A New Repair Method For Constrained Optimization

Patrick Koch
Cologne University of Applied Sciences,
51643 Gummersbach, Germany
patrick.koch@fh-koeln.de

Samineh Bagheri
Cologne University of Applied Sciences,
51643 Gummersbach, Germany
samineh.bagheri@fh-koeln.de

Wolfgang Konen
Cologne University of Applied Sciences,
51643 Gummersbach, Germany
wolfgang.konen@fh-koeln.de

Christophe Foussette
divis intelligent solutions GmbH,
44277 Dortmund, Germany
foussette@divis-gmbh.de

Peter Krause
divis intelligent solutions GmbH,
44277 Dortmund, Germany
krause@divis-gmbh.de

Thomas Bäck
Leiden University, LIACS
2333 CA Leiden, The Netherlands
T.H.W.Baeck@liacs.leidenuniv.nl

ABSTRACT

Nowadays, constraints play an important role in industry, because most industrial optimization tasks underly several restrictions. Finding good solutions for a particular problem with respect to all constraint functions can be expensive, especially when the dimensionality of the search space is large and many constraint functions are involved. Unfortunately function evaluations in industrial optimization are heavily limited, because often expensive simulations must be conducted. For such high-dimensional optimization tasks, the constraint optimization algorithm COBRA was proposed, making use of surrogate modeling for both the objective and the constraint functions.

In this paper we present a new mechanism for COBRA to repair infill solutions with slightly violated constraints. The repair mechanism is based on gradient descent on surrogates of the constraint functions and aims at finding nearby feasible solutions. We test the repair mechanism on a real-world problem from the automotive industry and on other synthetic test cases. It is shown in this paper that with the integration of the repair method, the percentage of infeasible solutions is significantly reduced, leading to faster convergence and better final results.

Categories and Subject Descriptors

G.1.6 [**Optimization**]: Constrained optimization

Keywords

Constrained optimization, Radial Basis Functions, constraint repair

GECCO '15, July 11 - 15, 2015, Madrid, Spain
© 2015 ACM. ISBN 978-1-4503-3472-3/15/07…$15.00
DOI: http://dx.doi.org/10.1145/2739480.2754658

1. INTRODUCTION

Real-world optimization problems are often subject to constraints, restricting the feasible region to a smaller subset of the search space. It is the goal of constraint optimizers to avoid infeasible solutions and to stay in the feasible region, in order to converge to the optimum. However, the search in constraint black-box optimization can be difficult, since knowledge about the size of the feasible region and the fitness landscape is usually unknown. This problem even turns out to be harder, when only a limited number of function evaluations is allowed for the search. However, in industry good solutions are requested in very restricted timeframes. An example is the well-known benchmark MOPTA 2008 [1].

In the past different strategies have been proposed to handle constraints. E. g., repair methods try to guide infeasible solutions into the feasible area. Penalty functions give a negative bias to the objective function value, when constraints are violated. Many constraint handling methods are available in the scientific literature, but often demand for a large number of function evaluations (e. g., results in [2, 3]). Up to now, only little work has been devoted to *efficient* constraint optimization (severely reduced number of function evaluations). A possible solution in that regard is to use *surrogate models* for the objective and constraint functions respectively. While the real function might be expensive to evaluate, evaluations on the surrogate functions are often much faster. As an example for this approach, the solver Constrained Optimization by Radial Basis Function Approximation (*COBRA*) was proposed by Regis [4] and outperforms many other algorithms on a large number of benchmark functions.

In this paper we analyze the COBRA algorithm and reveal its strengths and weaknesses. We extend the algorithm by a specific new repair method. Therefore we have defined the following research questions:

(H1) Can a repair mechanism for infeasible solutions help to achieve a faster convergence and smaller standard deviations as compared to the original COBRA algorithm proposed by Regis [4] without repair?

(H2) Does the proposed repair mechanism perform better than other state-of-the-art repair mechanisms like the one proposed by Chootinan and Chen [5]?

(H3) How often is the repair successful?

1.1 Related work

Following the surveys about constraint handling given by Michalewicz and Schoenauer [6], Eiben and Smith [7], Coello Coello [8] and Kramer [9] several approaches are available to handle constraints:

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

ii) repair algorithms to resolve constraint violations during the search

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

iv) stochastic ranking, where solutions are lexically ranked according to their performance in constraint or fitness values in a comparison with other offspring in a certain number of sweeps.

