 1, \dots, 9$). The large range of variation in the characteristics of the G-problems makes them even more interesting to be studied in order to test the strength of the optimization algorithms in facing different challenges.

In practice, except from the important mentioned features of a toy problem like dimension, type of fitness and constraint functions, there are many other features which can make a real difference in the difficulty of the optimization problems and as a result in finding a suitable technique to solve them.

Optimization problems have different bound constraints $\vec{x} \in [\vec{a}, \vec{b}]$, which restrict the input space in every dimension by the lower bounds $\vec{a}$ and the upper bounds $\vec{b}$. We defined the input space elongation ISE as a measure to indicate the ratio of the maximum input range and the minimum input range:

$$ISE = \frac{\max(\vec{b} - \vec{a})}{\min(\vec{b} - \vec{a})}. \tag{4.1}$$

In addition, we indicate the range of the objective values over the search space with a feature called FR. This feature can show the steepness of the fitness function. It is difficult to model steep functions with RBF interpolation. So, FR can be considered as a measure for the complexity level of a problem to be approached by an RBF surrogate assisted strategy.

$$FR = \max(f(\vec{x})) - \min(f(\vec{x})), \quad \vec{x} \in [\vec{a}, \vec{b}], \tag{4.2}$$

where, $f(\vec{x})$ is the value of the objective in point $\vec{x}$. We estimate the objective range FR by locating n points in the search space and calculating the fitness value in these points. Let us assume that, $F = \{f_1, f_2, \dots, f_n\}$ is a set of the objective values determined for n points distributed in the search space. Therefore, an estimation for FR can be determined as follows:

$$FR \approx \max(F) - \min(F). \tag{4.3}$$

Most of the G-problems have multiple constraints and the values of these constraint functions can be varied in different ranges. The constraint ratio feature GR is a measure to determine the ratio of the largest constraint value range to the smallest constraint value range. The range of every constraint function gr can be determined similar to how the objective range is determined:

$$gr = \max(g(\vec{x})) - \min(g(\vec{x})), \quad \vec{x} \in [\vec{a}, \vec{b}], \tag{4.4}$$

where, $g(\vec{x})$ is representing the value of the constraint in point $\vec{x}$. Let us consider that, for a problem with m constraint functions $G = \{gr_1, \ldots, gr_m\}$ is the set of the constraint ranges. Therefore, the constraint ratio feature GR is defined as follows:

$$GR = \frac{max(G)}{min(G)}. \tag{4.5}$$

In order to determine an estimation for this measure we place n points in the search space and measure all m constraint values for all n points distributed in the search space.

$$GR \approx \frac{\max_k \left(\max_i(g_i^{(k)}) - \min_i(g_i^{(k)})\right)}{\min_k \left(\max_i(g_i^{(k)}) - \min_i(g_i^{(k)})\right)}, \tag{4.6}$$

where, $i = 1, \ldots, n$ and $k = 1, \ldots, m$.

The feasibility ratio ρ is the fraction of the feasible space within the search space. This value is almost zero for problems with equality constraints. We evaluate the feasibility of n random points in the search space to estimate the feasibility ratio feature ρ by counting the number of feasible points and dividing it by the number of all n evaluated points.

$$\rho = \frac{\text{feasible space}}{\text{search space}} \approx \frac{\text{number of feasible points}}{\text{number of all evaluated points}}. \tag{4.7}$$

The problems with a low feasibility ratio ρ are usually more challenging than other problems with a wide feasible area.

Since the COBRA-R framework can only deal with inequality constraints, we modified problems with equality constraints like G03, G05 and G11 and transformed their equality constraint to inequality. Depending on the problem we decide whether the positive or negative side of the equality constraint should be considered as the infeasible region of the corresponding inequality. ρ^* used in Table 4.1 is the feasibility ratio of the modified problems.

The values shown in the Table 4.1 regarding ρ^*, FR and GR are obtained by placing 10 million random points in the search space and measuring the corresponding fitness and constraint values in order to determine the defined measures.

Table 4.1: Characteristics of the G-functions: d: dimension, ρ^*: feasibility rate(%) after changing equality constraints to inequality constraints, FR: range of the fitness values, GR: ratio of the constraints values, LI: number of linear inequalities, NI: number of nonlinear inequalities, NE: number of nonlinear equalities, a: number of active constraints in the optimum.

Fct.	d	type	ISE	ρ^*	FR	GR	LI	NI	NE	a
G01	13	quadratic	100	0.0003%	298.14	1.969	9	0	0	6
G02	20	nonlinear	1	99.997%	0.57	2.632	1	1	0	1
G03	20	nonlinear	1	0.0000%	92684985979.23	1.000	0	0	1	1
G03MOD	20	nonlinear	1	0.0000%	24.4	1.000	0	0	1	1
G04	5	quadratic	1.33	26.9217%	9832.45	2.161	0	6	0	2
G05	4	nonlinear	1090,91	0.0919%	8863.69	1788.74	2	0	3	3
G06	2	nonlinear	1.15	0.0072%	1246828.23	1.010	0	2	0	2
G07	10	quadratic	1	0.0000%	5928.19	12.671	3	5	0	6
G08	2	nonlinear	1	0.8751%	1821.61	2.393	0	2	0	0
G09	7	nonlinear	1	0.5207%	10013016.18	25.05	0	4	0	2
G10	8	linear	10	0.0008%	27610.89	3842702	3	3	0	3
G11	8	linear	1	66.7240%	4.99	1.000	3	3	0	1

4.1.1 Challenges of Optimization on the G-problems

As the initial optimization tests in COBRA-R, we started with the suggested parameter setting by Regis [34]. He uses $d+1$ random points to generate the initial population. In fact, COBRA creates the initial population by use of the minimum permitted points because in a d-dimensional space at least $d+1$ points are necessary in order to fit a RBF model. The distance requirement cycle in phase I is fixed to $\Theta = \{0.1, 0.05, 0.01, 0.005, 0.001, 0.0005\}$ and in phase II is selected either as $\Xi_{LOCAL} = \{0.01, 0.001, 0.0005\}$ or $\Xi_{GLOBAL} = \{0.1, 0.05, 0.01, 0.005, 0.001, 0.0005\}$ which has more elements. This leads to a more diverse search. Regis reports the results for these two choices of Ξ as COBRA-LOCAL and COBRA-GLOBAL, respectively. In addition, feasible and infeasible thresholds (τ_{feas}, τ_{infeas}) are set to $\left\lceil 2\sqrt{d} \right\rceil$. Moreover, $0.005l([a,b])$ is assigned to the initial and maximum margins for both algorithms. These parameters are defined in Section 2.4.2.

In the first step, we run our experiments with the same parameter settings. However, COBRA [34] and COBRA-R[20] are different in several aspects. The differences are described in Section 2.4.3. For instance, COBRA utilizes the fmincon function from the `MATLAB` optimization toolbox but COBRA-R uses several other solvers as the internal optimizer.

The very first results gave us the impression that the quality of the final solution found for the G-problems by COBRA-R is highly dependent on the parameter setting. Although the given experimental setup by Regis [34] gives intermediate results on a large set of test problems, it is not the optimal setting for all of them.

In order to find a better setting it can be helpful to investigate the possible obstacles COBRA-R has in handling optimization problems from different classes. We classified the possible challenges of COBRA-R solving the G-problem test suite as following:

- **Many local optima:** First, these type of problems are challenging to be modeled by means of RBF interpolation and limited amount of points. Second, even if the model is reasonably fitted, a local internal optimizer has difficulties to approach the global optimum, especially when the number of evaluations are restricted. As an illustration, we can name G02 and G08 which are problems with many local optima and a global optimizer technique should be used as the internal optimizer, otherwise the results determined by COBRA-R are very poor.

- **High FR:** The high variation of fitness values over the search space causes trouble for our RBF modeling approach to train a model with acceptable accuracy. On the other hand, a large FR is an indication for a steep fitness function. This means, that a small move in the input space can cause a large change in the objective. Therefore, the optimum of such a problem should be approached with smaller steps which means smaller values in the set of the distance requirement cycle are more desirable. COBRA-R has to deal this type of challenge to optimize the G03, G06, G09 and G10 problems. But only a careful selection of the DRC is not sufficient for problems similar to G03 with extremely a large fitness range ($FR > 10^{11}$). Another solution can be a logarithmic transformation of the fitness function which is also suggested in [34] and the modified version of G03 is, called G03MOD.

- **High GR:** When GR is high it means different constraints have very different ranges in the overall search space. This can cause severe problems for elimination of the infeasibility because most of our options as internal optimizers handle constraints with the penalty approach. Hence, this high variation causes that always some constraints are ignored. On the other hand, steep constraint functions are difficult to be modeled with RBF. G10 is one of the problems which suffers from this issue. Regis suggests a logarithmic transformation approach for the steep constraint functions of G10. This problem has

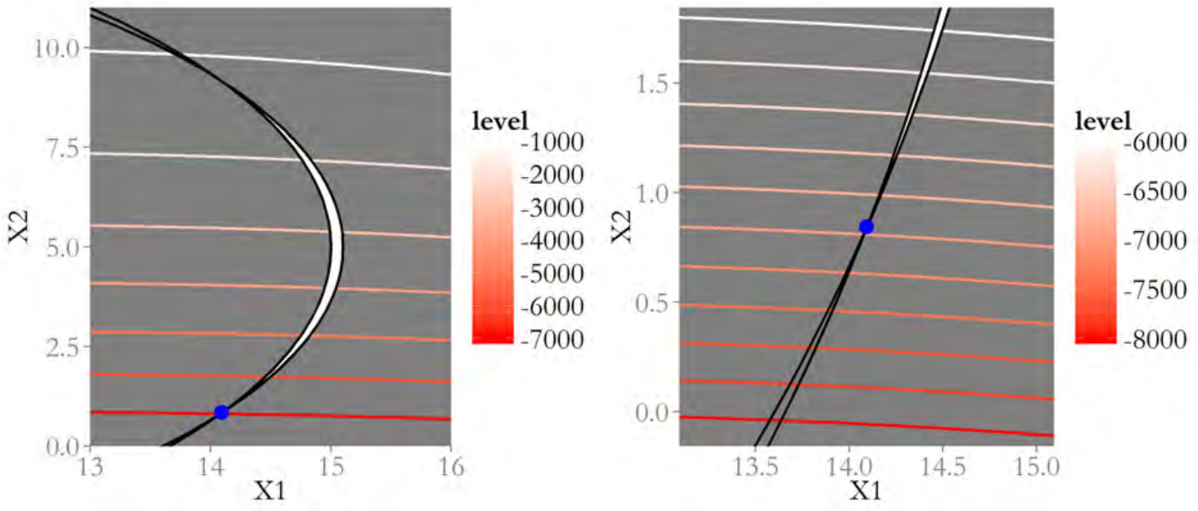


Figure 4.1: G06 problem. The black lines are representing constraint boundaries. The gray area describes the infeasible region and the white area describes the feasible zone. The curves which vary in color from red to white are showing the objective function contour lines, darker colors have lower objective values. The blue point represents the optimum. The original search space for this problem is $[13, 100] \times [0, 100]$. Left: overview of infeasible region for G06. Right: zoomed in to the interesting region.

8 constraints and 3 of them have large and fast variations. Regis applies a logarithmic transformation for those three constraints and names the modified test problem G10MOD in [34].

- **Zero or small feasible region (ρ^*):** COBRA-R has difficulties to find a feasible solution for problems with very small or zero ρ when the distance requirement cycle includes large values. The presence of large elements in the set of the distance requirement cycle always leads to undesired jumping in the search space and this can only be beneficial if the problem is highly multimodal. In some cases, adding an absolute zero value to the DRC set is required to find the optimum with a high accuracy. Otherwise, if a good point is found in the neighborhood of the optimum then large values of DRC, mostly guide the next iterates to the infeasible region. It can happen that the true optimum is placed in the forbidden region restricted by DRC and it can never be found. G01, G03, G07, G08, G10 have very small ρ^*. Although G06 has a large ρ^* in comparison to the other mentioned G-functions, it is still challenging and this is because of its needle shape in the interesting region. In this case, in the

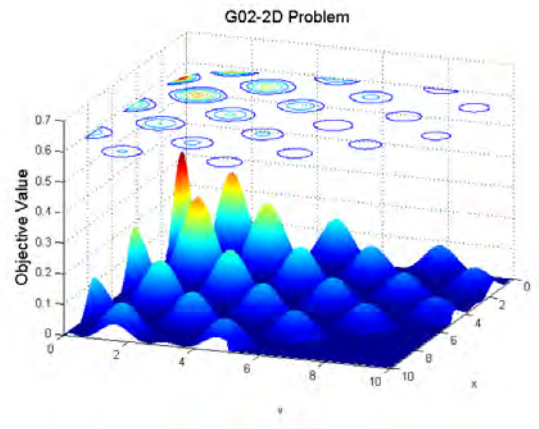


Figure 4.2: G02 problem.

neighborhood of the true optimum the feasibility ratio is much lower than the real feasibility ratio ρ^* (see Figure 4.1).

- **High input space elongation (ISE):** Problems which have some variables changing in a large range and some other variables having a relatively small min-max range can face difficulties to be correctly modeled by RBF interpolation. Also, it is very tough to select an appropriate DRC for problems with such an input space. For example, if the search is started from the wrong side of the space a small step size causes a very slow convergence and in the opposite scenario a large step may skip the interesting region. G05 is one example with this issue. Our experiments show that we cannot accomplish good solutions without normalizing the input space for such problems.

Summary of the challenges of the G-problems and their possible solutions in COBRA-R are listed in the Table 4.2.

4.1.2 Performance of COBRA-R with tuned Parameters

Since the G-problems differ in a large range of features and impose different types of challenges to the COBRA-R optimization framework, we tuned the initial parameters of COBRA-R for every single problem in a different manner to obtain successful optimization results. The results presented in this section are also partly shown in [20].

Table 4.2: Summary of the challenges of G-problems and their possible solutions in COBRA-R

	Challenge(s)	**Solution(s)**
G01	Small feasible region. Getting stuck in a local optimum with some initial population.	Add 0.3 to DRC (more exploration), from time to time start the internal search from a random point.
G02	Multimodal: Many local optima, especially in 20d	None!
G03	Nonlinear and non-separable objective. High dimension $d = 20$ and large `FR`.	Logarithmic transform of the objective function
G04	Fitness function and constraints with mixed terms x_1x_2	Easy with COBYLA (others fail: NMKB, ISRES)
G05	Extremely thin feasible region. Three nonlinear active constraints. Highly varying input ranges.	Rescale inputs to $[0, 1]^d$.
G06	Very thin feasible region, optimum at tip of needle. Steep objective function, large range `FR`.	Add 0.0 to DRC (avoid blocking the optimum)
G07	Constraints with mixed terms $x_1x_2 \rightarrow$ difficult for constraint surrogates. Relatively large `FR`. Very small $\rho = 0.0001\%$.	Use optimizer COBYLA (others fail: NMKB, ISRES)
G08	Shallow optimum in feasible region is masked by high (+/−) infeasible peaks, multimodal problem	Use optimizer ISRES (others fail: NMKB, COBYLA)
G09	Very large range `FR` $\rightarrow$ one wrong point can spoil the surrogate models	Start directly in the "good" region, use a local surrogate model
G10	Very small $\rho = 0.0007\% \rightarrow$ difficult to find a feasible solution. Large range GR.	Normalization of constraints.
G11	Equality constraint (Not a tough challenge).	Transforming equality constraint to inequality (We only pass problems with equality constrained to COBRA-R framework).

Table 4.3: Performance of the COBRA-R optimization algorithm on the G-functions with tuned parameters for each problem. The statistics on the optimization error $f(x) - f(x^*)$ are determined based on 30 independent runs. The statistics on the final optimization results $f(x)$ for the same runs are listed in Table 4.4.

	Best	Median	Mean	Worst	sd	fe
G01	2.9E-07	7.5E-05	6.2E-01	2.5E+00	1.1E+00	100
G03MOD	6.7E-07	6.4E-06	2.5E-05	3.2E-04	6.4E-05	400
G04	2.5E-10	6.3E-08	6.7E-07	1.1E-05	2.1E-06	200
G05	1.3E-05	3.0E-04	1.0E-03	2.5E-02	4.5E-03	200
G06	1.9E-04	2.2E-03	5.8E-03	8.1E-02	1.5E-02	100
G07	4.0E-08	6.3E-07	2.7E-06	4.1E-05	7.6E-06	200
G08	3.4E-09	6.6E-07	1.5E-06	2.0E-05	3.5E-06	200
G09	1.5E-08	3.5E-07	1.8E+00	3.8E+01	7.1E+00	300
G10MOD	5.1E-03	8.7E-02	1.6E-01	1.3E+00	2.6E-01	300
G11	3.7E-15	2.3E-13	3.4E-13	1.6E-12	3.9E-13	100

After finding the best parameter setting for each problem (Table 4.5), every test is repeated 30 times with a limited evaluation budget which also varies from problem to problem. The statistics of the final deviation from the optimum value and the final objective value are listed in Table 4.3 and Table 4.4, respectively. In addition, the applied parameter setting for each G-problem is presented in Table 4.5.

The COBRA-R optimization result for G02-20d is not listed in this table because we believe we cannot find any suitable parameter setting in COBRA-R to approach such problem with many local optima due to the modeling limitations (see Figure 4.2).

As it is listed in Table 4.4, COBRA-R with tuned parameters can approach the optimum value of all G-problems except G02-20d with a reasonable accuracy after a limited number of function evaluations.

We also displayed the progress of the manually tuned COBRA-R optimization process after the initialization phase for every G-problem in Figures 4.3 to 4.5. In these figures the result presented by Regis in [34] are also indicated by a red square. Regis [34] always allows only 100 function evaluations. If there is no red square in the figure this means there was no result presented for the corresponding problem. For instance, there is no result reported in [34] for the problems G01 and G11. Also, the missing red point in the plot of the G03 problem in Figure 4.3 is because we

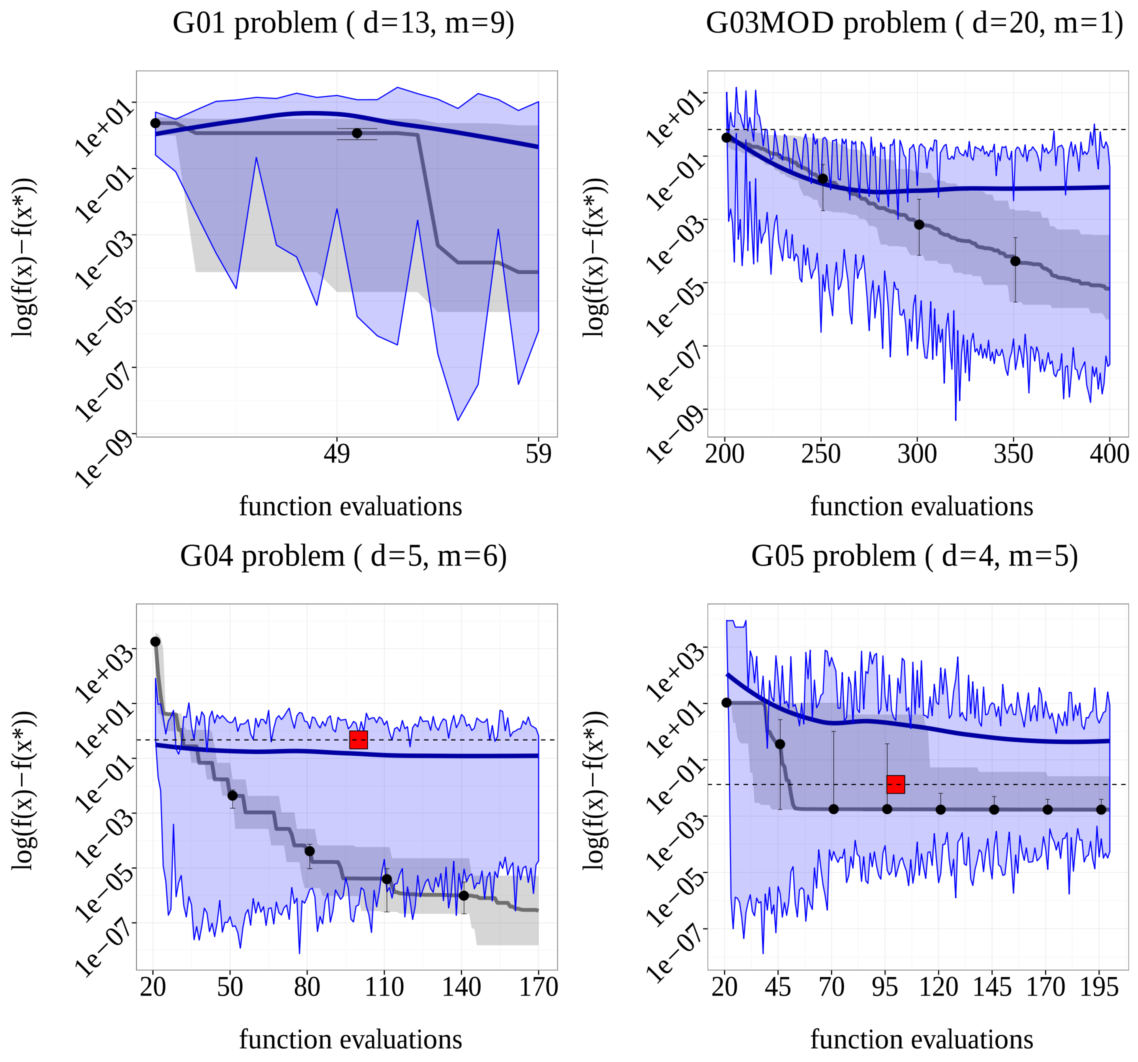


Figure 4.3: Manually tuned COBRA-R optimization process for G01, G03, G04 and G05. The gray curve is representing the median error per iteration for 30 independent trials. The gray shade around the median is showing the worst and the best error. The red square is the result reported in [34] after 100 iterations. The blue curve is a fitted smoother for the averaged RBF approximation error, shown in logarithmic scale $log(|S_0^{(k)}(\vec{x}_k) - f(\vec{x}_k)|)$, where $S_0^{(k)}$ is the RBF model of the fitness function in the k-th iteration trained with $k-1$ iterates excluding x_k. The blue ribbon around the averaged approximation error is representing the minimum and maximum approximation error among 30 runs in every iteration.