A frequently used approach to handle constraints is to incorporate static or dynamic penalty terms in order to stay in the feasible region [8, 9, 10]. Penalty functions can be very helpful for solving constrained problems, but their main drawback is that they often require additional parameters for balancing the fitness and penalty terms. Repair algorithms have been mainly proposed in the context of combinatorial optimization problems (see e. g., Xiao et al. [11], Mühlenbein [12], Orvosh and Davis [13], Tate and Smith [14]), where special-purpose repair heuristics are designed anew for each problem. For continuous problems Chootinan and Chen [5] use numerically derived gradient information of the constraint evaluations for repairing infeasible solutions.

Other techniques handle constraints by optimizing constraint functions and objective function separately in a lexical order [15, 16]. Stochastic ranking [17] is another example of this approach. Also multi-objective optimization algorithms have been designed for constraint-based optimization, considering the constraint functions as additional objectives [17, 18]. Beyer and Finck [19] propose an extension of CMA-ES which allows to handle special types of constraints successfully.

In the field of model-assisted optimization algorithms for constrained problems, Support Vector Machines (SVMs) have been used by Poloczek and Kramer [20]. They make use of SVMs as a classifier for predicting the feasibility of solutions, but achieve only slight improvements. Powell [21] proposes COBYLA, a direct search method which models the objective and the constraints by linear functions. Recently, Regis [4] developed COBRA, an efficient solver that makes use of Radial Basis Function (RBF) interpolation, and outperforms most algorithms in terms of required function evaluations on a large number of benchmark functions.

In this paper we analyze a new implementation of COBRA in `R` [22], which allows easy adaptation of certain components of the algorithm. In Sec. 2 we present the problem and the algorithm in more detail. In Sec. 3 we perform a thorough experimental study on analytical test functions and on a real-world benchmark function MOPTA 2008 [1]. The results are discussed in Sec. 4 and we give conclusive remarks in Sec. 5.

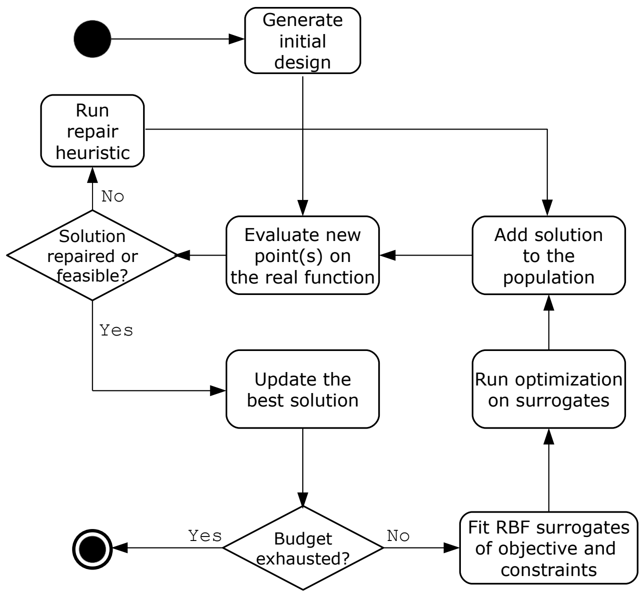
Generate initial design
Run repair heuristic
No
Solution repaired or feasible?
Evaluate new point(s) on the real function
Add solution to the population
Yes
Update the best solution
Run optimization on surrogates
Yes
Budget exhausted?
No
Fit RBF surrogates of objective and constraints

Figure 1: Flowchart of the COBRA algorithm.

2. METHODS

2.1 Constrained Optimization by Radial Basis Function Approximation

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

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

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

Constrained Optimization by Radial Basis Function Approximation (COBRA) was proposed by Regis [4]. The main idea of this method is to use approximations of the objective function and the constraint functions for saving real function evaluations. This can be especially beneficial for industrial optimization problems where function evaluations are expensive. Internally, COBRA uses radial basis function interpolation for modeling the objective and constraints. In each iteration k the solution $\vec{x}^{(k)}$ to be evaluated on the real function f is the result of an optimization performed on the RBF surrogates consisting of surrogates of the objective and the constraint functions.

Fig. 1 presents a flowchart of the algorithm. In the beginning an initial population or design is generated for building the initial RBF models. The initial solutions can be selected, e. g., by classical design of experiments (DoE) [23]. Another option is to perform an initial local search with a limited number of function evaluations using an unconstrained optimizer (e. g., Hooke & Jeeves pattern search [24]). The sequence of points in this local search is then used as the initial design. During this local search, we penalize infeasible solutions with a simple penalty function.