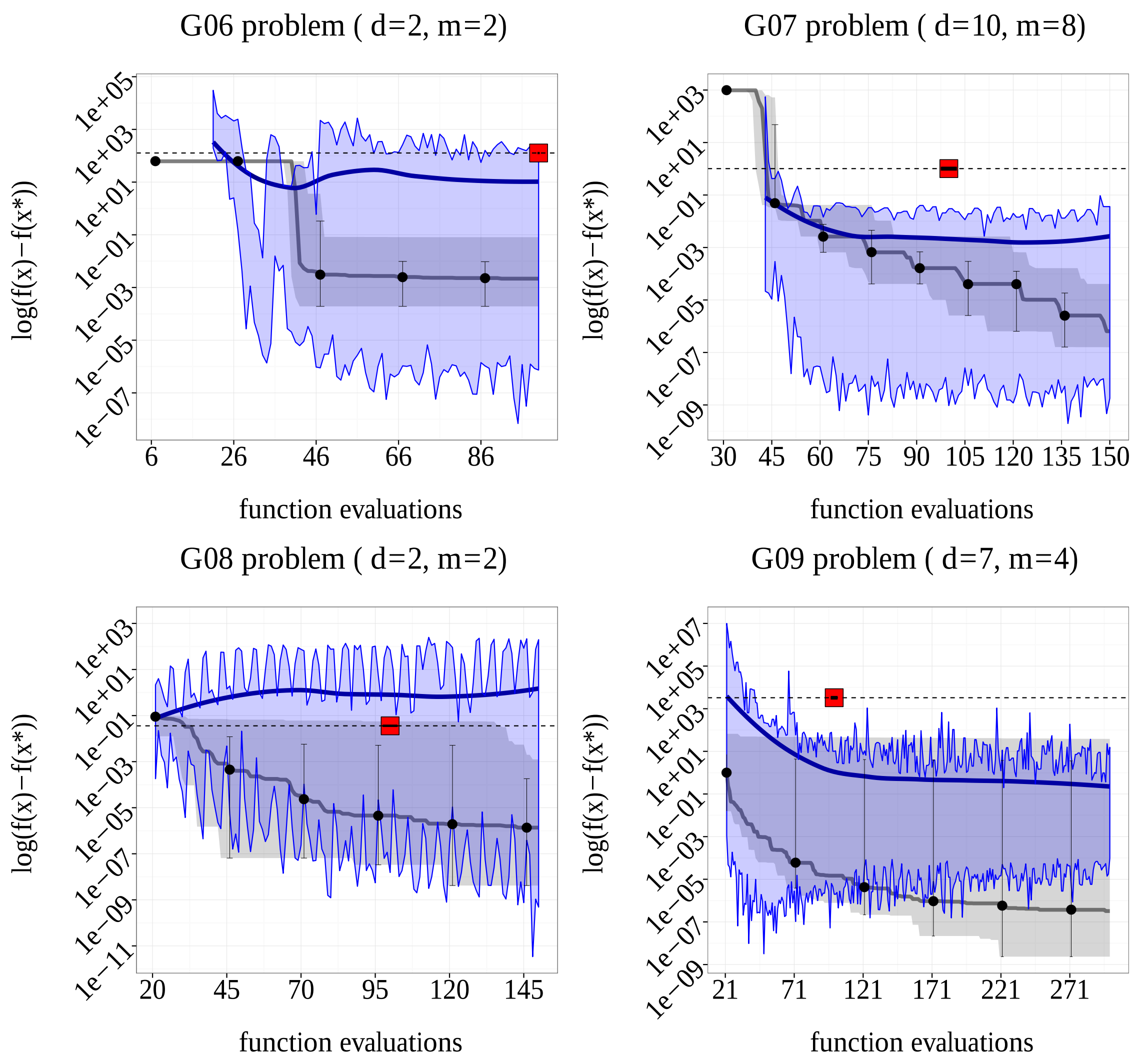


Figure 4.4: Manually tuned COBRA-R optimization process for G06, G07, G08 and G09. The gray curve is representing the median error per iteration for 30 independent trials. The gray shade around the median is showing the worst and the best error. The red square is the result reported in [34] after 100 iterations. The blue curve is a fitted smoother for the averaged RBF approximation error, shown in logarithmic scale $log(|S_0^{(k)}(\vec{x}_k) - f(\vec{x}_k)|)$, where $S_0^{(k)}$ is the RBF model of the fitness function in the k-th iteration trained with $k-1$ iterates excluding x_k. The blue ribbon around the averaged approximation error is representing the minimum and maximum approximation error among 30 runs in every iteration.

Table 4.4: Performance of the COBRA-R optimization algorithm on G-functions with tuned parameters for each problem. The statistics of the final feasible objective value $f(x)$ are determined based on 30 independent runs.The statistics on the optimization error $f(x) - f(x^*)$ for the same runs are listed in Table 4.3.

	optim	Best	Median	Mean	Worst	sd	ffc
G01	-15.00000	-15.00000	-14.99992	-14.37540	-12.45300	1.062	100
G03MOD	-0.69315	-0.69315	-0.69314	-0.69312	-0.69283	0.000	400
G04	-30665.53867	-30665.53867	-30665.53867	-30665.53867	-30665.53866	0.000	200
G05	5126.49748	5126.49812	5126.49841	5126.49915	5126.52284	0.004	200
G06	-6961.81474	-6961.81371	-6961.81173	-6961.80809	-6961.73258	0.015	100
G07	24.30620	24.30621	24.30621	24.30621	24.30625	0.000	200
G08	-0.09583	-0.09583	-0.09582	-0.09582	-0.09581	0.000	200
G09	680.63006	680.63006	680.63006	682.45706	719.04006	7.053	300
G10MOD	7049.24802	7049.25311	7049.33514	7049.40852	7050.56202	0.256	300
G11	0.75000	0.75000	0.75000	0.75000	0.75000	0.000	100

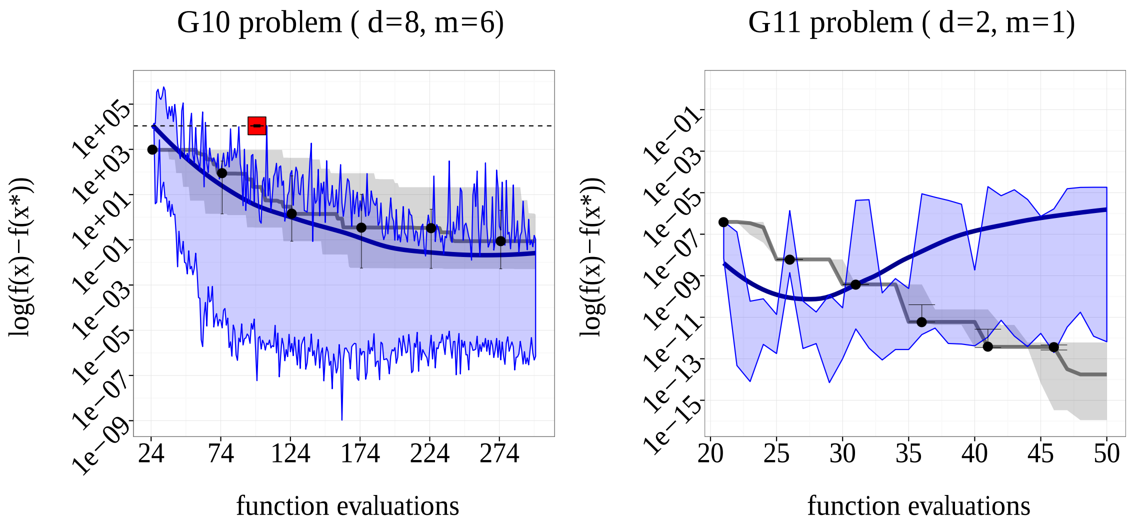


Figure 4.5: Manually tuned COBRA-R optimization process for G10 and G11. The gray curve is representing the median error per iteration for 30 independent trials. The gray shade around the median is showing the worst and the best error. The red square is the result reported in [34] after 100 iterations. The blue curve is a fitted smoother for the averaged RBF approximation error, shown in logarithmic scale $log(|S_0^{(k)}(\vec{x}_k) - f(\vec{x}_k)|)$, where $S_0^{(k)}$ is the RBF model of the fitness function in the k-th iteration trained with $k-1$ iterates excluding x_k. The blue ribbon around the averaged approximation error is representing the minimum and maximum approximation error among 30 runs in every iteration.

Table 4.5: Parameter setting used to determine the best results shown in Table 4.4 and Table 4.3. Initialization: the approach used to generate the initiative population, in parenthesis size of the initial population is shown. d is referring to the dimension of the problem. IOptimizer: Internal optimizer. DRC: Distance Requirement Cycle. SCALE: ✓ means that input space is normalized or constraint and/or objective function(s) are modified. RI: repair infeasible mode. (0: not Repaired, RI-2: RI-2 repair algorithm proposed in [21], CHO: Chootinan repair algorithm proposed in [6]).

	Initialization	IOptimizer	DRC	SCALE	RI
G01	LHS($3 \cdot d$)	COBYLA	$\{0.3, 10^{-2}, 10^{-3}, 5 \cdot 10^{-4}\}$	✗	0
G03	LHS(200)	COBYLA	$\{0.3, 10^{-2}, 10^{-3}, 5 \cdot 10^{-4}, 0\}$	✓	0
G04	LHS(20)	NMKB	$\{0.3, 10^{-2}, 10^{-3}, 5 \cdot 10^{-4}, 0\}$	✗	2
G05	BIASED(20)	COBYLA	$\{3, 0.1, 10^{-2}, 5 \cdot 10^{-3}, 0\}$	✓	2
G06	LHS($3 \cdot d$)	NMKB	$\{0.03, 10^{-3}, 10^{-4}, 5 \cdot 10^{-5}, 0\}$	✗	0
G07	OPTIMIZED($3 \cdot d$)	COBYLA	$\{0.03, 10^{-3}, 10^{-4}, 5 \cdot 10^{-5}, 0\}$	✗	0
G08	LHS(20)	ISRES	$\{0.1, 10^{-2}, 10^{-3}, 5 \cdot 10^{-4}, 10^{-6}\}$	✗	0
G09	OPTIMIZED($3 \cdot d$)	COBYLA	$\{0.03, 10^{-3}, 10^{-4}, 5 \cdot 10^{-5}, 0\}$	✗	0
G10	BIASESD($3 \cdot d$)	COBYLA	$\{0.03, 10^{-3}, 10^{-4}, 5 \cdot 10^{-5}, 0\}$	✓	2
G11	LHS(20)	COBYLA	$\{0.03, 10^{-3}, 10^{-4}, 5 \cdot 10^{-5}, 0\}$	✗	0

have the COBRA result after 100 iterations and we started to plot the optimization process for G03 after 200 iterations.

Comparison with COBRA [34]

Overall, we can claim that tuned COBRA-R is able to successfully optimize 10 problems out of 11 tested ones. In comparison with the results gained by the COBRA [34], the median of all results is significantly better than the median reported by Regis in [34]. In addition, we have results for two more functions (G01 and G11) which are also solvable by COBRA-R and the median of the results can approach the absolute optimum within a very few number of iterations after the initialization. Although there are several runs for the problems G01 and G09 which get stuck and do not improve, most of the other trials converge to the optimum.

Initialization

The COBRA-R optimization framework needs to be initialized with a population size of at least $d+1$ individuals due to the limitation imposed by the RBF surrogate approach. The size of the initial population and the method of generating this

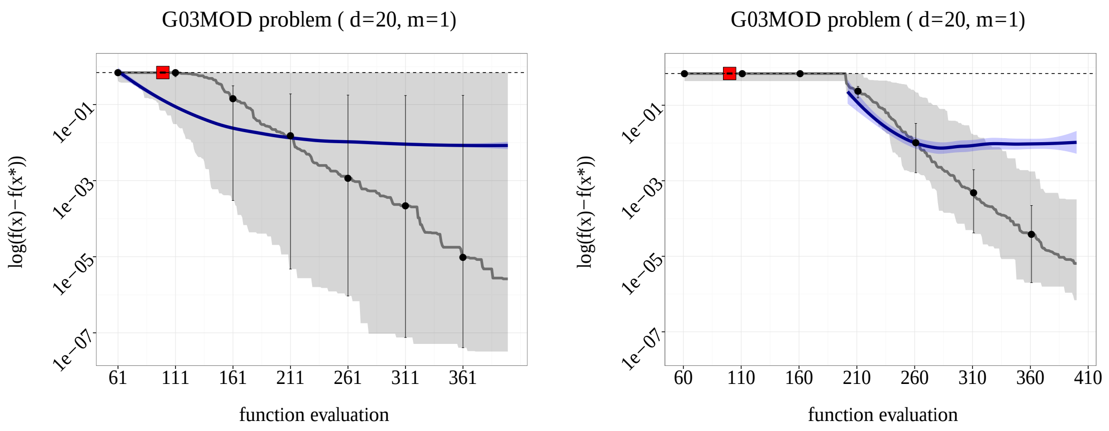


Figure 4.6: Impact of the initial population size on the COBRA-R performance for G03. A similar parameter setting is used for both tests and the only difference lies in the size of initial population. Left: initial population of size $3 \cdot d = 60$, Right: initial population of size $10 \cdot d = 200$. The test with more investment on the initial phase, starts the optimization process with a 10 times smaller approximation error for the fitness function, shown by blue curves.

population is an important parameter which can influence the quality of the objective and the constraint(s) model(s) and as a result, the quality of the final results.

In general, our experiments showed that most of the G-problems require a larger initial population than $d+1$ suggested by Regis [34]. As it is shown in Table 4.5, in many cases a population size of $3 \cdot d$ is required.

Also, some problems like G03 require even a larger initial population than $3 \cdot d$. We take a closer look at this problem. First, COBRA-R is not achieving any good result on the original G03 toy problem due to the very large value of FR. Therefore, a logarithmic transformation is utilized for its objective function which is described in [34, 36]. Whenever we mention G03MOD, we are referring to the G03 problem with a modified objective function. Second, the impact of the initial population size on the performance of the COBRA-R optimization for G03MOD is shown in Figure 4.6. In this figure, we are showing the COBRA-R optimization process with the same parameter setting and the only difference is the size of the starting population (Left: $3 \cdot d = 60$, Right: $10 \cdot d = 200$). The existence of several runs which do not improve for the case with the smaller population size, shows us that the model trained based on some initial designs are not good enough and the new iterates generated during the process are not correcting the model in the interesting region. Investing more evaluations for the initial phase is of benefit to start with a better fitness model (10 times smaller approximation error after initialization phase) for all 30 trials, but with this drawback that the optimization process has less iterations to move toward the optimum.

As it is described in Section 2.4.3, the initialization phase can be done by various approaches. One of the common means used for our tests is a random generation based on the Latin hypercube design [41]. The benefit of this approach is that there is no need of any extra knowledge for the starting point. The Biased and Optimized approaches can be used only if we have a good starting point. For some problems such as G07, G09 and G10 a random initialization appeared to be insufficient. There is no recommended starting point for these two test problems in the literature. We started from a point in the neighborhood of the optimum to generate Biased or Optimized initial population for G07, G09 and G10.

Internal Optimizer

A careful selection of the internal optimizer is another important factor for conducting results with good quality by means of our approach. For most of the tested problems COBYLA [31] is performing well as the internal optimizer. But for problems with many local optima COBYLA fails. The point is that a local optimizer typically cannot solve optimization problems with multimodel objectives. The internal optimizers in COBRA-R are basically performing an optimization on the surrogates instead on the real models. A reasonable surrogate model for a multimodel problem normally should be also mutilmodal. Therefore, we cannot expect from a local internal optimizer to find the global optimum of a problem with many local optima. G02 and G08 are examples for such problems. Using the global optimizer ISRES [38] as the internal optimizer for G08 and G02-2d is recommended. But, when the dimension of the problem is increased ISRES cannot be an appropriate choice for the internal optimizer because of high computational time imposed in every iteration by such population based strategy.

Distance Requirement Cycle

A careful selection of the distance requirement cycle is a rather complex task. Since even a small adjustment – such as one extra element or less in the cycle – can completely change the behavior of the whole optimization process. But, what our experiment shows is that adding a 0.0 value to the cycle is necessary for some problems and does not have many negative influences even if it is not required. For problems with very large FR, a small move in the input space induces large changes in the objective, therefore, we should make sure to have small steps. But depending

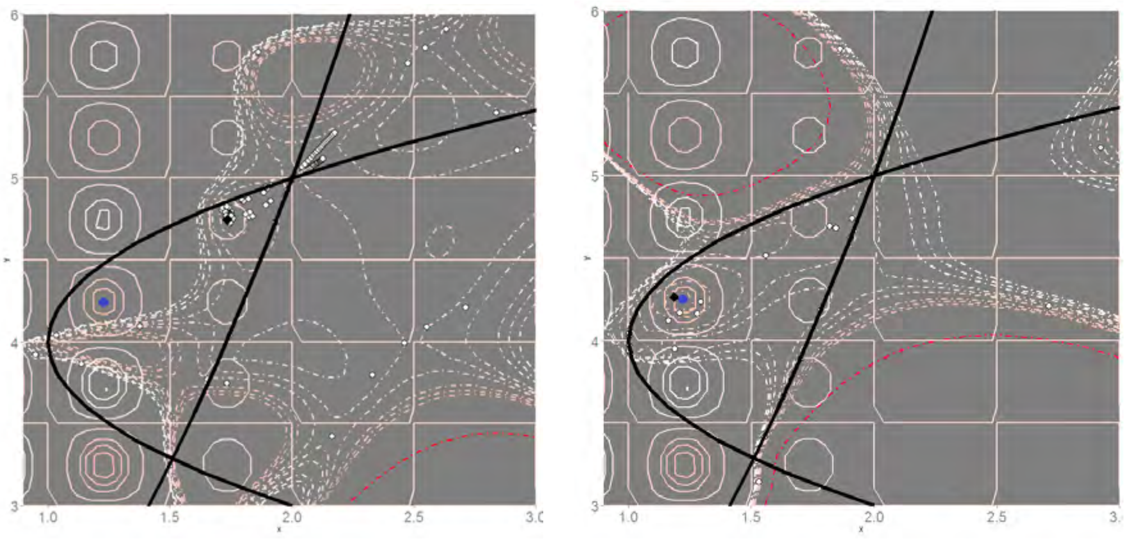


Figure 4.7: Impact of the internal optimizer selection on the COBRA-R performance for G08. Left: COBYLA is the internal optimizer, Right: ISRES is the internal optimizer. The solid curves are representing the contour line for the true objective function and the dashed curves are the surrogate model. The black solid curves are representing the constraint boundary. The color range from red to white is representing the objective value. Darker colors have lower values. The black point is the best ever found individual and the white points are showing the population of all evaluated points. The blue point is showing the real optimum. This figure is zoomed in to the feasible region restricted by the black curves but the actual search space is larger than what is shown in this figure.

on the type of the problem a small move can have different meanings. Since, in every iteration the surrogate models are changing, we are not expecting to have many repeated points during the COBRA-R process even in presence of the 0.0 element in the cycle. On the other hand, there are problems that need more exploration. Therefore, elements with larger values in the cycle are helpful. G01 is one of these type and adding a large element like 0.3 to the set of DRC is required. As an illustration, the influence of different selection of DRC for G02-2d is shown in Figure 4.8. Figure 4.8 clearly shows that the DRC in the right figure has a better exploration of the search space and approaches the true optimum far better.

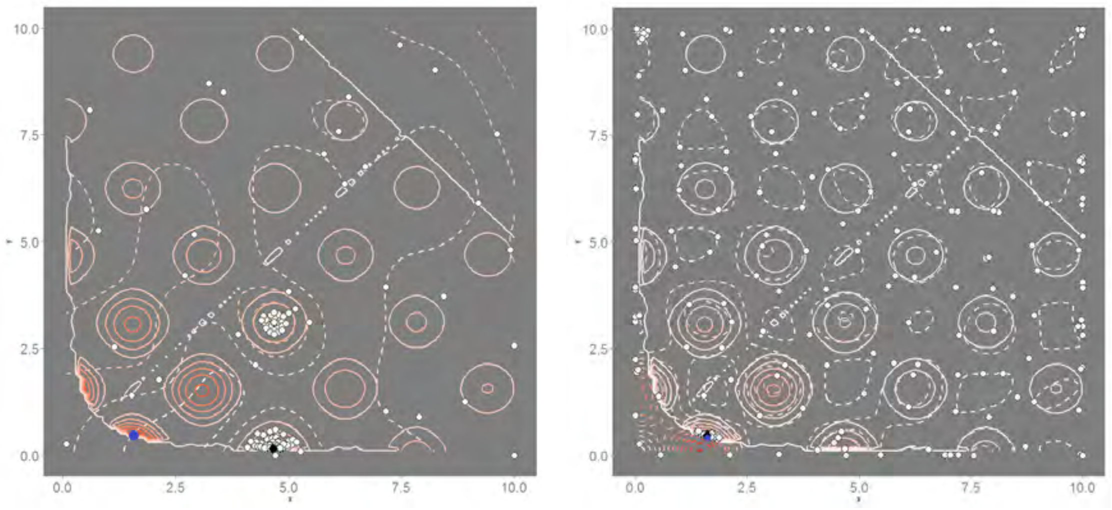


Figure 4.8: Impact of DRC selection on COBRA-R performance for G02 in 2 dimensions. Left: $\Xi = \{0.01, 0.001, 0.0005\}$, Right: $\Xi = \{3, 0.30.2, 0.1, 0.0\}$. The solid curves are representing the contour line for the true objective function and the dashed curves are the surrogate model. The black solid curves are representing the constraint boundary. The color range from red to white is representing the objective value. Darker colors have lower values. The black point is the best ever found individual and the white points are showing the population of all evaluated points. The blue point is showing the real optimum.

Scaling the Input Space

Normalizing the input space turns out to be very important for the G05 problem with a very large value of the input space elongation ($ISE = 1090, 91$). Our experiments show that COBRA-R cannot approach the optimum of the G05 problem unless the input space is scaled to $[0, 1]^d$.

Scaling Objective Function

Earlier, we had this idea of small elements in DRC for problems with large FR, but this is not adequate for problems like G03 with extremely large FR. The solution is to apply a logarithmic transformation to the fitness function of this problem to reduce the steepness of the objective function and make it possible to be modeled with RBF interpolation approach. The logarithmic transformation that we have used is described in [34, 36].

Scaling the Constraint Function(s)

Problems like G10 with very large GR appeared to be a challenge for the COBRA-R method since 3 constraints are very steep in comparison with 5 others. Hence, the constraints cannot be removed easily. Regis, in [34], modified the G10 problem by applying a logarithmic transformation (described in [34, 36]) to the three constraints with a large range of variation. We achieved good results for the G10 problem by normalizing them in a linear manner. The min-max range of each constraint over the search space was measured in a prior search and every constraint is divided by the determined values. We believe, our approach for modifying the constraint functions is more effective. Although applying a logarithmic transformation on the steep constraint functions helps us to have a better surrogate model for the underlying constraints, still some constraints can have a larger variation range in comparison with others. Normalization of all constraints helps to weight all the constraints equally and avoid ignoring the effect of some constraints and emphasizing on the other ones.

Repair

It is shown in a recent study in [21] that the repairing algorithm embedded in the COBRA-R framework does not have any large affect on the final quality of optimization results for G-problems.

4.1.3 Performance of Self-Adaptive COBRA-R

Although the results presented in the latter section have a good quality, the parameter selection and the function modification had to be done manually, mainly based on our extra information from the problems. This information usually is not known for black-box problems. Hence, we introduced three extensions to tackle the possible challenges automatically. We name the extended framework self-adaptive COBRA-R. In this section, the self-adaptive COBRA-R will be discussed and the results obtained with this approach will be presented.

The self-adaptive COBRA-R or SACOBRA-R algorithm includes three additional extensions in comparison with the COBRA-R optimization framework. First, we explain the input space rescaling step which occurs before the initialization. Second, we introduce an approach for starting the internal optimizer from a randomly selected point in the search space. Third, the parameter/function(s) adaptation phase which takes place after the initialization step will be described in detail. After describing these extensions, we explain how the initialization step is done in SACOBRA-R.

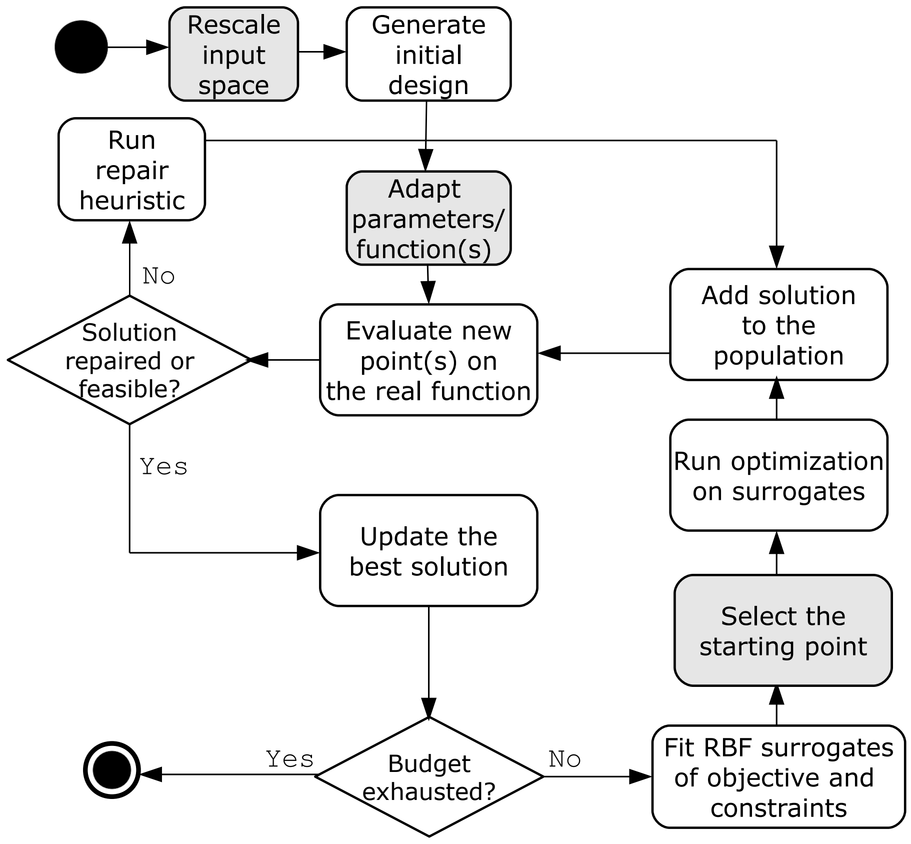

Rescale input space
Generate initial design
Run repair heuristic
Adapt parameters/ function(s)
No
Solution repaired or feasible?
Evaluate new point(s) on the real function
Add solution to the population
Yes
Run optimization on surrogates
Update the best solution
Select the starting point
Yes
Budget exhausted?
No
Fit RBF surrogates of objective and constraints

Figure 4.9: Flowchart of the Self-adaptive COBRA-R algorithm. The self-adaptive COBRA-R algorithm is a variant of COBRA-R [20] with three extra extensions shown with gray blocks.

Additionally, we explain how the internal optimizer is selected for SACOBRA-R. Furthermore, the results achieved by SACOBRA-R are presented and analyzed.

Rescaling the Input Space

The G-problems have a different input space restricted by the bound constraints $\vec{x} \in [\vec{a}, \vec{b}]$. In [34], it is mentioned that all problems passed to the COBRA optimization framework [34] are rescaled from $[a, b]$ to $[0, 1]^d$. We have applied an input space rescaling only to the problems with large ISE (described in Section 4.1). It is important to note that the input space elongation is defined based on the bound constraints of the problem (lower and upper bounds, $\vec{a}$ and $\vec{b}$, resp). This value

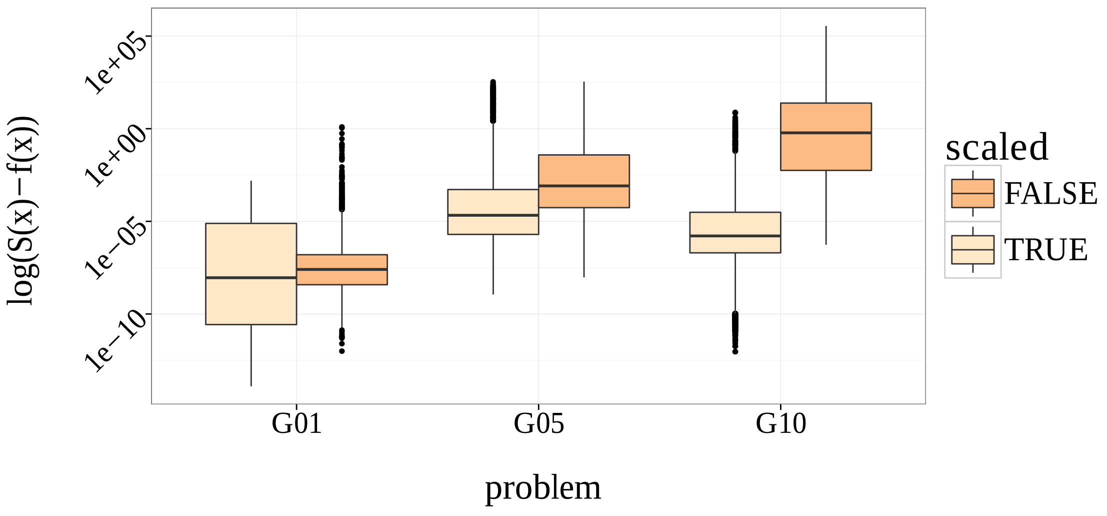


Figure 4.10: Impact of the input space normalization on the approximation error ($|S_0^k - f(x_k)|$) of the objective function for G01, G05 and G10 with large ISE. The results are drieven out of 30 independent trials for each setting. S_0^k is the RBF model of the objective function in the k-th iteration trained with $k-1$-th iterates excluding x_k. It is apparent that scaling the input space in all three cases has a significant impact on the accuracy of the objective model.

is known for black-box problems without any further evaluation or analysis. The bounds of a problem describe the range of variation of each variable and typically, these bounds are known. Therefore, in the self-adaptive COBRA-R framework, before generation of the initial population, the input space is automatically rescaled only for problems with large ISE. Among our test problems, G01, G05 and G10 have a relatively large input space elongation (according to Table 4.1).

In Figure 4.10, the boxplot of the objective function approximation error for three G-functions with and without normalization of the input space is plotted. It is evident, that in all three cases the scaling of the input space has a notable influence on the accuracy of the RBF model for the fitness function. This influence is significant for the G10 problem. It is surprising that without rescaling the input space the objective approximation error is very large for the G10 problem which has a simple linear polynomial $f_{G10}(x) = x_1 + x_2 + x_3$ objective function. The reason behind such large error for modeling such a simple objective function can be traced back to numerical issues of RBF modeling. It is mentioned in Section 2.4.1 that a cubic radial basis function $\phi(r) = r^3 = ||x - x_i||^3$ with combination of a polynomial tail is utilized to model the objective and constraint functions. Consider, x varying in a range of $[100, 10000]$ which is the case for G10, this large values in the input space lead to very large values for the output of the cubic radial basis function used for interpolation purpose and these large values neglect the existence of the polynomial tail in practice. This is why the surrogate model predicts wrong values. This problem is resolved by normalizing the input space and rescaling the bound constraint from $[a, b]$ to $[0, 1]^d$.

Random Start Algorithm

We observed in our earlier experiments (shown in Section 4.1.2) that for several test sets – despite the fact that the median of the 30 trials is converging to the optimum very fast – there are some trials which never improve after the initialization phase. Most likely, the search gets stuck in a wrong region (in one of the local optima) due to the initial model obtained based on the initial population. We assume two possible scenarios for such a situation: 1. The true function is multimodal and the surrogate model did not model the global optimum due to lack of information about the interesting region. 2: The surrogate model is accurate enough in the interesting region but the internal local optimizer is not successful in escaping the local optima. It seems like the iterates generated during the optimization process are not improving the results; therefore, certain randomly generated initial designs never lead to a good result.

For instance, the G01 problem is facing such difficulties for 10 out of 30 runs. For this reason, we propose an approach which occasionally selects a random point in the search space and starts the internal search with this point, instead of always starting the internal search from the best found individual.

This idea can be implemented in various fashions: One may define a fixed probability P_r to start the internal search from a random point. We believe, this approach can be problematic, since a suitable value for P_r may vary for different problems, which would require additional tuning effort. An alternative approach could trigger a random start if the minimization results did not improve for a certain number C_s of iterations in a row (count of stagnation). Hence, the random start does not impose any extra iterations for problems which are progressing well during the optimization procedure and is not called unless a stagnation of the optimization process is observed. The parameter C_s can be selected by the user. The described approach has its own drawback: if the search is converging slowly towards a local optimum then C_s may be reset and the search is trapped in the neighborhood of a local optimum. For this reason, we combined the first and second mentioned approaches. In the k-th iteration, the probability to start the internal optimization with a random point in the search space instead of the best point found by the optimization procedure, is P_k. The random start probability $P(k)$ is decayed after each iteration, following a sigmoid function as described in Equation 4.8.

$$P(k) = \frac{P_{max} - P_{min}}{2} \cdot \tanh(-(k - T)) + \frac{P_{max} + P_{min}}{2}, \tag{4.8}$$

we select the parameters as following: $P_{min} = \frac{P_{max}}{10}$, $P_{max} = 0.3$ and $T = 15 + 3 \cdot d$.

The probability of starting the internal search from a random point is decreasing over time. A high probability of a random initialization during the early iterations is chosen since we intend to correct the model or find the interesting region during the early iterations. A low probability of random initialization during the last iterations is due to the assumption that the interesting region is typically found in the last iterations and it is preferable to spend the limited number of remaining function evaluations on the improvement of the model in the interesting region and to avoid high rate of exploration.

Algorithm 1 is embedded in the COBRA-R framework as a step before optimization on the surrogates. This is shown in Figure 4.9.

Applying the random start algorithm as an extension to the COBRA-R optimization framework appears to be beneficial for several test problems such as G01, G05 and G08 in comparison to the results with manual tuning, as presented in Section 4.1.2. The top-left plot in Figure 4.12 is showing the optimization progress for the G01 test problem with the possibility of a random initialization of the internal optimizer. It is apparent that all 30 runs converge to the optimum, while only 90% of the trials with manually tuned parameters and without random starting algorithm tend to converge to the optimum. In addition, the optimization processes for G08 shown in Figure 4.13 and 4.4 indicates that the random initialization algorithm leads to a faster convergence. Moreover, G05 is another example where the random start algorithm shows benefits. The plot regarding the optimization process for G05 in Figure 4.12 shows that even the worst result for G05 has a small error of 10^{-3} after only 50 iterations, but without random start algorithm the worst case error drops only after 120 iterations.

One can pose this question that why the worst case errors for G09 problem is not improved after applying the random start algorithm? The answer is that the bad results observed for G09 are mainly due to the wrong model for the objective function. It is shown in Figure 4.13 that the surrogate approximation error is very large for G09. Therefore, a random starting algorithm cannot be effective for these types of problems. Table 4.6 lists the worst error obtained by the SACOBRA-R and COBRA-R algorithm among 30 independent runs for the G-problems. Comparing

Algorithm 1 Random Start Algorithm. Input: The counter for the number of iterations in a row without a progress c, The best point found in the k-th and $(k-1)$-th iteration, $x_{best}(k)$ and $x_{best}(k-1)$, resp. Output: The selected starting point $x_{start}(k+1)$ which will be passed to the internal optimizer in the $(k-1)$-th iteration, Updated counter c.

```
1:  $B$ : maximum budget for the function evaluations.
2:  $C_s := \frac{B}{10}$ : threshold on number of iterations without a progress.
3:  $k$ : The current iteration

4:  **if** The best result is updated in the last iteration $(x_{best}(k) < x_{best}(k-1))$ **then**
5:      $c \leftarrow 0$
6:  **else**
7:      $c \leftarrow c + 1$
8:  **end if**
9:  $\epsilon \leftarrow$ generate a random value $\in [0, 1]$
10: **if** $(c > C_s)$ || $(\epsilon < P(k))$ **then**        $\triangleright$ $P(k)$ is defined in Equation 4.8
11:     $x_{start}(k+1) \leftarrow$ a random point in the search space
12:     $c \leftarrow 0$
13: **else**
14:     $x_{start}(k+1) \leftarrow x_{best}(k)$
15: **end if**
```

these results shows that the worst case result is improved for 6 problems out of 10. This can be due to the embedding the random start algorithm extension.

Parameter/Function(s) Adaptation Algorithm

In Section 4.1.2, it is shown that considering the required problem adjustments before starting the COBRA-R procedure and selecting a suitable parameter setting to start up the COBRA-R, helps to minimize the G-problems efficiently. However, the parameter selection was based on our additional knowledge about the characteristics of the test problems. Since these information are not provided for black-box problems, it is desirable to add a mechanism to the COBRA-R framework with which the sensitive parameters can be adapted to the problem automatically. Also, it is important to decide whether the objective function should be transformed (e.g. logarithmic or scaled) for the purpose of the internal COBRA operations.

Table 4.6: Comparing the worst results achieved by SACOBRA-R and manually tuned COBRA-R. Performance of the Self-adaptive COBRA-R optimization algorithm on G functions to improve worst case result by means of random start algorithm. The same configuration is utilized for all problems except G02 and G08, which use ISRES as the internal optimizer instead of COBYLA. The worst optimization error $f(x) - f(x^*)$ are determined from 30 independent runs.

	COBRA-R	SACOBRA-R	sd	fe
G01	2.5E+00	**1.2E-06**	5.2E-07	100
G03	**3.2E-04**	0.994	0.419	300
G04	1.1E-05	**2.8E-07**	1.8E-07	200
G05	2.5E-02	**5.2E-03**	7.0E-04	200
G06	8.1E-02	**1.5E-02**	3.4E-05	100
G07	4.1E-05	**1.0E-05**	2.0E-06	200
G08	2.0E-05	**5.2E-11**	1.5E-11	200
G09	**3.8E+01**	2.2E+05	4.0E+04	300
G10	**1.3E+00**	2.0E+02	3.8E+01	300
G11	**1.6E-12**	6.4E-11	1.3E-11	100

As it was mentioned before, the correct choice of the distance requirement cycle plays an important role in the overall performance of COBRA-R. Moreover, the wise logarithmic or linear transformation of the fitness and constraint functions made a real difference in the accuracy of the surrogates and the quality of the optimization results.

In SACOBRA-R the initialization approach and the internal optimizer are fixed for all test-problems but the distance requirement is automatically adjusted during the parameter/function(s) adaptation step. In the parameter/function(s) adaptation step which is performed directly after the initialization phase, it is decided whether the objective function or the constraint function require to be modified, and if so, the modification is done according to the information driven from the initial population. In the following, we explain how the parameter/function(s) adaptation step works.

Objective Function Transformation As mentioned before, the best results presented in Section 4.1.2 for the G03 problem with a very large value of FR can just be achieved if the fitness function is modified by a logarithmic transformation. As shown in Table 4.1, G03 has a value of FR in the order of billions. This large range of variation for the fitness function can cause difficulties for the RBF modeling in order to fit a reasonably correct model. G03MOD is the modified version of G03 by applying a logarithmic transformation [34, 36] on the fitness function. Table 4.1 shows that FR is reduced by a factor of 10 million. Figure 4.11 illustrates how this transformation can play a role in improving the fitness model approximation error and as a result the enhancing the success of the optimization process. However, in black-box optimization we do not have the possibility to obtain the exact value of FR for different problems, a rough estimation of this value after the initialization step can give useful information about the range of the objective value. So, in the parameter adaptation step $\widehat{FR}$ (which is a rough estimation of FR described in Equation 4.2) is determined by measuring the fitness value for the $3 \cdot d$ individuals placed in the search space during the initialization phase. If the measured $\widehat{FR}$ after the first phase is larger than FR_u then a logarithmic transformation is automatically applied to the objective function. Therefore, no extra knowledge about the fitness behavior is required before the problem is passed to the self-adaptive COBRA-R. The FR_u is a constant which can be defined by the user. We set this value to 10^8.