The resulting RBF surrogates from the initial design are used to find the next iterate to be evaluated on the real function. Therefore, a sequential search is performed on the surrogate functions. The best point obtained from this search is referred to as *infill point* in the remainder of this paper. Note that only the infill points are evaluated on the real function. This makes the algorithm efficient in terms of saving real function evaluations. If the infill point is better than the current best solution, the best solution is replaced by the infill point. In any case the RBF models are updated using the new information. In the next round a new search on the updated surrogate functions is performed. These steps are repeated, until the the budget for the optimization exceeds a given limit.

Distance requirement cycle.

A distance requirement is integrated in COBRA which defines how closely a new infill solution may approach the previous ones. The idea is to avoid frequent updates in the direct neighbourhood of the best solution at present. The distance requirement can be passed by the user as external parameter. This parameter is a *vector* of distance requirements whose elements are cyclically reused. In each iteration COBRA checks as additional constraint the distance of the proposed infill solution to all previous infill points. If this constraint is not fulfilled, COBRA handles this as a normal constraint violation, e. g., the solution becomes infeasible. The distance requirement cycle is a clever idea, since the user can allow for switching between exploration and exploitation of the search space.

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 regularization factor is added to the predicted constraint values. As a consequence, solutions evaluated on the RBF models are only feasible, if they are not too close to the constraint boundary. In other words, the regularization factor would correct small errors of too optimistic constraint models, and thereby forces the algorithm to more likely create feasible infill solutions.

2.2 Repairing infeasible solutions

In this paper we propose a new repair mechanism RI-2 for reducing the constraint violation of infill solutions obtained from the internal optimization algorithm. We compare the method with the algorithm of Chootinan and Chen [5].

2.2.1 RI-2

A repair mechanism for real-valued constraints can benefit from using gradient information of the constraint functions. If the gradient is not available analytically – as in the case of black-box constraints – it can be calculated numerically. This can be however too costly in the case of expensive black-box functions. Within the COBRA framework we take advantage of the surrogate functions and calculate the gradient *on the constraint surrogates* at no extra cost.

Before we describe our repair algorithm RI-2 in formal details, we have a look at two instructive examples:

A typical situation as it may arise with multiple constraints is shown in Figure 2. For simplicity we show only two violated constraints here, but the same idea applies for multiple violated constraints as well. The green and blue arrow show the negative gradient direction for the green

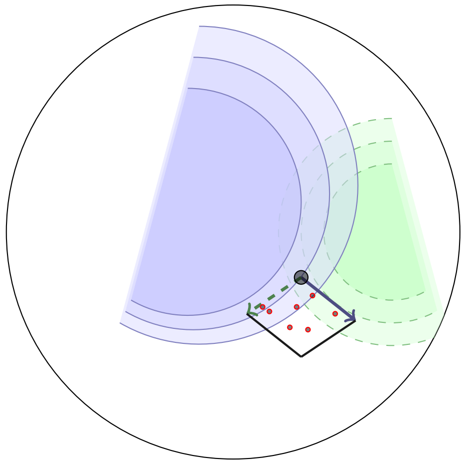

Figure 2: Repair strategy in the case of two constraints (blue-solid and green-dashed). The white area is the feasible region.

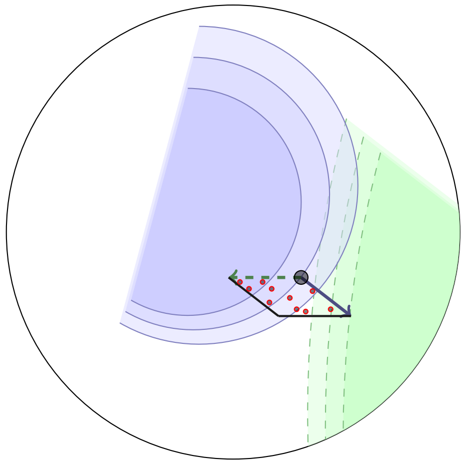

Figure 3: Repair strategy in the case of one violated constraint (blue-solid) and one constraint (green-dashed) being ϵ-infeasible.

and blue constraint, resp. The length of each arrow is q times larger than the estimated distance to the constraint boundary, where $q > 1$ is a parameter of the algorithm. Two or more such arrows form a parallelepiped. Our repair algorithm RI-2 draws m_{max} random points from this parallelepiped (red points) and tests whether they are feasible on the constraint surrogates. One of these surrogate-feasible points is selected (according to some quality measures detailed below) and returned.