We can conclude two important notes from Figure 4.11. 1. Before applying the logarithmic transformation to the objective function, the prediction error is about 10^5 in average, but with the logarithmic transformation the approximation error is reduced to 10^3 in average which is a great improvement. 2. As it is expected no good solution is found when the approximation error is wrong, whereas an improved model of the fitness function results in progress during the optimization process.

Distance Requirement Cycle We discussed earlier that a problem with very large FR is difficult for our surrogate approach to be handled. On the other hand, a large FR signifies that the fitness function is very steep and a small movement in the input space yields in a large change in the output space. Therefore, large values in the set of the distance requirement cycle can be harmful for such problems. We define two sets of distance requirement cycles. The first one – which is set as default DRC – is called large DRC and it is shown with Ξ_l in Table 4.11. This set has 5 elements with values differing from 0.3 to 0.0. This set of distance requirement cycles is designed to be used for problems which are not facing challenges induced by large values of FR. The second DRC is denoted as Ξ_s in Table 4.11 and has only two elements, one is the absolute zero and the second element is 0.001 which is added to prevent repetitive iterates. In the self adaptation step, after obtaining the $\widehat{FR}$ value, the distance requirement cycle is set to the short DRC Ξ_s if the rough approximation of the fitness range is larger than FR_l and smaller than FR_u. Otherwise the default DRC which is Ξ_l is utilized. FR_l is a constant value which can be assigned by the user. We fix this value to 10^3.

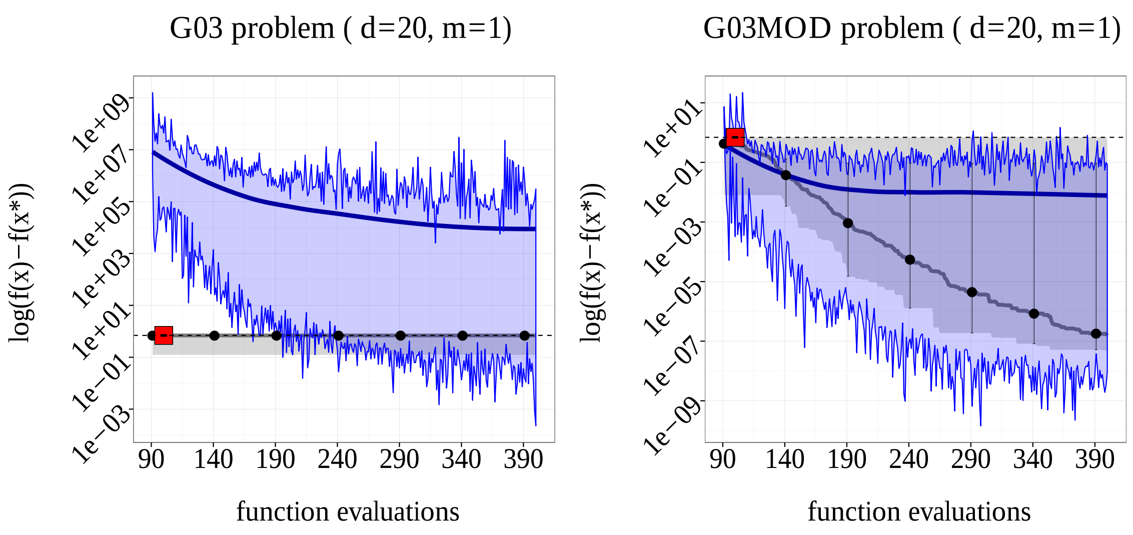


Figure 4.11: Impact of the logarithmic transformation of the objective function for problems with very large FR. Left: COBRA-R optimization process for the G03 problem without an objective function modification. Right: COBRA-R optimization process for the G03 problem after a logarithmic transformation is performed on the objective function. The gray curve is representing the median error for each iteration driven from the results of 30 independent trials. The gray shade around the median curve is showing the worst and best error. The blue curve is a fitted smoother for the average RBF approximation error, shown in logarithmic scale $log(|S_0^{(k)}(\vec{x}_k) - f(\vec{x}_k)|)$, where $S_0^{(k)}$ is the RBF model of the fitness function in the k-th iteration trained with $k-1$ iterates excluding x_k. The blue ribbon around the average approximation error is representing the minimum and maximum approximation error among 30 runs in every iteration.

Constraint Function Transformation In order to achieve successful results for constrained optimization problems with a surrogate assisted approach only an efficient model for the fitness function is not sufficient, constraint functions also should be modeled appropriately. Modeling a very steep objective function with a large variation range is a difficult task for RBF interpolation. This also applies to each single constraint function. It is also important to mention that in the COBRA-R optimization process constraints are mostly handled by means of penalty functions; therefore, a high variation range for some constraints in comparison to the others can result negligence of some constraints but stressing the others. The ratio of the constraint value ranges is shown as the GR feature in Table 4.1. G10 is one example of such a problem with highly varied ranges for different constraints (high GR). Regis in [34] suggests a logarithmic transformation for the constraints with large values. We believe that a logarithmic transformation is not the best solution for such a problem. Although this approach can help to reduce the variation range of every steep constraint function and lead to train better RBF models for them, some constraints still have larger ranges than the others. Therefore, we believe that normalizing all constraints and assuring that all of the constraints are changing in more or less similar ranges can be more effective. In the self-adaptive COBRA-R the modification of the constraint functions is done in the parameter adaptation step, right after the initialization phase. A rough estimation of GR is obtained from the information provided during the initialization step by locating $3 \cdot d$ points in the search space. If the rough estimation of the constraint ranges $\widehat{GR}$ is larger than GR_l, then every constraint function is adapted and divided by the min-max range of the constraint variation. Consider, a problem with m constraints. For the j-th constraint ($j = 1 \ldots m$) the min-max range is determined by calculating $\max_i(g_i^{(j)}) - \min_i(g_i^{(j)})$, where $g_i^{(j)}$ is the value of the j-th constraint function in the i-th point of the initial population.

The other parameters of SACOBRA-R such as initialization approach and internal optimizer are selected as follows:

Initialization Several algorithms for generating the initial population are implemented and embedded in the COBRA-R optimization framework such as Optimized, Biased and LHS which are described in Section 2.4.3. All of the mentioned initialization algorithms, with exception of LHS, require a feasible starting point. In many black-box problems like all G-functions a starting point is not given. Therefore, a random initialization by means of Latin Hypercube Design is our default choice for generating the initial population. Our experiments in the last section showed that $d + 1$ points are usually not sufficient to start with a good assistant model. On the other hand, as it is described the parameter adaptation step relies on the information coming from the initial population. Therefore, we consider a generation of $3 \cdot d$ random points for the initial population for all the test-problems.

Internal Optimizer Table 4.5 shows that most of the G-functions were performing well with COBYLA or NM as the internal optimizer. Both algorithms are based on the simplex method, COBYLA is more sophisticated and uses dynamic penalty internally. For this reason, we use COBYLA as the internal optimizer for all G-problems except G02 and G08 which have a multimodel fitness function with many local optima. For G02 and G08 ISRES is used as the internal optimizer.

The results of the optimization of the G-problems by means of the self-adaptive COBRA-R algorithms are listed in Table 4.7. Every test is repeated 30 times. For

Algorithm 2 Parameter/function(s) Adaptation Step. Input: $3 \cdot d$ points generated in the initialization phase and the objective and constraint functions. Output: The set of distance requirement cycle Ξ and the modified objective and constraint functions, $f(x)$ and $g^{(j)}$, $j = 1, \dots, m$, resp.

```
1: $\widehat{FR}$ : A rough estimation of $FR$ according to $3 \cdot d$ points located randomly on the
   search space in the initialization phase.
2: $\widehat{GR}$ : A rough estimation of $GR$ according to $3 \cdot d$ points located randomly on the
   search space in the initialization phase.
3: $\widehat{H}^{(j)}$ : A rough estimation of the min-max-range of the $j$-th constraint according to
   $3 \cdot d$ points located randomly on the search space in the initialization phase.
4: $FR_u := 10^8$
5: $FR_l := 10^3$
6: $GR_l := 10^5$
7: $m$ : Number of constraints.
8: $\Xi_s = \{0.001, 0.0\}$
9: $\Xi_l = \{0.3, 0.05, 0.001, 0.0005, 0.0\}$

10: if $\widehat{FR} > FR_u$ then
11:     Apply a logarithmic transformation to the objective function
12: else if $\widehat{FR} > FR_l$ then
13:     $\Xi \leftarrow \Xi_s$                              ▷ Adapt the DRC
14: else
15:     $\Xi \leftarrow \Xi_l$
16: end if

17: if $\widehat{GR} > GR_l$ then
18:     for $j = 1, \dots m$ do
19:         $g^{(j)} \leftarrow \frac{g^{(j)}}{\widehat{H}^{(j)}}$              ▷ Normalize constraint functions
20:     end for
21: end if
```

Table 4.7: Performance of the self-adaptive COBRA-R optimization algorithm on G functions. The same configuration is utilized for all problems except G02 and G08, which use ISRES as the internal optimizer instead of COBYLA. The statistics on the optimization error $f(x) - f(x^*)$ are determined from 30 independent runs. The statistics on the final optimization results $f(x)$ for the same runs are listed in Table 4.8

	Best	Median	Mean	Worst	sd	fe
G01	1.8E-08	2.9E-07	5.2E-07	1.2E-06	5.2E-07	100
G02-10d	1.4E-01	2.8E-01	2.8E-01	4.4E-01	7.0E-02	500
G03	2.2E-07	2.9E-03	0.36	0.994	0.419	300
G04	1.6E-08	1.5E-07	1.5E-07	2.8E-07	1.8E-07	200
G05	1.4E-03	1.5E-03	1.6E-03	5.2E-03	7.0E-04	200
G06	1.5E-02	1.5E-02	1.5E-02	1.5E-02	3.4E-05	100
G07	7.6E-09	6.4E-07	1.0E-06	1.0E-05	2.0E-06	200
G08	3.4E-13	9.0E-12	1.5E-11	5.2E-11	1.5E-11	200
G09	5.2E+01	7.0E+02	8.9E+03	2.2E+05	4.0E+04	300
G10	1.5E-05	1.2E-03	1.0E+01	2.0E+02	3.8E+01	300
G11	9.2E-15	2.5E-12	8.1E-12	6.4E-11	1.3E-11	100

Table 4.8: Performance of the self-adaptive COBRA-R optimization algorithm on G functions. The same configuration is utilized for all problems except G02 and G08 which use ISRES as the internal optimizer instead of COBYLA. The statistics on the objective value $f(x)$ are determined from 30 independent runs. The statistics on the optimization error $f(x) - f(x^*)$ for the same runs are listed in Table 4.7

	optim	Best	Median	Mean	Worst	sd	fe
G01	-15.00000	-15.00000	-15.00000	-15.00000	-15.00000	5.2E-07	100
G02-10d	-0.71355	-0.57052	-0.43815	-0.43111	-0.27057	7.0E-02	500
G03	-1	-1	-0.9941	-0.474133	-0.00000	0.42	300
G04	-30665.53867	-30665.53867	-30665.53867	-30665.53867	-30665.53867	1.8E-07	200
G05	5126.49670	5126.49811	5126.49816	5126.49834	5126.50191	7.0E-04	200
G06	-6961.81388	-6961.79856	-6961.79839	-6961.79840	-6961.79839	3.4E-05	100
G07	24.30621	24.30621	24.30621	24.30621	24.30622	2.0E-06	200
G08	-0.09583	-0.09583	-0.09583	-0.09583	-0.09583	1.5E-11	200
G09	680.63006	732.51012	1378.28968	9594.51480	218444.86203	4.0E+04	300
G10	7049.24802	7049.24804	7049.24920	7059.25564	7250.98785	3.8E+01	300
G11	0.75000	0.75000	0.75000	0.75000	0.75000	1.3E-11	100

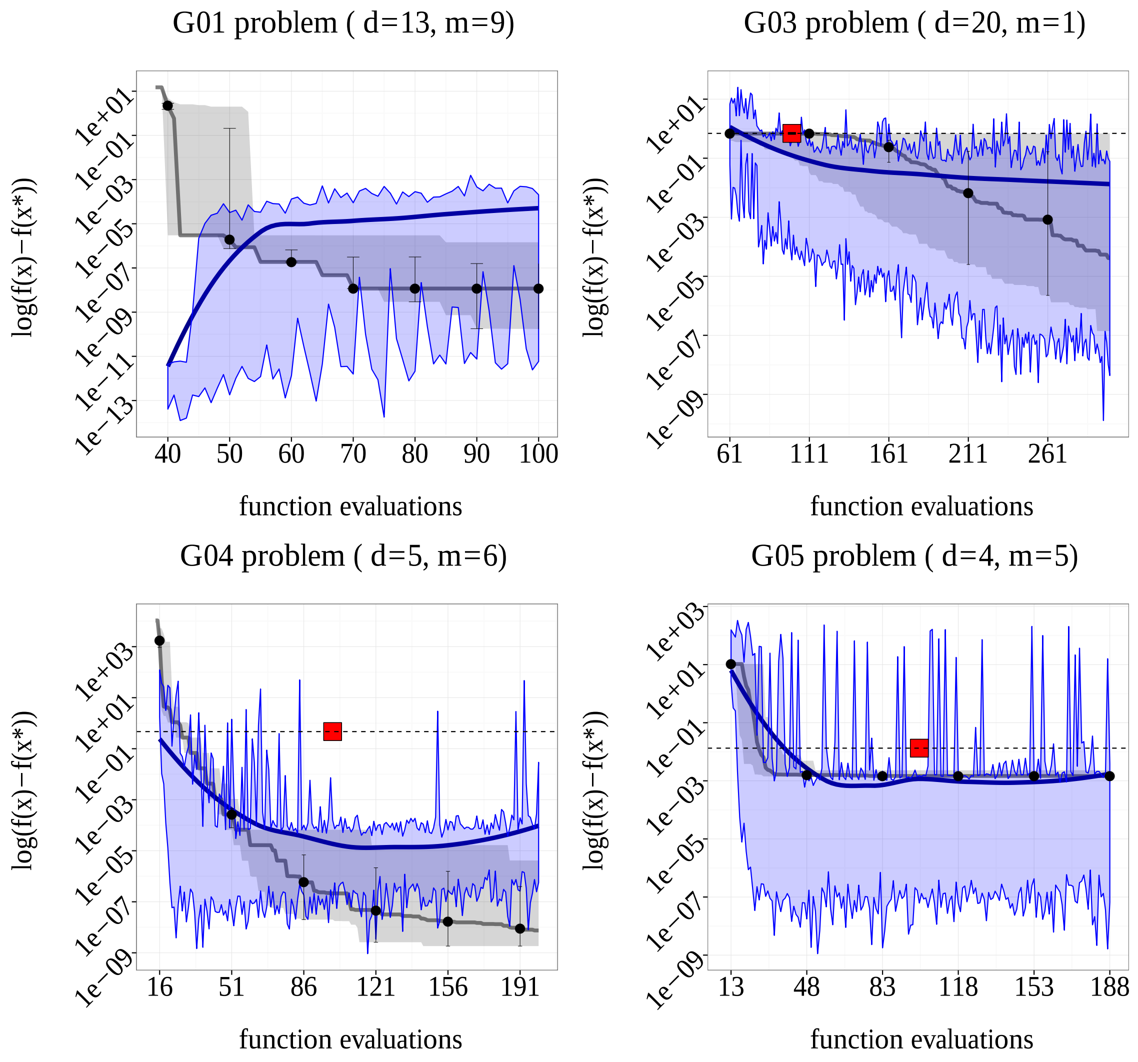


Figure 4.12: Self-adaptive COBRA-R optimization process for G01, G03, G04 and G05. The gray curve is representing the median error per iteration for 30 independent trials. The gray shade around the median is showing the worst and the best error. The red square is the result reported in [34] after 100 iterations. The blue curve is a fitted smoother for the averaged RBF approximation error, shown in logarithmic scale $log(|S_0^{(k)}(\vec{x}_k) - f(\vec{x}_k)|)$, where $S_0^{(k)}$ is the RBF model of the fitness function in the k-th iteration trained with $k-1$ iterates excluding x_k. The blue ribbon around the averaged approximation error is representing the minimum and maximum approximation error among 30 runs in every iteration.

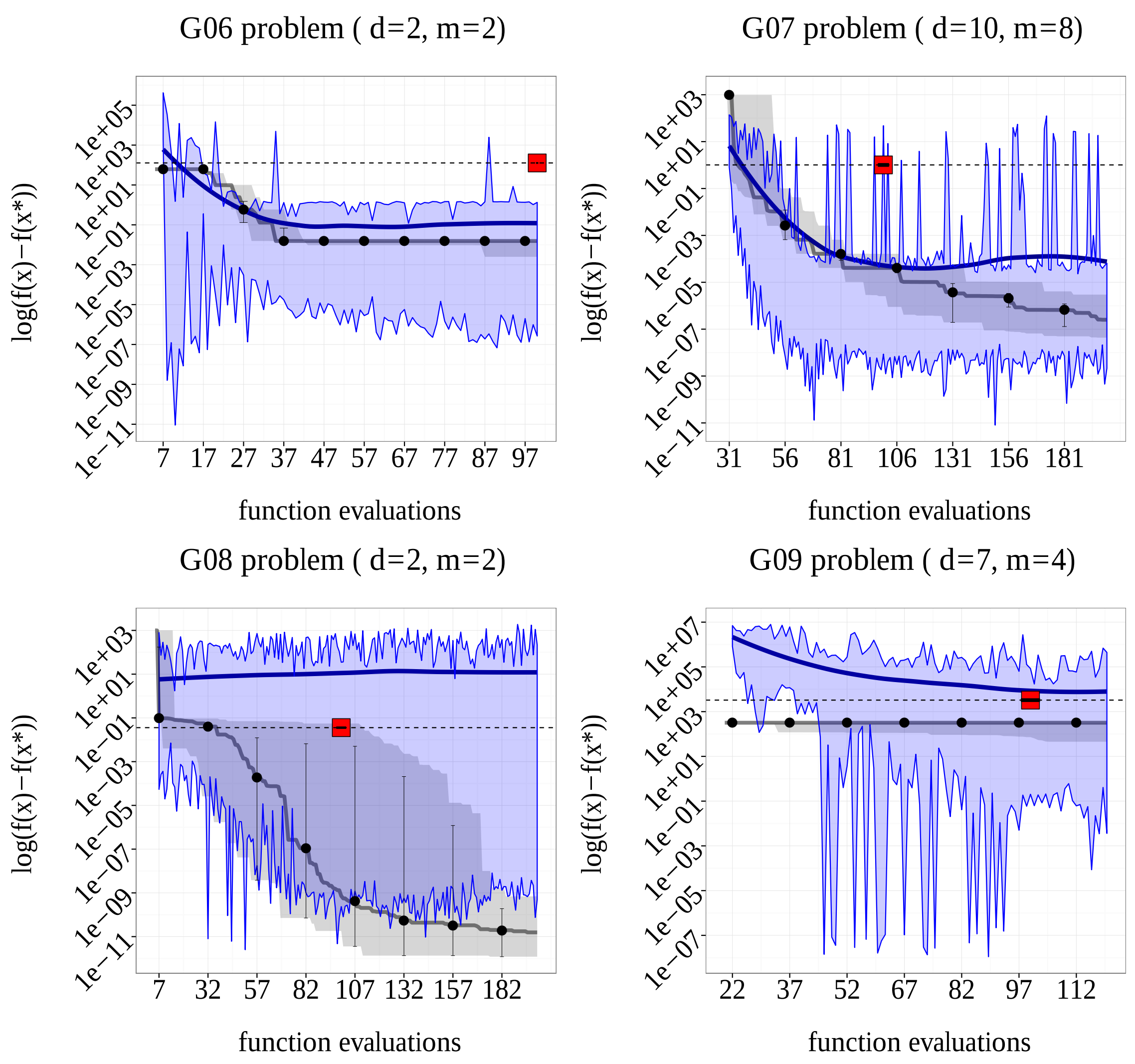


Figure 4.13: Self-adaptive COBRA-R optimization process for G06, G07, G08 and G09.The gray curve is representing the median error per iteration for 30 independent trials. The gray shade around the median is showing worst and the best error. The red square is the result reported in [34] after 100 iterations. The blue curve is a fitted smoother for the averaged RBF approximation error, shown in logarithmic scale $log(|S_0^{(k)}(\vec{x}_k) - f(\vec{x}_k)|)$, where $S_0^{(k)}$ is the RBF model of the fitness function in the k-th iteration trained with $k-1$ iterates excluding x_k. The blue ribbon around the averaged approximation error is representing the minimum and maximum approximation error among 30 runs in every iteration.

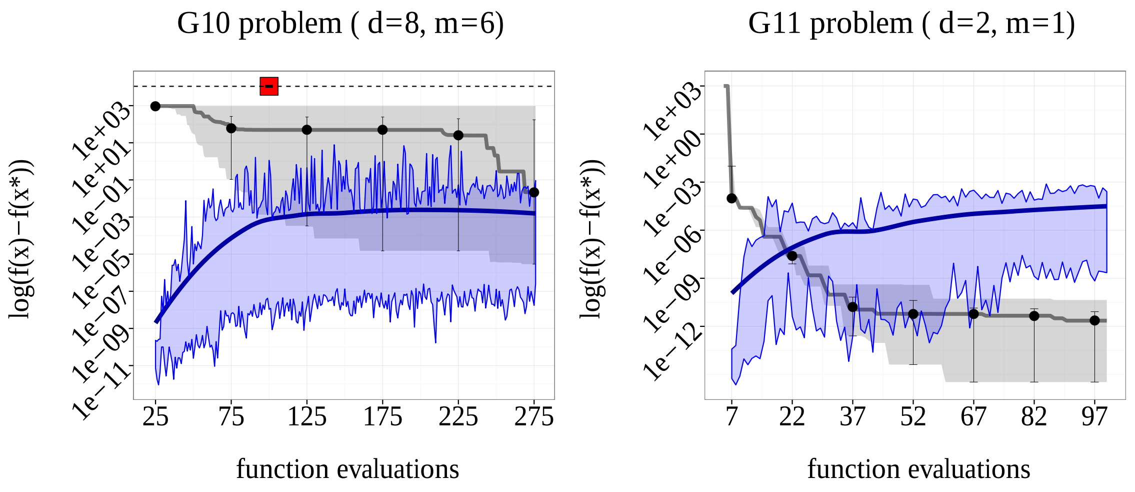


Figure 4.14: Self-adaptive COBRA-R optimization process for G10 and G11. The gray curve is representing the median error over iteration for 30 independent trials. The gray shade around the median is showing worst and the best error. The red square is the result reported in [34] after 100 iterations. The blue curve is a fitted smoother for the averaged RBF approximation error, shown in logarithmic scale $log(|S_0^{(k)}(\vec{x}_k) - f(\vec{x}_k)|)$, where $S_0^{(k)}$ is the RBF model of the fitness function in the k-th iteration trained with $k-1$ iterates excluding x_k. The blue ribbon around the averaged approximation error is representing the minimum and maximum approximation error among 30 runs in every iteration.

each problem a limited budget is given which is also listed as "fe", standing for function evaluation. For 9 out of 11 test problems, the best solution has an objective value with a distance of approx.10^{-2} or smaller from the objective value of the global optimum. The statistics regarding the final objective value obtained by self-adaptive COBRA-R for the same problems are given in Table 4.8. In order to have a better view on the self-adaptive COBRA-R optimization process, the optimization error $f(x) - f(x^*)$ over iteration for G01 to G11 (except G02) is shown in Figure 4.12, 4.13 and 4.14. According to the mentioned figures, the median error is converging towards the optimum for 9 out of 11 problems. Results show that self-adaptive COBRA-R is successful for all the G-problems except G02 and G09.

As shown in Figure 4.13, the algorithm is not performing successfully for the G09 problem. We believe this failure is due to the complex objective function of G09. In the same figure the mean of approximation error is shown with a blue curve. The extremely large values for the fitness approximation error is an indication that the optimization process fails due to the wrong model trained by RBF for the objective function. This issue can be a good starting point for future work to investigate more about the strengths and weaknesses of the cubic RBF interpolation and seeking for solutions to improve the weaknesses.

The G02 problem is not an easy benchmark for the COBRA-R algorithm and its variants. In this problem constraints are easy to resolve but the objective function with many local optima is not easy to be modeled with RBF functions. Therefore, a surrogate assisted approach is not recommended for such a problem. The difficulty to model the G02 problem grows higher with increasing the dimensionality. Figure 4.15 illustrates the self-adaptive COBRA-R optimization progress for G02-2d and G02-10d. It is apparent that the average approximation error is 10 times larger for G02-10d in comparison with G02-2d. Moreover, after 300 function evaluations no better point is found for G02-10d while in G02-2d after about 180 iterations many trials converge to the optimum. We can also observe from the same figure that the minimum approximation error is mostly very small ($\approx 1e - 11$) for both G02-2d and G02-10d. This indicates that the model is improving locally but nevertheless has wrong values in most of the search space since the average error is much larger.

In [34], Regis presented the COBRA optimization results for 9 of the G-problems. The median error by COBRA after 100 iterations is also plotted with a red square point in Figure 4.12, 4.13, 4.14 and 4.15. The results achieved in this study are significantly better than what is reported in [34], except for G03 and G09. Our results on these two problems are not worse than Regis' COBRA but also not significantl improved. Overall, we believe that self-adaptive COBRA-R outperforms COBRA [34] in solving the G-problems, because no tuning – automatic or manual – is done in [34] for approaching different problems, though the suitable parameter settings for COBRA are varying from problem to problem. Also, we believe that the utilized DRC in [34] is relatively large for many of G-problems.

Although we do not have any better results for the G03 problem after 100 iterations in comparison with [34], the convergence starts during later iterations. After 160 iterations the median of 30 runs is converging towards the optimum and after 300 iterations the median error is approx. 10^{-5}. Although many runs (out of 30) converge to the optimum, there are still a few runs which never progress within the 300 iterations. This is because the trained initial model, based on certain initial designs are wrong and the points added to the population during the optimization process are not good enough to rectify the model in the interesting region. It appears that this problem requires a larger initial population or a higher probability for starting from random points in the search space. It was shown in Section 4.1.2 that for the G03 problem it is worthy to spend a large number of function evaluations on the initial population (see Figure 4.6).

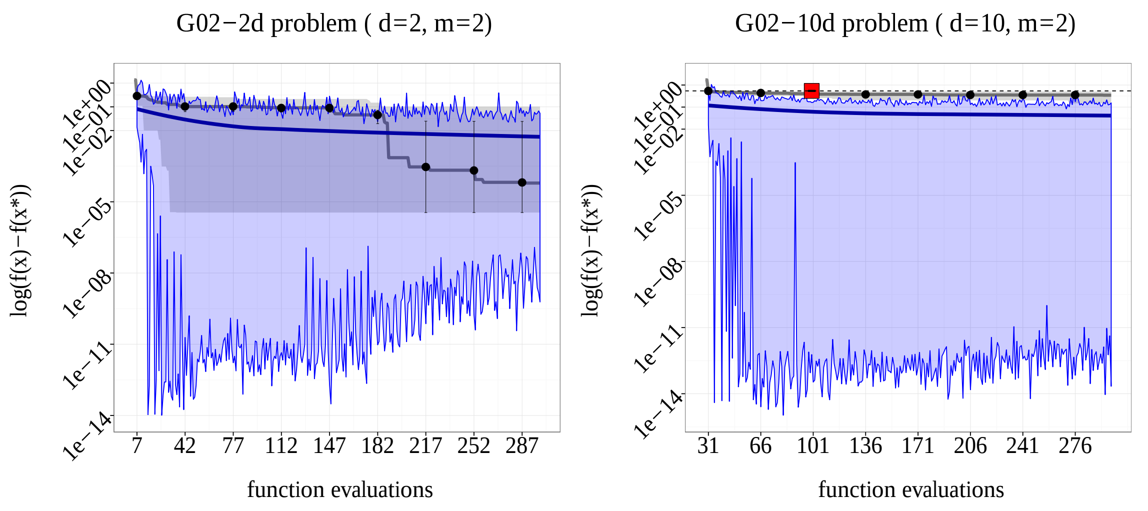
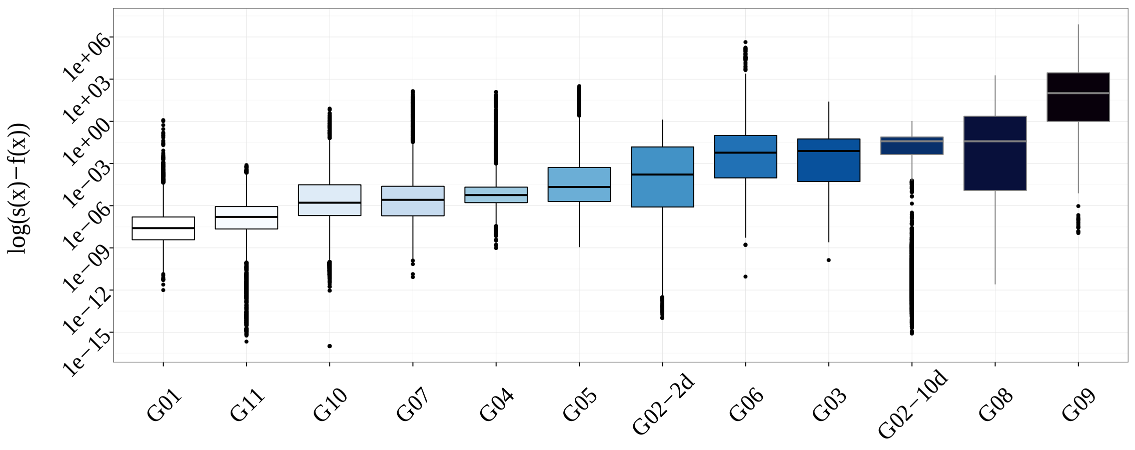


Figure 4.15: Self-adaptive COBRA-R optimization process for the G02 problem. Left: $d = 2$, Right: $d = 10$. The gray shade around the median is showing worst and the best error. The red square is the result reported in [34] after 100 iterations. The blue curve is a fitted smoother for the averaged RBF approximation error, shown in logarithmic scale $log(|S_0^{(k)}(\vec{x}_k) - f(\vec{x}_k)|)$, where $S_0^{(k)}$ is the RBF model of the fitness function in the k-th iteration trained with $k-1$ iterates excluding x_k. The blue ribbon around the averaged approximation error is representing the minimum and maximum approximation error among 30 runs in every iteration.

The reason behind the weak results of the self-adaptive COBRA-R for G09 was discussed before. Comparing our results with Regis shows that the COBRA [34] optimization framework did not solve G09 successfully as well. We assume that the failure for solving G09 with self-adaptive COBRA-R and COBRA [34] is due to the lack of a correct model for the objective function, as the objective function is very complex to be modeled with cubic RBF interpolation.

Figure 4.16 provides an overview for the prediction error of the RBF models trained for the objective function of all tested G-problems. The RBF approximation error of the fitness function $|S_0^k - f(x_k)|$ is basically the difference of the predicted value by RBF model and the real value. Assume, S_0^k is the RBF model of the objective function in the k-th iteration built on the $k-1$ points determined during the former iterations. It is important to note that the surrogate model in the k-th iterate is trained based on the all points in the population excluding $\vec{x}_k$. Otherwise, $|S_0^k - f(x_k)|$ would have a value of zero.



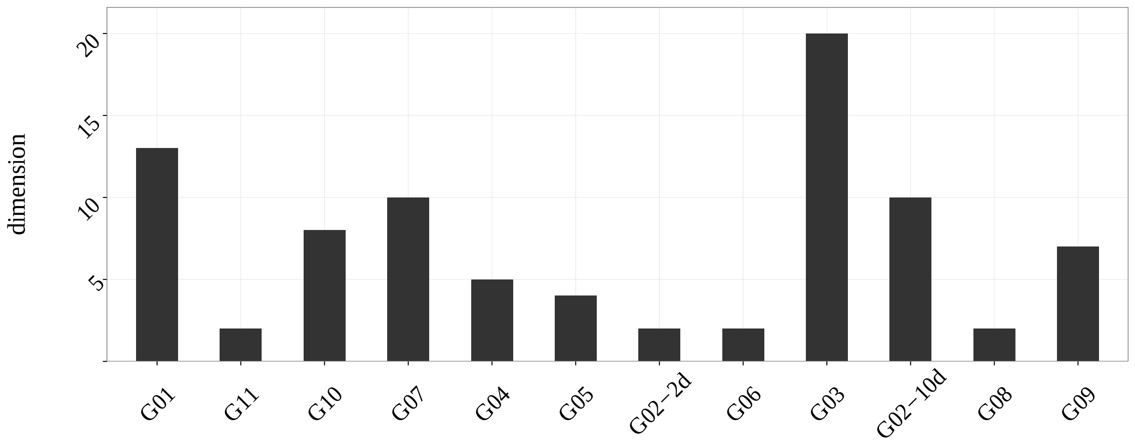


Figure 4.16: Top: Objective function approximation error for all tested G-problems during the self-adaptive COBRA-R optimization process shown in logarithmic scale. The approximation error of the objective function is described as $|S_0^{(k)}(\vec{x}_k) - f(\vec{x}_k)|$, where $S_0^{(k)}$ is the RBF model of the fitness function in the k-th iteration trained with $k-1$ iterates excluding x_k. Bottom: each bar is representing dimension of all tested G-problems. It is clear that the RBF interpolation performance is not dependent on the dimension of the problem.

In surrogate assisted optimization we cannot expect to minimize the objective function if the objective model is wrong. Also, the internal solvers can handle the constraints only if reasonable models for constraint functions are provided by the modeling approach.

It is apparent in Figure 4.16 that the RBF approximation error is not dependent on the dimension of the problem. We can see that almost never during the self-adaptive optimization process a good model is provided for G09. This is possibly due to the complexity of the G09 objective function and its large FR. Furthermore, we can observe that for the G08 problem with 2 dimensions the objective approximation error is relatively large comparing to the others. This is because the G08 problem has many local optima although the model becomes better in the interesting feasible region, most of the exploratory moves are in an unknown region for the model. This increases the approximation error in average. The G02-10d problem is also suffering from similar issues with multi-modality. The objective functions of G01 and G11 seem to be very easy for our approach to be modeled. As a result the convergence to the optimum is also very fast and within a few evaluations. After the initialization phase the search finds a solution in the neighborhood of the optimum with a small distance of ($< 10^{-7}$). The objective function of G01 is a polynomial with linear terms and one quadratic term. The objective function of G11 is a combination of a simple polynomial term and a radial basis function. It is necessary to mention that the boxplot shown as the approximation error of G10 is gained after the normalization of the input space. Otherwise, the error rate would be much higher. It is also clear that the approximation error shown for G03 problem is achieved during the self-adaptive COBRA-R optimization process, so the objective function is logarithmic transformed. G06 has the approximation error from very low values like 10^{-12} to large values like 1 but most of the time the approximation error is less than 10^{-2} which is moderately accurate. The reason can be traced back to the large FR value for G06. A steep function is a difficult task for RBF modeling. Additionally, we have G04 and G05 which have a slightly more complex objective function with mixed terms but not as complex or steep as G06 and G09.

4.1.4 Comparing the Performance of SACOBRA-R and COBRA-R

Table 4.9: Comparison of the error obtained by COBRA-R & self-adaptive COBRA-R. Best, median, mean, and worst optimization error $f(x) - f(x^*)$ and their standard deviation (sd) determined in 30 independent runs. Same number of function evaluations are performed by both algorithms. The number of function evaluations (fe) for every problem are listed in Table 4.8.

Fct.	Alg	best	median	mean	worst	sd
G01	COBRA-R	2.9E-07	7.5E-05	6.2E-01	2.5E+00	1.1E+00
	SACOBRA-R	**1.8E-08**	**2.9E-07**	**5.2E-07**	**1.2E-06**	**5.2E-07**
G03	COBRA-R	6.7E-07	**6.4E-06**	**2.5E-05**	**3.2E-04**	**6.4E-05**
	SACOBRA-R	**2.2E-07**	2.9E-03	0.36	0.994	0.419
G04	COBRA-R	**2.5E-10**	**6.3E-08**	6.7E-07	1.1E-05	2.1E-06
	SACOBRA-R	1.6E-08	1.5E-07	**1.5E-07**	**2.8E-07**	**1.8E-07**
G05	COBRA-R	**1.3E-05**	**3.0E-04**	**1.0E-03**	2.5E-02	4.5E-03
	SACOBRA-R	1.4E-03	1.5E-03	1.6E-03	**5.2E-03**	**7.0E-04**
G06	COBRA-R	**1.9E-04**	**2.2E-03**	**5.8E-03**	8.1E-02	1.5E-02
	SACOBRA-R	1.5E-02	1.5E-02	1.5E-02	**1.5E-02**	**3.4E-05**
G07	COBRA-R	4.0E-08	**6.3E-07**	2.7E-06	4.1E-05	7.6E-06
	SACOBRA-R	**7.6E-09**	6.4E-07	**1.0E-06**	**1.0E-05**	**2.0E-06**
G08	COBRA-R	3.4E-09	6.6E-07	1.5E-06	2.0E-05	3.5E-06
	SACOBRA-R	**3.4E-13**	**9.0E-12**	**1.5E-11**	**5.2E-11**	**1.5E-11**
G09	COBRA-R	**1.5E-08**	**3.5E-07**	**1.8E+00**	**3.8E+01**	**7.1E+00**
	SACOBRA-R	5.2E+01	7.0E+02	8.9E+03	2.2E+05	4.0E+04
G10	COBRA-R	5.1E-03	8.7E-02	**1.6E-01**	**1.3E+00**	**2.6E-01**
	SACOBRA-R	**1.5E-05**	**1.2E-03**	1.0E+01	2.0E+02	3.8E+01
G11	COBRA-R	**3.7E-15**	**2.3E-13**	**3.4E-13**	**1.6E-12**	**3.9E-13**
	SACOBRA-R	9.2E-15	2.5E-12	8.1E-12	6.4E-11	1.3E-11

Table 4.9 shows the statistics of the final error determined by self-adaptive COBRA-R and COBRA-R for 30 independent trials. Comparing the median error shows that self-adaptive COBRA-R is capable of achieving similar accuracy for 9 out of 10 problems. The good results achieved by COBRA-R for G09 are only determined if the initialization phase is done by the optimized approach with a given feasible point in the neighborhood of the optimum. This information is usually not available

and we did not consider any known starting point for G09 when using self-adaptive COBRA-R approach. The median error for 3 problems is improved by applying self-adaptive COBRA-R. The mean error of 4 tests is better by using self-adaptive COBRA-R. Except from G08 and G10, in all other cases the best error found by COBRA-R approach has smaller value. But, the worst error is improved for 6 tests which is mostly because of the random start algorithm extension used in self-adaptive COBRA-R.

In general, self-adaptive COBRA-R does not provide better optimization results comparing with manually tuned COBRA-R. But, it is worthy to note that the reasonable results achieved by self-adaptive COBRA-R are achieved with no need for parameter tuning. Real-world problems in industry are usually black-box and only parameter-free strategies are promising to address such problems.

4.1.5 Comparison with other Techniques

Table 4.10: Best (b), median (m) mean (avg) and worst (w) results and their standard deviation (sd) determined in 30 independent runs with different approaches. Average number of function evaluations (fe). The COBYLA approach very often returns slightly infeasible points. The COBYLA results listed in this table are determined by allowing 10^{-08} infeasibility, although still some solutions are infeasible. The number of the infeasible points returned by COBYLA is shown in parenthesis in front of the sd value.