A second typical situation is shown in Fig. 3: Here the current infill point violates only the blue constraint. But a correction along the direction of the blue arrow would make the new point infeasible in the second constraint (green). We define a point $\vec{x}$ to be **ϵ-feasible** in constraint s_i if

$$s_i(x) + \epsilon \leq 0.$$

Otherwise it is said to be **ϵ-infeasible**. As a solution to the problem in Fig. 3 we propose to treat all constraints which are ϵ-infeasible as violated constraints. Then we have again a blue and a green arrow forming a parallelepiped. Among the random realizations in this parallelepiped (red points) a feasible point can be found.

Algorithm 1 Repair algorithm RI-2. Input: an infeasible point $\vec{x} \in \mathbb{R}^d$ and its true constraint vector $\vec{s}(\vec{x}) \in \mathbb{R}^m$. Output: a suggestion $\vec{x} + \vec{\Delta}$ of a nearby feasible point.

```
 1: function RI-2($\vec{x}, \vec{s}(\vec{x})$)
 2:     Set $\mathbf{E}$ to the $d \times d$ identity matrix
 3:     $\vec{g}^{(k)}$: gradient at $\vec{x}$ of the $k$'th surrogate
 4:     $I$: set of $\epsilon$-infeasible constraints for $\vec{x}$.
 5:     Calculate for each constraint $k \in I$:
            $\vec{\Delta}^{(k)} = -\frac{s_k(\vec{x}) + \epsilon}{\|\mathbf{E}\vec{g}^{(k)}\|^2} \mathbf{E}\vec{g}^{(k)}$        (1)
 6:     $\vec{\Delta}$ = BestRandomRealization($\vec{\Delta}^{(k)}$)
 7:     Check if $\vec{x} + \vec{\Delta}$ is outside the search region in any dimension $D$. If so, then set the $D$'th diagonal element(s) of $\mathbf{E}$ to zero and repeat from Step 5.
 8:     return $(\vec{x} + \vec{\Delta})$
 9: end function
10:
11: function BestRandomRealization($\vec{\Delta}^{(k)}$)
12:     $R = \{\}$
13:     for $m = 1, \ldots, m_{max}$ do
14:         Draw $|I|$ random coefficients $\alpha_k \in [0, q]$
15:         $R = R \cup \{\sum_{k \in I} \alpha_k \vec{\Delta}^{(k)}\}$
16:     end for
17:     $S = \{\vec{r} \in R \mid \vec{x} + \vec{r}$ is $\epsilon$-feasible on all surrogates$\}$
18:     if $S \neq \{\}$ then
19:         $\vec{r}^{(bst)}$ = member of $S$ with minimal length
20:     else
21:         $n_v$ = minimum number of violated constraints
22:         $T = \{\vec{r} \in R \mid \vec{x} + \vec{r}$ has $n_v$ violated constraints$\}$
23:         $\vec{r}^{(bst)}$ = member of $T$ with lowest max(violation)
24:     end if
25:     return $\vec{r}^{(bst)}$
26: end function
```

The complete description of our repair algorithm RI-2 is given in Algorithm 1. The algorithm has three parameters ϵ, q and m_{max}. Note that all constraint calculations are done on the surrogate models, so no extra evaluations of the true constraint functions are needed inside RI-2.

The step $\vec{\Delta}^{(k)}$ in Eq. (1) is designed as follows: If s_k were the only violated constraint and if it were a linear function, then $\vec{\Delta}^{(k)}$ would bring us directly to the border of ϵ-feasibilty in s_k. If the nearest border point is outside the search region, then we move with the help of matrix $\mathbf{E}$ to another border point which is just inside the search region (Fig. 4).

Among the random realizations the best point is selected according to these rules: (a) If ϵ-feasible solutions exist, then select the one closest to $\vec{x}$. This should change the objective value as little as possible. (b) If not, select among the solutions with the mimumum number of violated constraints the one with lowest maximum violation.

2.2.2 *Chootinan Repair*

A repair method based on derivatives of the constraint functions has been also proposed by Chootinan and Chen [5]. They calculate the gradients of all violated constraints similar to RI-2 and put them into a matrix $\nabla_x V$. Then the Moore-Penrose inverse or pseudoinverse $(\nabla_x V)^+$ is taken. In each time step t the infeasible solution is updated by adding the pseudoinverse $(\nabla_x V)^+$ at infill point $\vec{x}^{(t)}$ times the constraint violations $\Delta\vec{V}$ at $\vec{x}^{(t)}$:

$$\vec{x}^{(t+1)} = \vec{x}^{(t)} + (\nabla_x V)^+ \cdot \Delta\vec{V} \qquad (2)$$

This is iterated until $\vec{x}^{(t+1)}$ becomes feasible or until its change drops below a predefined threshold η.

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 can lead to quick fixes of constraint violations, it can also induce possible new violations of constraint functions. In this paper we undertake a comparison between Chootinan and Chen's repair and RI-2.

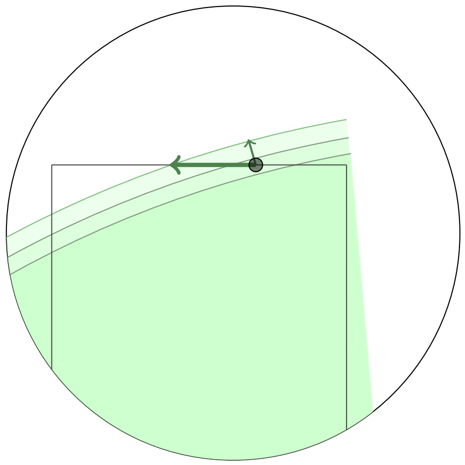

Figure 4: Strategy for staying in the search region: The current point (grey) is infeasible and on the boundary of the search space (black rectangle). Thus, a move in the negative gradient direction (thin arrow) is forbidden. The solution: project out the forbidden dimension(s) (matrix E) and make a larger move in the allowed direction (thick arrow).

3. EXPERIMENTAL ANALYSIS

3.1 Test functions

3.1.1 *MOPTA 2008*

The MOPTA 2008 benchmark (MOPTA08) by Jones [1] is a substitute for high-dimensional real-world problems encountered in the automotive industry: It is a problem with $d = 124$ dimensions and with 68 constraints. The problem should be solved within $1860 = 15 \cdot d$ function evaluations which in reality refers to one month of computation time on a high-performance computer.

3.1.2 *G-problem test suite*

For further evaluation we use popular benchmark functions, namely the G-problem test suite described in [6]. The underlying functions differ in dimension ($d = 2, \ldots, 20$), type of objective function (linear, quadratic, nonlinear) and number and type of constraints ($m = 1, \ldots, 9$). Since these functions are often used in the scientific literature for analyzing constrained-based solvers, they provide a good analytical testbed for our algorithms.

3.1.3 Experimental setup

Each experiment was repeated 10 times in the case of MOPTA08 and 30 times in the case of the G-problems, each time with a different random seed.

The initial design was either obtained from sampling LHS random points for the G-functions or from an initial optimization run using Hooke & Jeeves pattern search with additional penalty term for the MOPTA08 benchmark. The other parameters for COBRA (distance requirement cycle and others) were taken from the best setting COBRA-LOCAL in Regis' original work [4].

We compare three repair options: our algorithm RI-2, the method of Chootinan (CHO) [5] and no repair. The RI-2 algorithm was used with parameters $q = 3, \epsilon = 10^{-4}$ and $m_{max} = 1000$. The CHO algorithm was used with parameter $\eta = 10^{-5}$.

In our experiments we used the COBYLA [21] implementation from the *nloptr* R package version version 1.0.4. [1] With this implementation, some of our runs resulted in occasional "freezes" of the COBYLA-optimizer, making it necessary to implement a stopping mechanism in case of this undesirable behaviour. For statistical convenience we did not include these incomplete runs, but set up new runs until 10 or 30 runs for MOPTA08 or the G-problems, resp., were completed.

3.2 Results

3.2.1 MOPTA 2008

COBRA-R with the new initial design method (taking the iterates from local search with Hooke & Jeeves, cf. Sec. 2.1) shows in Fig. 5 a clear benefit for the first 500 iterations as compared to the initial design from Regis' COBRA (diamond symbols in Fig. 5, COBRA-LOCAL from [4]).