Fct.	Optimum		SACOBRA-R	COBRA [34]	ISRES [38]	RGA 10% [6]	COBYLA [31]
		b	**-15.0**	NA	**-15.0**	**-15.0**	**-15.0**
		m	**-15.0**	NA	**-15.0**	NA	-13.83
		avg	**-15.0**	NA	**-15.0**	**-15.0**	-13.12
G01	-15.0	w	**-15.0**	NA	**-15.0**	**-15.0**	-10.1
		sd	5.2E-07	NA	**5.8e-14**	0.0	1.19
		fe	100	NA	350000	95512	12743.9
		b	-0.409403	NA	**-0.803619**	-0.801119	-0.272
		m	-0.346592	NA	**-0.793082**	NA	-0.197
		avg	-0.346592	NA	-0.782715	**-0.7857**	-0.203
G02	-0.80355	w	-0.281917	NA	-0.723591	**-0.745329**	-0.164
		sd	0.028	NA	2.2e-02	**0.0137**	0.023(5)
		fe	500	NA	349600	331972	97391.28
		b	**-1.0**	-0.8965	**-1.001**	-0.9999	**-1.0**
		m	**-0.9941**	-0.09	**-1.001**	NA	**-1.0**
		avg	-0.474133	0.00	**-1.001**	**-0.9999**	-0.47
G03	-1.0	w	0.000	0.00	**-1.001**	**-0.9997**	0.0

(Continued on next page)

Table 4.10: Best (b), median (m) mean (avg) and worst (w) results and their standard deviation (sd) determined in 30 independent runs with different approaches. Average number of function evaluations (fe).(continued)

Fct.	Optimum		SACOBRA-R	COBRA [34]	ISRES [38]	RGA 10% [6]	COBYLA [31]
		sd	0.42	0.03	**8.2e-09**	0.0	0.42(3)
		fe	300	100	349200	399804	31069.1
		b	**-30665.53867**	-30665.49	**-30665.539**	**-30665.5386**	**-30665.539**
		m	**-30665.53867**	-30665.15	**-30665.539**	NA	**-30665.539**
		avg	**-30665.53867**	-30665.07	**-30665.539**	**-30665.5386**	**-30665.539**
G04	-30665.53867	w	-30665.53867	-30664.58	**-30665.539**	**-30665.5386**	**-30665.539**
		sd	1.8E-07	0.04	**1.1e-11**	0.0	2.2e-07
		fe	200	100	192000	26981	418.6
		b	5126.4981	5126.5	**5126.497**	5126.498	5126.498
		m	5126.4982	5126.51	**5126.497**	NA	5126.498
		avg	5126.4983	5126.51	**5126.497**	5126.498	5126.498
G05	5126.4967	w	5126.5019	5126.53	**5126.497**	5126.498	5126.498
		sd	7.0E-04	0.0	**7.2e-13**	0.0	**1.3e-12**(7)
		fe	200	100	195600	39459	194.34
		b	-6961.7986	-6944.54	**-6961.81**	**-6961.8139**	**-6961.81**
		m	-6961.7984	-6834.48	**-6961.81**	NA	**-6961.81**
		avg	-6961.7984	-6795.6	**-6961.81**	**-6961.8139**	**-6961.81**
G06	-6961.8138	w	-6961.7983	-6460.53	-6961.81	-6961.8139	**-6961.81**
		sd	3.4E-05	24.6	**1.9e-12**	0.0	2.22-05(3)
		fe	100	100	168800	13577	134
		b	**24.3062**	24.48	**24.306**	24.3294	**24.306**
		m	**24.3062**	25.32	**24.306**	NA	**24.306**
		avg	**24.3062**	25.4	**24.306**	24.4719	**24.306**
G07	24.3062	w	**24.3062**	29.33	**24.306**	24.8352	**24.306**
		sd	**2.0E-06**	0.15	6.3e-05	0.1291	5.4e-08(6)
		fe	200	100	350000	428314	13072.1
		b	**-0.0958**	-0.1	**-0.095825**	**-0.0958**	**-0.0958**
		m	**-0.0958**	-0.1	**-0.095825**	NA	-0.0282
		avg	**-0.0958**	-0.09	**-0.095825**	**-0.0958**	-0.0335
G08	-0.095825	w	**-0.0958**	-0.06	**-0.095825**	**-0.0958**	0.0
		sd	1.5E-11	0.0	**2.7e-17**	0.0	2.1e-02
		fe	200	100	160000	6217	553.6
		b	732.51	847.09	**680.630**	680.6303	**680.6300**
		m	1378.29	3953.97	**680.630**	NA	**680.6300**
		avg	9594.52	4515.67	**680.630**	680.6381	**680.6300**
G09	680.6301	w	218444.86	11623.82	**680.630**	680.6538	**680.6300**

(Continued on next page)

Table 4.10: Best (b), median (m) mean (avg) and worst (w) results and their standard deviation (sd) determined in 30 independent runs with different approaches. Average number of function evaluations (fe).(continued)

Fct.	Optimum		SACOBRA-R	COBRA [34]	ISRES [38]	RGA 10% [6]	COBYLA [31]
		sd	4.0E+04	487.05	**3.2e-13**	0.0066	3.7e-10(2)
		fe	300	100	271200	388453	8973.4
G10	7049.2480	b	**7049.2480**	8238.78	**7049.248**	7049.2607	7050.3
		m	7049.2492	18031.74	**7049.248**	NA	7064.8
		avg	7059.25	17498.57	**7049.25**	7049.5659	8085.5
		w	7250.98	25086.88	**7049.27**	7051.6857	11259.7
		sd	3.8E+01	892.28	**3.2e-03**	0.5699	1.9e+03(22)
		fe	300	100	348800	572629	270840.2
G11	0.75	b	**0.75**	NA	**0.75**	**0.75**	**0.75**
		m	**0.75**	NA	**0.75**	NA	**0.75**
		avg	**0.75**	NA	**0.75**	**0.75**	**0.77**
		w	**0.75**	NA	**0.75**	**0.75**	**1.4**
		sd	1.3E-11	NA	**1.1e-16**	0.0	0.12
		fe	100	NA	137200	7215	11788.7

In order to evaluate the results gained by self-adaptive COBRA-R (SACOBRA-R), we compared our results for the G-functions with the results obtained by means of several other approaches. The improved stochastic ranking evolutionary strategy (ISRES) [38], genetic algorithm combined with a repair mechanism (RGA) [6] and two other surrogate assisted approaches COBYLA [30] and Regis' COBRA [34] are used in this study. The algorithms ISRES and RGA are population based strategies. Although these methods can successfully be performed on many optimization problems, the high number of required function evaluations makes them inappropriate for expensive optimization tasks. Table 4.10 lists statistics on the final solutions found by 5 different optimization means including SACOBRA-R. Also, the averaged number of function calls is listed in the same table as fe.

The solutions found by our approach are close to what is gained with ISRES and RGA for 7 problems out of 11, with the difference that the SACOBRA-R needs a much smaller amount of function evaluations. For instance, the optimum of G01 is found by SACOBRA-R with a similar accuracy as ISRES, but our approach required to call the real functions 3500 times less than ISRES. G04, G05, G06, G07, G08 and G11 are also problems which can be solved by SACOBRA-R with a very limited number of function evaluations and good accuracy. The best solutions found for the problems G03 and G10 are also competing with ISRES and RGA, but our overall results on these two functions are weaker in comparison with ISRES and RGA. This is due to the existence of several unfortunate starting initial designs. We believe that increasing the probability of random initialization of the internal optimizer by choosing larger values for P_{max} and T (Equation 4.8) can help to improve on the worst results for these problems. Although SACOBRA-R cannot obtain as good results as ISRES and RGA in approaching G03 and G10, the median error achieved by RGA, ISRES and SACOBRA-R are similar. Our approach accomplished this by 1000 times less function evaluations than ISRES and RGA.

On the other hand, SACOBRA-R fails to minimize G02 and G09 due to the complexity of their objective functions to be modeled by our surrogate approach. It seems like the evolutionary based algorithms can successfully find the optimum of G02 and G09 within approx. 350000 iterations. Our approach is not the only algorithm which fails to optimize these two functions. COBRA [34] and COBYLA also perform poor in optimizing the G02 problem. The reason for the bad results achieved by the COBRA [34] can be similar to what we faced by applying SACOBRA-R, so that a problem with many local optima cannot be addressed with the current model based approaches. COBYLA is a local optimizer, therefore, we cannot expect too much from such a technique to find the global optimum of a problem like G02 with many local optima. There is no embedded trick defined for COBYLA approach for escaping a local optimum. Although, COBYLA performs many iterations before terminating the search on G02 (in average 97391.28 iterations), the results are not promising. This approach appears to be successful for solving the G09 problem, but 2 solutions out of 30 have a infeasibility level of $> 10^{-8}$. Also, COBYLA needs thousands of real function evaluations to solve G09.

In many cases SACOBRA-R is outperforming COBYLA in terms of averaged required function evaluations. The results from G05, G06 and G07 achieved by COBYLA need a relatively low number of evaluations but still SACOBRA-R is a better approach since all 30 runs can locate a feasible point in the neighborhood of the real optimum. COBYLA fails to find a feasible point for G05, G06 and G07, respectively, 7, 3 and 6 times out of 30.

In general, we can claim that the SACOBRA-R optimization algorithm outperforms other techniques for 7 G-problems out of 11 tested ones because of finding good feasible solutions within very small number of function evaluations. For 2 other problems (G03, G10) SACOBRA-R has intermediate performance because of the early stagnation of several trials. We should also state that SACOBRA-R fails to minimize 2 of the tested G-problems (G02, G09) out of 11.

4.2 MOPTA 2008

The MOPTA 2008 benchmark (MOPTA08) by Jones [19] is a simplified version of a high-dimensional real-world mass optimization problem which arises in the automotive industry: The problem is described by $d = 124$ variables and 68 constraints. MOPTA08 is an expensive problem where, in reality, only 60 points can be computed within 24 hours. It is desirable to find the optimum in less than a month which means the problem should be solved after $30 \times 60 \approx 15 \cdot d$ iterations. We believe, that such a highly constrained large scaled problem depicts the ideal benchmark to assess the capability of the COBRA-R optimization framework.

A promising solver for MOPTA08 should be able to either find a fully feasible solution with an objective value of smaller than 225 after $15 \cdot d$ iterations, or should have a fast convergence to the feasible point with objective value of 228 or less in about $8 \cdot d$ function evaluations [19].

The simulation of MOPTA08 was provided by Jones as a FORTRAN program. One point is given as starting point which is feasible and has the objective value of 251.07. All input parameters are normalized to $[0, 1]$. The constraint values are meaningfully scaled, e. g., a constraint value s_i of 0.05 indicates that the constraint is violated by a percentage of 5%. Unlike many G-problems, MOPTA08 does not require to be rescaled and modified because input space and constraint values are normalized. But, MOPTA08 still remains challenging due to the high dimensional space and the large number of constraints.

Jones suggests Powell's COBYLA [31] as a promising approach to solve MOPTA08 efficiently, although our initial tests with COBYLA showed that it is difficult for COBYLA to resolve all of the constraint violations. Figure 4.17 illustrates the COBYLA optimization process for MOPTA08. We used the COBYLA implementation in `R` [17] from the nloptr package.

As it is shown in Figure 4.17, COBYLA most of the time returns infeasible points. In the first 1000 iterations the objective value improves dramatically from 251 to 228, but after the first 1000 iterations it does not improve anymore. This poor performance may be the result of the underpenalization [37]. The term underpenalization is used for the situations where small penalty factors result in ignoring the influence of constraints. The inaccuracy of the linear approximation could be another possible reason for weak performance of COBYLA.

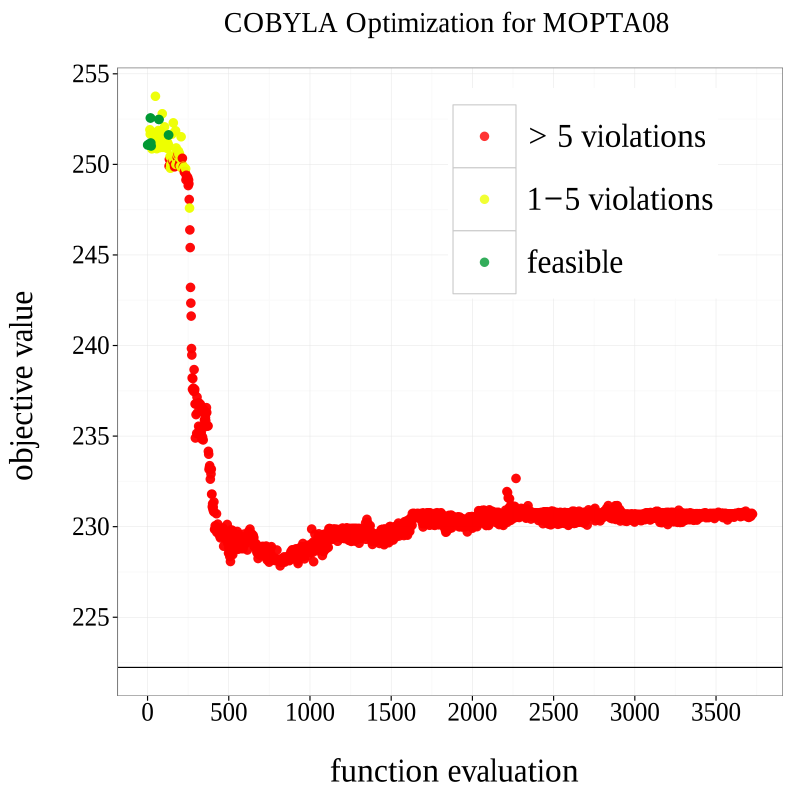


Figure 4.17: COBYLA optimization process for MOPTA08. These results were generated by using the COBYLA implementation from the nloptr package in R. The parameters are all set as default. Red: the solutions with more than 5 violated constraints. Yellow: the solutions with 1 up to 5 violated constraints. Green: no constraint is violated (feasible point).

4.2.1 Performance of COBRA-R with different Internal Optimizers

Since a feasible starting point is provided for MOPTA08, Biased and Optimized initialization described in 2.4.3 are more desirable to be used in comparison with random initialization. Our initial tests showed that Optimized initialization is better in the case of MOPTA08. Therefore, we selected this approach for the further experiments. On the other hand, the achieved results on the G-functions showed that the choice of a relatively large value in the set of the distance requirement cycle can be helpful for the problems which require more exploration. However, this is not necessarily true for all types of problems. We started our first tests with $\Xi = \Xi_1 = \{0.3, 0.01, 0.001, 0.0005\}$.

As it is described in [20] and Section 2.4.3, the COBRA-R framework is fully flexible to adapt or change different parameters and also it is possible to embed new algorithms as internal optimizer or penalty handling techniques. We used different internal optimizers like ISRES [38] from the nloptr package, HJKB [14] and NMKB [27] from the dfoptim package to investigate the performance of the COBRA-R with different internal optimizers for the MOPTA08 problem.

In general, the usage of ISRES as the internal optimizer of COBRA-R for high dimensional problems such as MOPTA08 is not practical, since ISRES is a population based strategy and as a consequence, not computationally efficient. Figure 4.18 shows that ISRES as the internal optimizer totally fails in optimizing MOPTA08. COBRA-R(ISRES) returns either feasible points with large objective or infeasible points. This may be due to the fact that we set the maximum function evaluation of the internal optimizer to 10000 and this is indeed not enough for ISRES to solve a 124 dimensional problem. ISRES needs about 350000 function evaluations for solving most of the G-problems with much smaller dimensions and less number of constraints than MOPTA08. Increasing the maximum number of evaluations for the internal optimizer is not a choice due to the high computational time.

The performance of NMKB, HJKB and COBYLA as internal optimizer in the COBRA-R framework was shown in an earlier study [20]. In Figure 4.19 (left), we can see the best ever feasible point found so far in every iteration. At the first glance, COBRA-R(COBYLA) appears to be the worst with late and slow improvement, while COBRA-R(HJKB) outperforms the other two approaches. But a closer look at the results reveals that COBRA-R(COBYLA) is improving the objective value, although with slightly violated solutions, as illustrated in Figure 4.19 (right). We assume, that points with small infeasibility are somewhere very close to the borders of the feasible region. Therefore, we consider that some levels of infeasibility can be permitted. If we tolerate only a small violation ($< 0.5\%$), then COBRA-R(COBYLA) has the fastest improvement. The left and right curves regarding COBRA-R(HJKB) and COBRA(NMKB) in Figure 4.19 are identical which implies that these two algorithms did not produce any point with 0.5% infeasibility.

HJKB and NMKB are both unconstrained optimization algorithms and as it is described in Section 2.4.3 they handle the constrained subproblem by a static penalty approach. The penalty coefficient changes and gets adapted over the whole optimization process and remains constant during the internal optimization loop. This can be considered as a weak point. We believe, that the observed early stagnation in Figure 4.19 occurred for COBRA-R(HJKB) and COBRA-R(NMKB) is due to over-

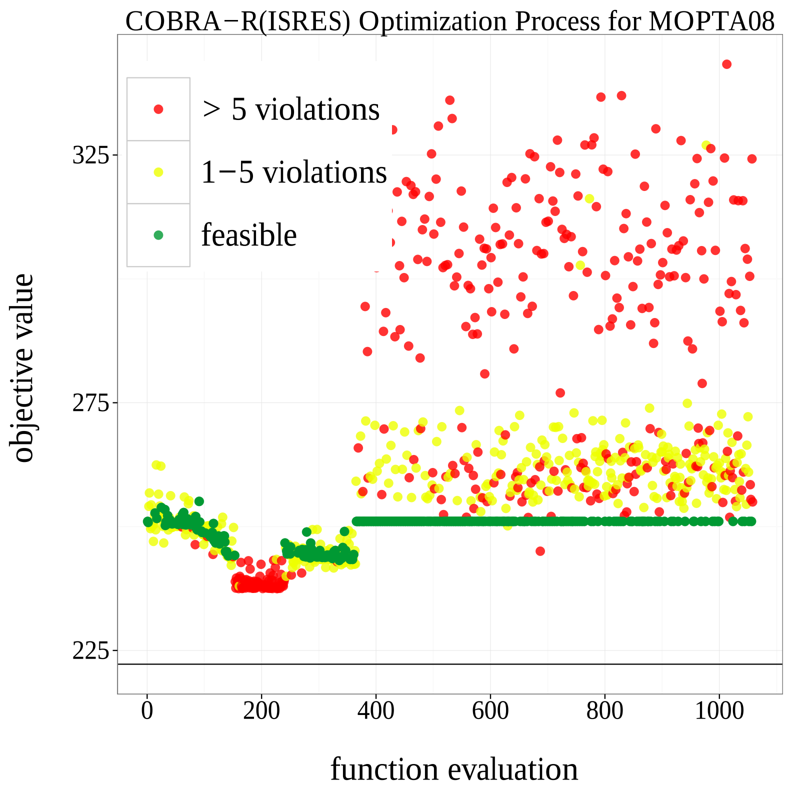


Figure 4.18: COBRA-R optimization process for MOPTA08 with ISRES as the internal optimizer. The process fails to optimize MOPTA08 because ISRES typically needs a large number of function evaluations in order to solve an optimization problem –especially for high dimensional problems.

Table 4.11: Distance Requirement Cycles used in COBRA-R SACOBRA-R optimization frameworks.

Ξ_{name}	DRC
Ξ_l	$\{0.3, 0.05, 0.001, 0.0005, 0.0\}$
Ξ_s	$\{0.001, 0.0\}$
Ξ_1	$\{0.3, 0.01, 0.001, 0.0005\}$
Ξ_2	$\{0.01, 0.001, 0.0005\}$

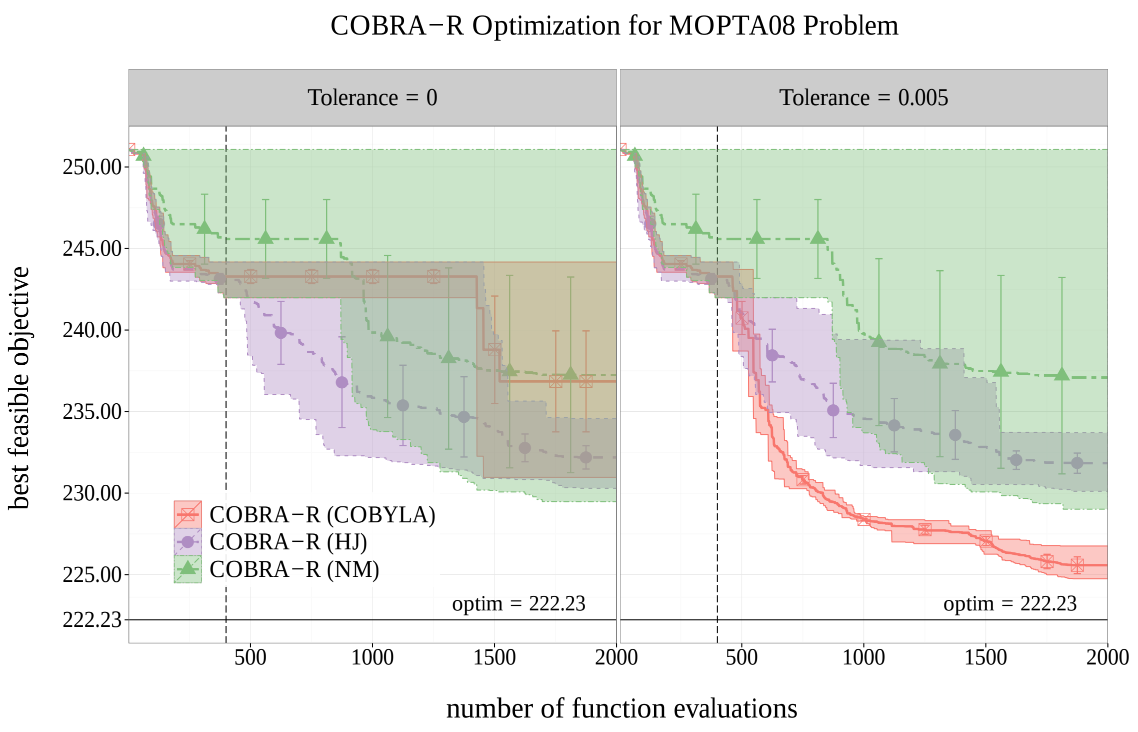


Figure 4.19: Left: objective value of the ever best fully feasible point found in every iteration by COBRA-R with HJKB, NMKB and COBYLA as internal optimizers. Right: the best objective value of the points with infeasibility of $< 0.5\%$ found in every iteration by COBRA-R with HJKB, NMKB and COBYLA as internal optimizers. The best known objective for MOPTA08 is shown with a horizontal line, this point is determined by COBRA [34] after 4000 iterations. The curves regarding COBRA-R(HJKB) and COBRA-R(NMKB) are identical but COBRA-R(COBYLA) drops down after tolerating a small amount of infeasibility.

penalization [37] which happens in a case that the penalty coefficients are too large that only the constraints are taken into account and the impact of the objective value is ignored. Figure 4.19 also indicates that the points returned by COBRA-R(HJKB) and COBRA-R(NMKB) are mostly feasible but they do not improve the objective value too often.