But after the initial design, our first solution (Fig. 5, red curve *COBRA-R w/o repair*) got stuck and only after 1500 iterations there was some progress to a mediocre mean objective function value 236. We noticed however that COBYLA was making steady progress on the objective function, but always with slightly infeasible solutions. This can be seen in Fig. 6 where we plot the best objective value when allowing each constraint to be violated up to 0.5%. Now the red curve comes much closer to the optimum.

This led us to the development of the repair algorithm RI-2. The repair algorithm improves the performance of COBRA-R a lot, as the curve *COBRA-R with repair [RI-2]* in Fig. 5 shows. This curve is remarkable for two reasons: (a) The best feasible solution improves dramatically fast in the first 1000 iterations, leading to objective value 226 at iteration 1000, significantly better than Regis' result. (b) The results are pretty stable with respect to random initial conditions of 10 runs: In Fig. 5 both the standard deviations and the spans between best and worst solution are much smaller than for the other COBRA-R curves.

We look now with more detail into the repair results: An inspection of the iterates shows that the repair algorithm

[1]http://cran.r-project.org/web/packages/nloptr/index.html

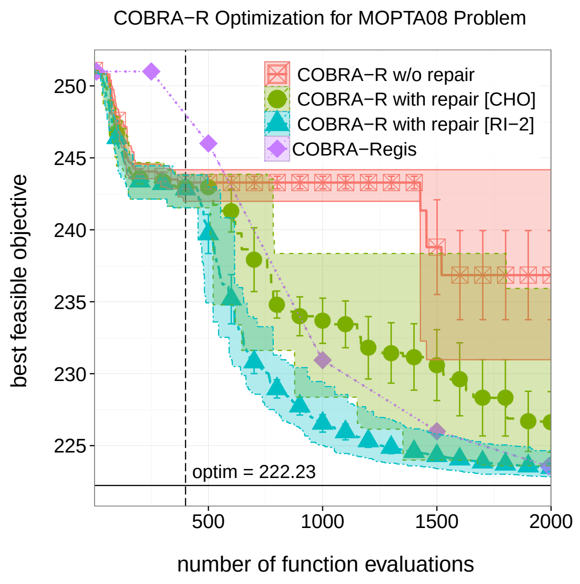

Figure 5: COBRA-R optimization performance for MOPTA08 problem. Every point represents the mean of ten independent runs. The ribbons around the curves indicate the best and the worst run. Error bars indicate standard deviations.

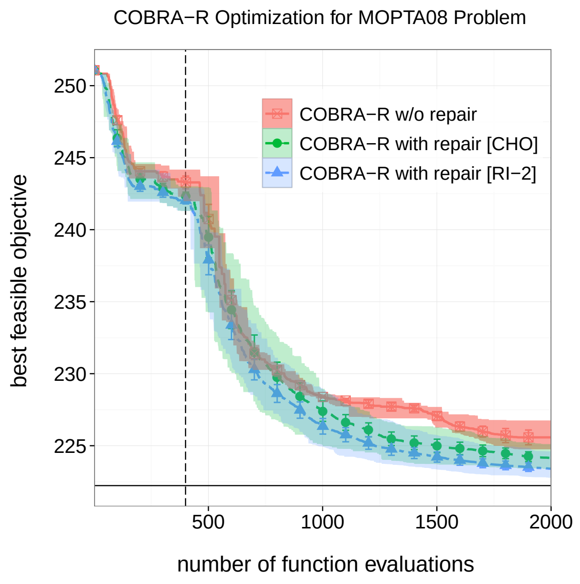

Figure 6: COBRA-R optimization performance if we allow an infeasibility tolerance of 0.5%.

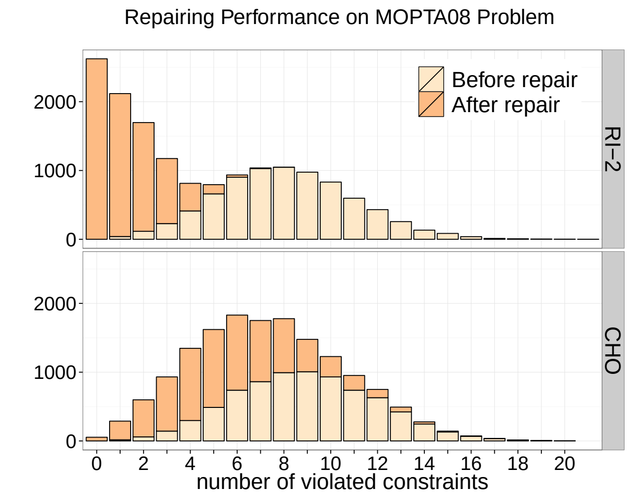

Figure 7: Number of violated constraints for MOPTA08 benchmark with 68 constraints. For each infill point entering and leaving the repair method we count its number of violated constraints. The bars are stacked.