4.2.2 Repairing infeasible for MOPTA08

Based on what we observed in the last section, COBRA-R(COBYLA) outperforms the other tested algorithms if the points with a small infeasibility (up to 0.5%) are accepted or if a repairing algorithm could guide the infeasible points close to the feasible boundaries to the feasible region. Since we are using surrogate models, a repair algorithm only imposes one extra function evaluation for each repair. In [21], a new gradient based repair algorithm is proposed and embedded in the COBRA-R framework. The performance of the proposed repair algorithm "RI-2" was compared with another gradient based repair proposed by Chootinan in [6]. This repair method which we called "CHO", is used originally in combination with a Genetic Algorithm in [6]. But in COBRA-R, the repair algorithms are embedded in a way that they attempt to repair the infeasible points returned by internal optimizer on the surrogate models only if the maximum constraint violation is smaller than 0.1. After a point is returned by the repair algorithm, the real function is called to evaluate the new result.

In Figure 4.20, all of the curves except COBRA-Regis are results of the COBRA-R optimization with COBYLA as internal optimizer. Different curves are indicating results with different types of repairing approaches and different distance requirement cycles (4.11). As it was mentioned before, COBRA [34] uses an interior point algorithm from MATLAB's Fmincon as the internal optimizer, and also there is no repair approach embedded in the COBRA framework from Regis [34]. All runs with COBRA-R(COBYLA) generate the initial population using Optimized initialization approach described in Section 2.4.3. But Regis produces the initial population in a biased manner [34], e. g., apart from the starting point, d other individuals are selected on each positive coordinates with 0.005 distance from the given starting point.

In an earlier study [21], it is shown that COBRA-R combined with the "RI-2" repair algorithm outperforms COBRA-R with CHO repair algorithm in order to solve MOPTA08. In the following we present the results of COBRA-R(COBYLA) with RI2 repair algorithm and a different distance requirement cycle Ξ_2. The newly tested distance requirement cycle is identical to what is called local distance requirement

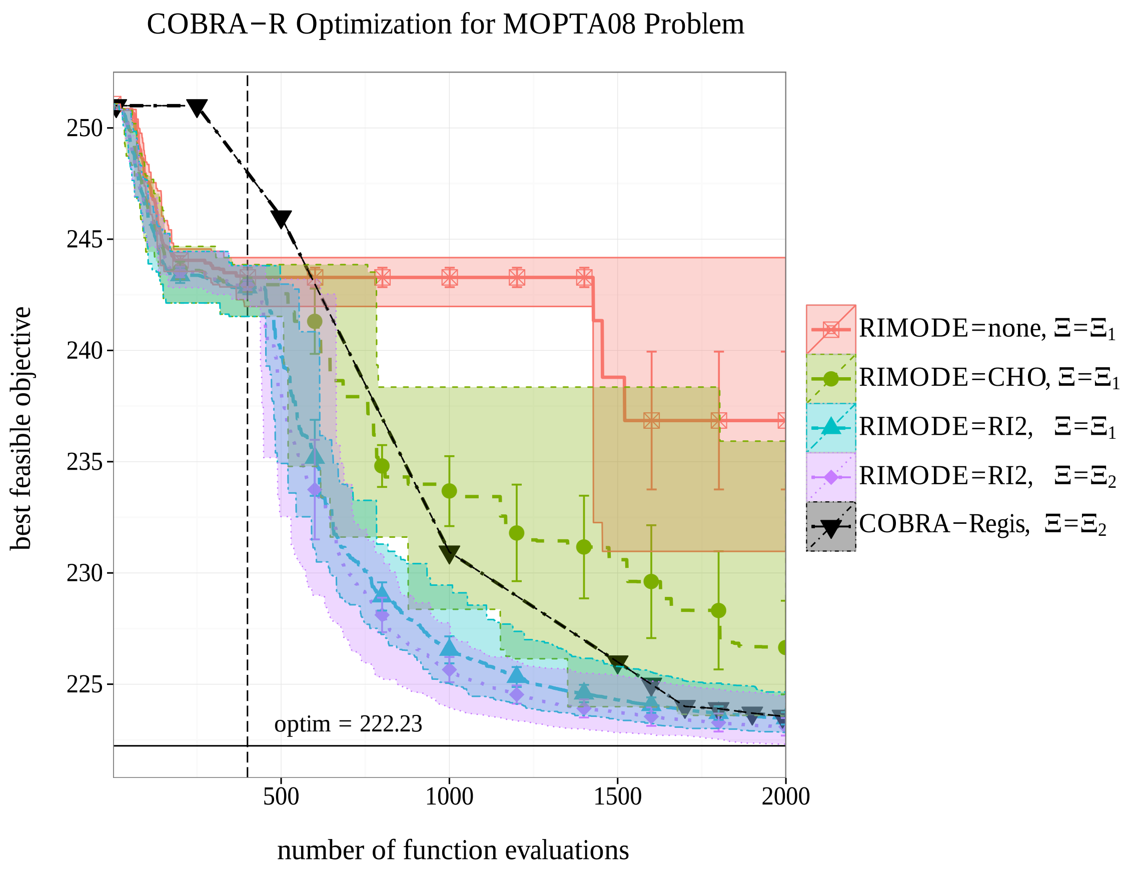


Figure 4.20: Different optimization processes for MOPTA08. All curves except the black one are the result of COBRA-R(COBYLA) optimization and are average over 10 independent runs. The bounds around the curves are representing the best and worst values among the 10 runs. (RIMODE=none): no repair algorithm is used during the optimization process. (RIMODE=RI2): the repair algorithm proposed in [21] is utilized. (RIMODE=CHO): the repair algorithm proposed in [6] is applied during the COBRA-R(COBYLA) optimization process. Ξ_1 and Ξ_2 are different sets of distance requirements (DRC) from Table 4.11. The points of the black curve are results achieved by COBRA-LOCAL in [34]. The distance requirement cycle in COBRA-LOCAL is Ξ_2 with no repair algorithm. The initialization phase is separated by a dashed vertical line. After the initialization phase is completed, the COBRA-R(COBYLA)[RI-2,Ξ_2] algorithm – indicated by the magenta curve – outperforms all the other algorithms during the optimization process. Figure 4.21 represents a zoomed version of this figure for the last iterations

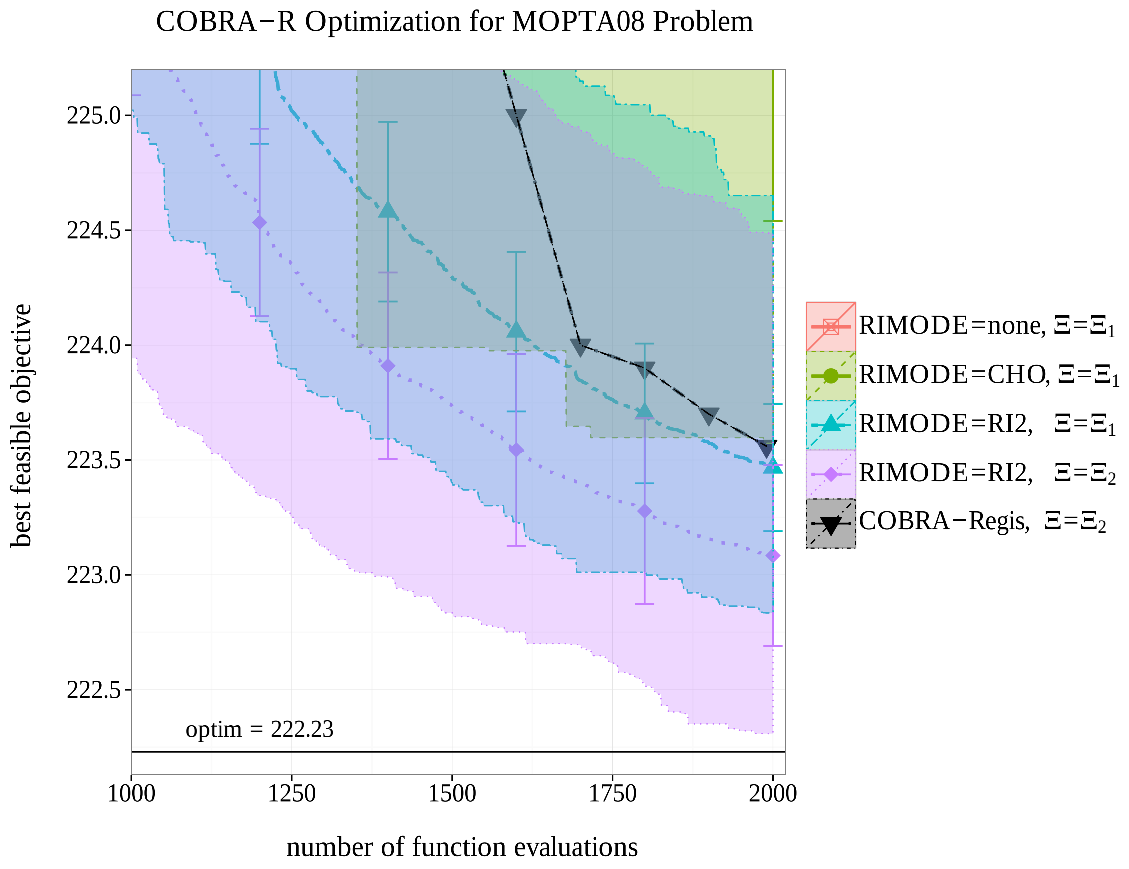


Figure 4.21: Different optimization processes for MOPTA08, zoomed into the last iterations to have a clear view of the final values in 2000-th iteration. The average and the best solution found by COBRA-R(COBYLA) – with RI-2 repair algorithm and Ξ_2 – has a smaller objective value than all other shown approaches.

cycle by Regis [34]. The fact that larger moves by the CHO repair algorithm yields in the worst results made us come up with this idea that maybe for MOPTA08 smaller moves are preferable due to the high number of constraints. We believe that the benefits of COBRA-R in comparison with what Regis reports in [34, 33] as the best results for MOPTA08 is not due to the different selection of DRC but because of the other components. Therefore, a new set of distance requirements Ξ_2 was tested which was similar to Ξ_1 only the largest value in the cycle (0.3) is omitted.

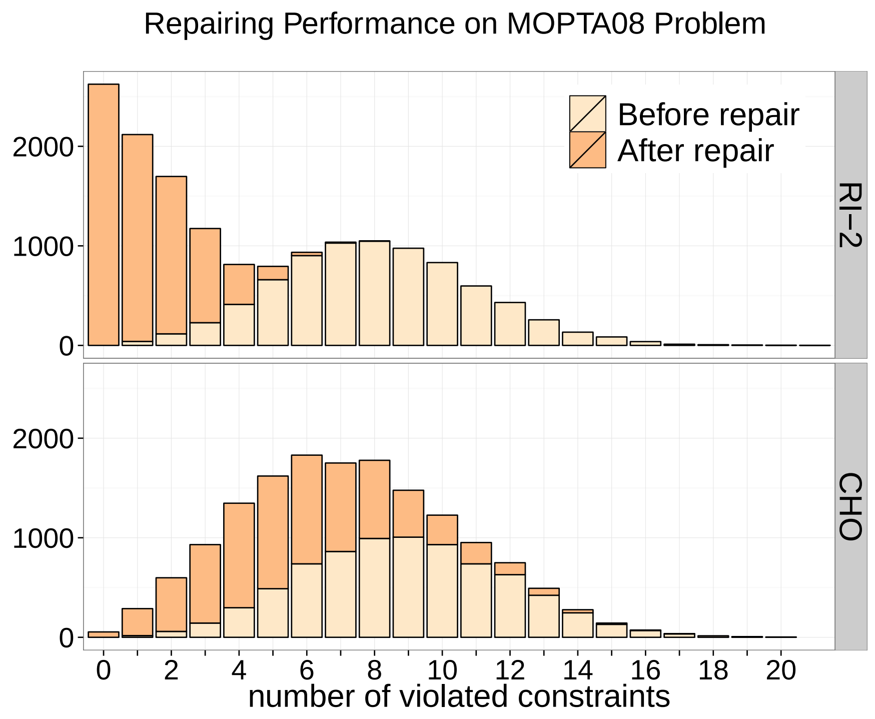


Figure 4.22: Repairing performance in reducing the number of the violated constraints for MOPTA08 problem. Before performing any repair mechanism, often many constraints are violated (In average 8-9 constraints). This figure clearly shows that the RI-2 mechanism can reduce the number of the violated constraints to zero in many cases or reduce them to smaller numbers while the Chootinan repair mechanism can rarely eliminate all of the constraints violations. Even after applying the Chootinan repair the average number of the violated constraints remains relatively high (5-6 constraints are violated most of the time after performing the Chootinan repair).

COBRA-R(COBYLA) with RI-2 and a distance requirement cycle of Ξ_2 is shown as the magenta curve in Figure 4.20. It is apparent that after the initialization phase, the results represented by the magenta curve are better than the others and also better than COBRA-Regis. Therefore, COBRA-R with the proposed RI-2 approach is clearly outperforming COBRA-Regis for the same set of distance requirement cycles. In the early iterations the difference is very significant. In order to have a better view on the final values we replotted the Figure 4.20 with a zoom on the last 1000 iterations (see Figure 4.21). The best and the averaged result achieved by COBRA-R(COBYLA) with RI-2,Ξ_2 after 2000 iterations is clearly better than the result from COBRA-Regis.

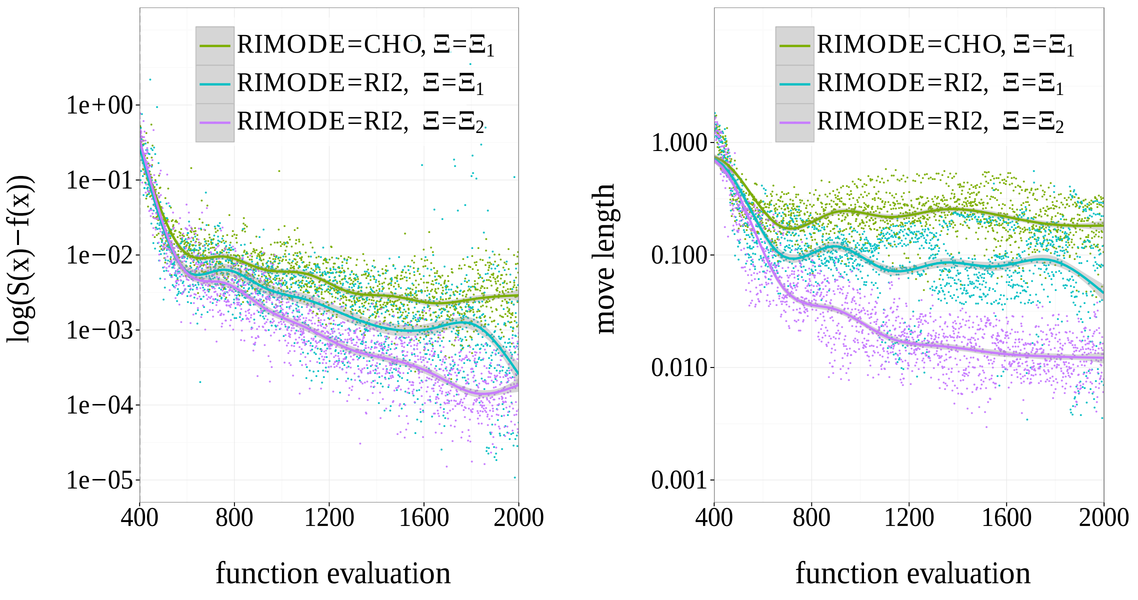


Figure 4.23: Left: RBF approximation error of fitness function per iteration. Assume, $S_0^{(k)}$ is the RBF approximation of the fitness in the kth iterate, on the y-axis the absolute difference of the true fitness function value for the $\vec{x}_k$ and the predicted value with the kth surrogate is shown in a logarithmic scale, $log(|S_0^{(k)}(\vec{x}_k) - f(\vec{x}_k)|)$. Right: Move length in the input space per iteration. Consider $\vec{x}_k$ is the new iterate in kth iteration, on the y-axis the Euclidean distance of $\vec{x}_k$ from $\vec{x}_{k-1}$ in the logarithmic scale is shown $log(||\vec{x}_k - \vec{x}_{k-1}||)$.

Figure 4.22 clearly shows that the RI-2 mechanism can reduce the number of the violated constraints to zero in many cases or reduce them to smaller numbers while the Chootinan repair mechanism can rarely eliminate all of the constraints violations. Even after applying the Chootinan repair the average number of the violated constraints remains relatively high (5-6 constraints are violated most of the time after performing the Chootinan repair).

The main difference between Chootinan and Chen's repair method and RI-2 is the different update of the search point. While RI-2 tries to explore new points in the feasible parallelepiped, Chootinan and Chen's update procedure can result in larger steps in the search space. Although this may lead to quick fixes of constraint violations, it can also induce possible new violations of constraint functions [21]. This is the case for the MOPTA08 problem with 68 constraints. As it is shown in Figure 4.22, CHO is not successful in reducing the number of violated constraints for MOPTA08. On the other hand, in the last test with smaller Ξ_2 we show that the overall performance is enhanced. Figure 4.23 (right) illustrates the length of the moves in the input space in every iteration for three approaches. By length of the move, we mean distance of every new infill point from the one before $||\vec{x}_k - \vec{x}_{k-1}||$. Comparing the blue and green curves in Figure 4.23 (right) clarifies our claim that the Chootinan repair makes longer moves. Also, it is apparent that the average move length during the optimization procedure with Ξ_2 is smaller than a COBRA-R procedure with Ξ_1. It is interesting to see how the approximation error in Figure 4.23 (left) is proportional to the length of the moves in the input space.

Chapter 5

Conclusion and Future work

5.1 Conclusion

Solving black-box constrained optimization problems under severely limited budgets is a demanding task for the existing minimization techniques. Although several surrogate assisted approaches have been proposed in the last few years with the aim of reducing the real function evaluations, the progress in this direction is not significant. One of the most promising algorithms for addressing such demands is proposed by Regis in [34] which utilizes the RBF interpolation for modeling both objective and constraint functions and attempts to solve constrained subproblem on the surrogate models. However, COBRA can accomplish promising results for the MOPTA08 problem with 124 dimensions and 68 constraints, the minimization results reported for the G-problems are not competing with what is achieved by other approaches like variants of evolutionary algorithms.

In this study, we investigated the performance of the COBRA-R [20] optimization framework which is a re-implementation of Regis' COBRA [34] in `R` with several contrasts listed below:

- The COBRA-R framework is implemented in `R` unlike COBRA which is programmed in `MATLAB`.
- The *internal optimizer* can be selected by the user among COBYLA, HJKB, NMKB and ISRES instead of using `Fmincon` from `MATLAB`.
- The *initialization phase* can be done by one of the following approaches: LHS, Biased, Optimized while in Regis' COBRA [34] the initialization is done either randomly by the LHS approach or in a biased manner.
- Regis uses $d + 1$ individuals for the initial population, whereas we assign a larger *size for the initial design* ($\geq 3 \cdot d$).

- Embedding the *repair infeasible algorithm* [21] to move the solutions with a small infeasibility found by the internal optimizer into the feasible region.

G-problems with widely varying features impose various challenges to the COBRA-R algorithm. In order to answer the first research question [**Q.1**] we can refer to Table 4.2 which listed a summary of the challenges imposed by the test functions G01 to G11. In general, the demands to solve the G-problems can be categorized in to three classes: 1. Complex or tough objective or constraint functions to be modeled with a cubic RBF interpolation approach, 2. Numerical issues appeared in training a cubic RBF model for simple functions but with large values in the input space, 3. Difficulty to select a suitable distance requirement cycle for different problems with various sizes of search space and feasible space.

For the first experiments handling the different challenges is done by manual parameter tuning and modification of tough objective or constraint functions of different G-problems. After assigning the suitable parameter setting listed in Table 4.1, based on the gained results shown in Table 4.4 we can answer [**Q.2**] positively for 10 out of 11 tested G-problems. Some problems only need very few function evaluations after the initialization to reach the optimum with an error smaller than 10^{-6}. COBRA-R with tuned parameters needs less than 60 iterations to solve G01 and G11. In general, except from G02 which is a tough problem to be solved with surrogate assisted techniques due to many local optima in 20 dimensions, all other tested G-problems could be solved with less than 500 function evaluations. Suitable variants of evolutionary algorithms to address constrained problems can solve the G-problems within thousands of function evaluations; therefore, the reduction of real function evaluations by COBRA-R is significant whereas, the final error is also reasonably small.

The positive answer to [**Q.3**] can be directly illustrated in Figure 4.3, 4.4 and 4.5. Comparing the median optimization error obtained by COBRA and COBRA-R within 100 iterations shows that our approach is performing better for all the tested G-problems. Only for the G03MOD problem in Figure 4.6, it is shown that both algorithms cannot achieve good results with 100 function evaluations. But we also show that COBRA-R is capable of getting close to the optimum after 400 iterations with the median error of $< 10^{-5}$. The main reason for the improvement of COBRA-R is due to the use of a manually tuned parameter setting for each problem unlike COBRA [34] which uses one parameter setting for handling all G-problems. The distance requirement cycle suggested by Regis appears to have elements that are too large for many of the G-problems with a tiny feasible region ρ or large fitness range FR. On the other hand, some problems need more exploration, so larger elements in the DRC are desirable. The improved results achieved for G10 were due to different approaches used to modify the steep constraints. Regis in [34] utilizes a logarithmic transformation to modify steep functions. But we used a linear transformation to map all constraint values to the range of -1 to 1. When all the constraints are varying in a similar range then none of the constraints are underpenalized or overpenalized. The manually tuned parameters for G-problems are listed in Table 4.5.

In order to improve the overall performance of COBRA-R, we introduced three new extensions to be combined with the COBRA-R framework:

a: Input space rescaling.

b: Random start algorithm (Algorithm 1).

c: Parameter/function(s) adaptation step (Algorithm 2).

We called the extended COBRA-R framework with the three listed extensions as "self-adaptive COBRA-R" or "SACOBRA-R".

Before the initialization phase of SACOBRA-R, it is decided whether according to the bounds of the problem the input space should be rescaled to $[0, 1]^d$.

The random start algorithm is a step before the optimization on the surrogates (see Figure 4.9). In this step the starting point which will be passed to the internal optimizer is selected. Most of the time the starting point is the best solution found so far during the optimization process. But with a low probability a random point in the search space is passed to the internal optimizer as the starting point of the search on the surrogates. We investigated whether this algorithm can improve the worst-case results and assist the search to escape from a possible local optimum. The G01 problem provides a test-bed where the functionality of the proposed approach can be evaluated (see Figure 5.1). Overall, the worst case error is improved for 6 problems after applying the random start algorithm. Therefore, we can answer [**Q.4**] positively.

The parameter/function(s) adaptation algorithm is proposed with the aim of reducing the sensitivity of the COBRA-R optimization to parameter selection and function modification. This algorithm is performed after the initialization phase and utilizes the information gained from $3 \cdot d$ individuals located randomly in the search space in the initialization phase. This algorithm includes three main parts:

1. Transformation of the objective function if FR is extremely large.

2. Normalization of the constraints if the constraint ratio GR is large.

3. If the objective function range FR is large, replacement of the default distance requirement cycle Ξ_l with a DRC including only two small elements Ξ_s.

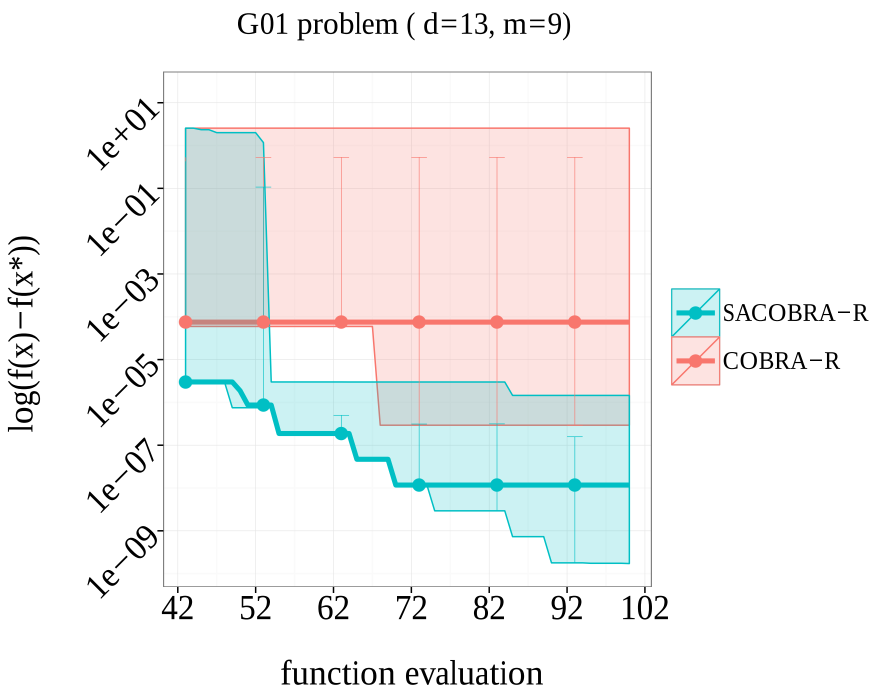


Figure 5.1: Impact of random start algorithm on G01

The results shown in Table 4.7 are achieved by self-adaptive COBRA-R initialized with one configuration for all G-problems. So, the answer of [**Q.5**] question according to these results is positive for 9 cases out of 11. All G-problems can be approached reasonably well with self-adaptive COBRA-R except G02 and G09 which are tough problems in terms of modeling their objective function with RBF interpolation trained on very limited amount of individuals. It is worthwhile to mention that although self-adaptive COBRA-R cannot always achieve as good results as manually tuned COBRA-R, it eliminates the need for adjusting functions and tuning the DRC parameter. For black-box optimization problems a parameter-free optimization algorithm is desirable.

Additionally, a high dimensional highly constrained benchmark (MOPTA08) was used to evaluate COBRA-R. We answer the question posed in [**Q.6**] positively by referring to Figure 4.20 and 4.21. It is shown that COBRA-R found a better feasible solution for MOPTA08 in comparison with Regis's COBRA [34]. The improvement can be due to using the combination of the optimized initialization, COBYLA as the internal optimizer and the RI-2 repair mechanism.

5.2 Future Work

Although we have accomplished a reduction of the sensitivity of the COBRA-R algorithm to the parameter selection, there is still considerable scope for improvements. At present, the best internal optimizer embedded in our optimization framework is COBYLA, but we have shown that COBYLA is not the best choice for the internal optimizer when the problems have many local optima. ISRES could be a practical alternative for problems in low dimensions. Further investigation on a more generic solver for the internal optimizer can be considered for future work. According to the No Free Lunch Theorem for optimization [44], a solver can be appropriate for a group of problems but not effective on many others. Therefore, looking for an intelligent way of selecting the internal optimizer, is another interesting field to be investigated in future work.

We are planning to evaluate our optimization framework with more high dimensional, real-world benchmarks, for example the engineering examples evaluated in [34]. We believe that evaluating our framework with more real-world problems brings a better insight into the existing weaknesses of our algorithm, so we can have the opportunity to do further improvements. It is also interesting to test the ability of the self-adaptive COBRA-R on the same problems but in a noisy environment. However, we believe this can be a challenge for the current version of the algorithm.

Investigations in future could concentrate on the improvement of the surrogate approach in different means. On the one hand, we observed that in several cases the failure of the optimization process was due to the weak model. On the other hand, to our knowledge, RBF models are one of the best approaches in terms of computational time, even in high dimensions. Hence, investigation on the performance of different kinds of radial basis functions in our framework is one of the main tasks we will follow.

Bibliography

[1] Arnold, D., Hansen, N.: A (1+1)-CMA-ES for Constrained Optimisation. In: Soule, T., Moore, J.H. (eds.) Proceedings of the 14th International Conference on Genetic and Evolutionary Computation. pp. 297–304. ACM (2012)

[2] Bäck, T.: Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford University Press, Oxford, UK (1996)

[3] Bäck, T., Hoffmeister, F., Schwefel, H.: A survey of evolution strategies. In: Belew, R.K., Booker, L.B. (eds.) Proceedings of the 4th International Conference on Genetic Algorithms, San Diego, CA, USA. pp. 2–9. Morgan Kaufmann (1991)

[4] Basudhar, A., Dribusch, C., Lacaze, S., Missoum, S.: Constrained Efficient Global Optimization with Support Vector Machines. Structural and Multidisciplinary Optimization 46(2), 201–221 (2012)

[5] Buhmann, M.D.: Radial Basis Functions. Cambridge University Press, New York, NY, USA (2003)

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

[7] Coello, C.A.C.: Constraint-handling using an evolutionary multiobjective optimization technique. Civil Engineering and Environmental Systems 17, 319–346 (2000)

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

[9] Deb, K.: An efficient constraint handling method for genetic algorithms. Computer Methods in Applied Mechanics and Engineering 186(24), 311–338 (2000)

[10] Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edn. (1989)

[11] Ho, Y.C., Pepyne, D.: Simple explanation of the no free lunch theorem of optimization. In: Zhu, I.J., et al. (eds.) Proceedings of the 40th IEEE Conference on Decision and Control, 2001. vol. 5, pp. 4409–4414. IEEE, Piscataway, NJ (2001)

[12] Hoffmeister, F., Bäck, T.: Genetic algorithms and evolution strategies: Similarities and differences. In: Schwefel, H.P., Männer, R. (eds.) Parallel Problem Solving from Nature, Lecture Notes in Computer Science, vol. 496, pp. 455–469. Springer Berlin Heidelberg (1991)

[13] Holmström, K., Quttineh, N.H., Edvall, M.: An adaptive radial basis algorithm (arbf) for expensive black-box mixed-integer constrained global optimization. Optimization and Engineering 9(4), 311–339 (2008)

[14] Hooke, R., Jeeves, T.A.: Direct search solution of numerical and statistical problems. J. ACM 8(2), 212–229 (1961)

[15] Jakobsson, S., Patriksson, M., Rudholm, J., Wojciechowski, A.: A method for simulation based optimization using radial basis functions. Optimization and Engineering 11(4), 501–532 (2010)

[16] Jiao, L., Li, L., Shang, R., Liu, F., Stolkin, R.: A novel selection evolutionary strategy for constrained optimization. Information Sciences 239(0), 122–141 (2013)

[17] Johnson, S.G.: The NLopt nonlinear-optimization package, [Online. Last Accessed 28.02.2015]

[18] Jones, D., Schonlau, M., Welch, W.: Efficient global optimization of expensive black-box functions. Journal of Global Optimization 13(4), 455–492 (1998)

[19] Jones, D.: Large-Scale Multi-Disciplinary Mass Optimization in the Auto Industry. Modeling and Optimization: Theory and Applications Conference (MOPTA) (2008)

[20] Koch, P., Bagheri, S., Foussette, C., Krause, P., Bäck, T., Konen, W.: Constrained optimization with a limited number of function evaluations. In: Hoffmann, F., Hllermeier, E. (eds.) Proceedings 24. Workshop Computational Intelligence. pp. 119–134. Universitätsverlag Karlsruhe (2014), young Author Award GMA-CI

[21] Koch, P., Bagheri, S., Konen, W., Foussette, C., Krause, P., Bäck, T.: A new repair method for constrained optimization. In: Jimnez-Laredo, J.L. (ed.) Proceedings of the 17th Genetic and Evolutionary Computation Conference. GECCO'15, vol. (submitted) (2015)

[22] Kramer, O., Schwefel, H.P.: On Three New Approaches To Handle Constraints Within Evolution Strategies. Natural Computing 5(4), 363–385 (2006)

[23] Kramer, O.: Self-Adaptive Heuristics for Evolutionary Computation, Studies in Computational Intelligence, vol. 147. Springer Berlin Heidelberg (2008)

[24] Luersen, M.A., Le Riche, R.: Globalized nelder-mead method for engineering optimization. In: Topping, B.H.V., Bittnar, Z. (eds.) Proceedings of the Third International Conference on Engineering Computational Technology. pp. 165–166. ICECT'03, Civil-Comp press, Edinburgh, UK (2002)

[25] Michalewicz, Z., Nazhiyath, G.: Genocop III: a co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. In: IEEE International Conference on Evolutionary Computation. vol. 2, pp. 647–651 vol.2. IEEE., Piscataway, NJ (1995)

[26] Michalewicz, Z., Schoenauer, M.: Evolutionary Algorithms for Constrained Parameter Optimization Problems. Evolutionary Computation 4(1), 1–32 (1996)

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

[28] Pham, H.: Reduction of function evaluation in differential evolution using nearest neighbor comparison. Vietnam Journal of Computer Science pp. 1–11 (2014)

[29] Poloczek, J., Kramer, O.: Local SVM Constraint Surrogate Models for Self-adaptive Evolution Strategies. In: Timm, I.J., Thimm, M. (eds.) KI 2013: Advances in Artificial Intelligence. Lecture Notes in Computer Science, vol. 8077, pp. 164–175. Springer Berlin Heidelberg (2013)

[30] Powell, M.J.D.: The Theory of Radial Basis Function Approximation in 1990, pp. 105–210. Oxford University Press, USA (1992)

[31] Powell, M.: A direct search optimization method that models the objective and constraint functions by linear interpolation. In: Gomez, S., Hennart, J.P. (eds.) Optimization And Numerical Analysis, pp. 51–67. Kluweer Academic, Dordrecht (1994)

[32] Queipo, N.V., Haftka, R.T., Shyy, W., Goel, T., Vaidyanathan, R., Tucher, P.K.: Surrogate-based analysis and optimization. Progress in Aerospace Sciences 41, 1–28 (2005)

[33] Regis, R.G.: Stochastic radial basis function algorithms for large-scale optimization involving expensive black-box objective and constraint functions. Computers & OR 38(5), 837–853 (2011)

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

[35] Regis, R.G., Shoemaker, C.A.: Parallel radial basis function methods for the global optimization of expensive functions. European Journal of Operational Research 182(2), 514–535 (2007)

[36] Regis, R.G., Shoemaker, C.A.: A quasi-multistart framework for global optimization of expensive functions using response surface models. J. Global Optimization 56(4), 1719–1753 (2013)

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

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

[39] Schwefel, H.P.P.: Evolution and Optimum Seeking: The Sixth Generation. John Wiley & Sons, Inc., New York, USA (1993)

[40] Singh, H., Ray, T., Smith, W.: Surrogate assisted simulated annealing (sasa) for constrained multi-objective optimization. In: Evolutionary Computation (CEC), 2010 IEEE Congress on. pp. 1–8 (July 2010)

[41] Stein, M.: Large sample properties of simulations using latin hypercube sampling. Technometrics 29(2), 143–151 (1987)

[42] Takahama, T., Sakai, S.: Constrained optimization by applying the α constrained method to the nonlinear simplex method with mutations. IEEE Transactions on Evolutionary Computation 9(5), 437–451 (2005)

[43] Wang, G.G., Dong, Z., Aitchison, P.: Adaptive response surface method – a global optimization scheme for computation-intensive design problems. Journal of Optimization and Engineering 33, 707–734 (2001)

[44] Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1(1), 67–82 (1997)

Appendix A

Algorithms

Algorithm 3 Optimization algorithm NELDER-MEAD. Input: initial simplex represented by $\vec{P}_1, \vec{P}_2, \ldots, \vec{P}_{n+1}$ for an n-dimensional problem, termination criteria. Output: the best point with the lowest objective value, found by the algorithm.

```
1: $f(\vec{P})$ : objective function to be minimized.
2: $\alpha$ : Reflection coefficient.
3: $\gamma$ : Expansion coefficient, which is greater than 1.
4: $\beta$ : Contraction coefficient, which from 0 to 1.
5: $\vec{P}_w$ : Worst point in the current population.
6: $\vec{P}_b$ : Worst point in the current population.
7: $\bar{P}$ : Centroid of the all current points except the worst one.

8: while Termination criteria is not satisfied do
9:     Reflection: $\vec{P}_r = (1+\alpha)\bar{P} - \vec{P}_w$
10:    if $f(\vec{P}_r) < f(\vec{P}_w)$ then
11:        replace $\vec{P}_w$ with $\vec{P}_r$
12:        if $f(\vec{P}_r) < f(\vec{P}_b)$ then
13:            EXPANSION
14:        end if
15:    else
16:        CONTRACTIONNSHRINK
17:    end if
18: end while
```

```
1: function EXPANSION
2:     $\vec{P}_e = \gamma\vec{P}_r + (1-\gamma)\bar{P}$
3:     if $f(\vec{P}_e) < f(\vec{P}_b)$ then
4:         replace $\vec{P}_w$ with $\vec{P}_e$
5:     else
6:         replace $\vec{P}_w$ with $\vec{P}_r$
7:     end if
8: end function
```

```
9: function CONTRACTIONNSHRINK
10:    $\vec{P}_c = \beta\vec{P}_w + (1-\beta)\bar{P}$
11:    if $f(\vec{P}_w) < f(\vec{P}_c)$ then
12:        Shrink the simplex toward $P_c$
13:    else
14:        replace $\vec{P}_w$ with $\vec{P}_c$
15:    end if
16: end function
```

Algorithm 4 Optimization algorithm Hooke-Jeeves.
Input: $\vec{x} = \{x_1, x_2, ..., x_d\} \in \mathbb{R}^d$, termination criteria. Output: the best point found by the algorithm.

```
1: $f(\vec{x})$ : objective function to be minimized.
2: $\epsilon$ : Step size in exploratory moves.
3: $\alpha$ : Acceleration in pattern move.
4: $iter := 0$ : Number of real function evaluations.
5: $\vec{x}^{best} := \vec{x}$ : Assign initial point as the best known point so far.
```

```
1:  function Main($\vec{x}^{best}$)
2:      while Termination criteria are not met do
3:          $\vec{x}^{new} = $ Explore($\vec{x}^{best}$)
4:          if $f(\vec{x}^{new}) < f(\vec{x}^{best})$ then
5:              $\vec{x}^{best} \leftarrow$ Exploit($\vec{x}^{best}$, $\vec{x}^{new}$)
6:          else
7:              $\epsilon \leftarrow \epsilon/2$
8:          end if
9:      end while
10:     return $\vec{x}^{best}$
11: end function

12: function Explore($\vec{x}^{best}$)
13:     for $i = 1, \ldots, d$ do
14:         $\vec{x}^{temp} \leftarrow \vec{x}_i^{best} + \epsilon$
15:         if $f(\vec{x}^{temp}) < f(\vec{x}^{new})$ then
16:             $iter \leftarrow iter + 1$
17:             $\vec{x}^{new} \leftarrow \vec{x}^{temp}$
18:         else
19:             $\vec{x}^{temp} \leftarrow \vec{x}_i^{best} - \epsilon$
20:             if $f(\vec{x}^{temp}) < f(\vec{x}^{new})$ then
21:                 $iter \leftarrow iter + 1$
22:                 $\vec{x}^{new} \leftarrow \vec{x}^{temp}$
23:             end if
24:         end if
25:     end for
26:     return $\vec{x}^{new}$
27: end function

28: function Exploit($\vec{x}^{best}$, $\vec{x}^{new}$)
29:     $\vec{x}^{pMove} = \vec{x}^{new} + \alpha(\vec{x}^{new} - \vec{x}^{best})$
30:     if $f(\vec{x}^{pMove}) < \vec{x}^{new}$ then
31:         $iter \leftarrow iter + 1$
32:         $\vec{x}^{best} \leftarrow \vec{x}^{pMove}$
33:     else
34:         $\vec{x}^{best} \leftarrow \vec{x}^{new}$
35:     end if
36:     return $\vec{x}^{best}$
37: end function
```

Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbständig verfasst zu haben.
Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht.
Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

(Ort, Datum) Samineh Bagheri