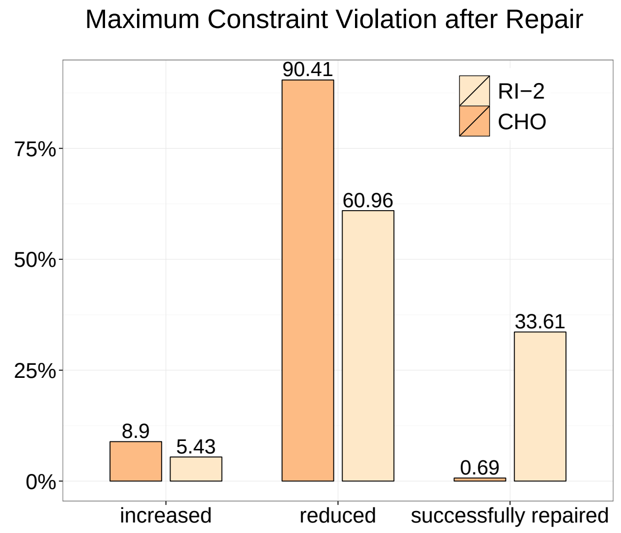

Figure 8: Comparing the performance of different repair methods on maximum violation for MOPTA08 problem. More than 30% of RI-2 repair calls contribute a fully feasible point. The success rate of CHO is less than 1%.

is called roughly every second iteration. This means that COBYLA produces mostly infeasible solutions.

How can these infeasible solutions be characterized? It is seen in Fig. 7 that the number of violated constraints is pretty large: 8-9 violated constraints on average and some extreme cases with up to 18 violated constraints. (The number of ϵ-infeasible constraints may be even higher.) This means that each repair algorithm faces a tough constraint satisfaction problem to be solved. Algorithm RI-2 very often reduces the number of violated constraints to zero or to a small number, while the method of Chootinan (CHO) only seldomly succeeds in doing this.

Figure 8 shows the success of repair on MOPTA08: While both methods, RI-2 and CHO, only rarely increase the maximum violation, RI-2 is much better in producing feasible solutions (successfully repaired $> 30\%$) than CHO ($< 1\%$).

3.2.2 *G-problem test suite*

In our previous work [25] we have studied the G-problem test suite in detail and compared COBRA-R on this test suite with other state-of-the-art constraint optimization algorithms [5],[17],[21]. The main result was that COBRA-R could achieve on all G-problems a performance similar to the other algorithms, but needed only a fraction of true function evaluations (fe) grace to the surrogate models: 50–360 fe for COBRA-R as compared to 6 000–350 000 fe for [5],[17]). The only exception was problem G02 where COBRA-R did not achieve good results, because the objective function in 20 dimensions has many local peaks and is not modelled well by the surrogate functions. We therefore exclude G02 from the following comparison.

Here we concentrate on the comparison of the different repair options on the G-problem test suite. Table 1 shows the results: Three problems are significantly better solved with repair, three are better solved without repair and three problems are indifferent. Among the two repair algorithms RI-2 is always equal or better than CHO.

The significance was tested with the Wilcoxon signed rank sum test (Table 2), which is a paired test on the 30 pairs with different seeds. In Table 3 we count the significant wins and losses and find slight advantages for RI-2 over *no repair* over Chootinan.

Table 1 shows in addition the repair success probability $p_{success}$, i. e. the probability that an infeasible solution becomes feasible on all true constraints after repair. On most G-problems the method RI-2 has a significantly larger $p_{success}$ than CHO (with the exception of problem G06).

4. DISCUSSION

4.1 Repairing infeasible solutions

4.1.1 *MOPTA 2008*

The results in Sec. 3.2.1 on MOPTA08 show that a repair mechanism is urgently needed in the case of COBRA-R with COBYLA as internal optimizer. The reason can be traced back to COBYLA (in combination with COBRA-surrogates): COBYLA produces mostly solutions which are very good in objective value, but slightly infeasible. It has to be noted that the Regis' original COBRA implementation [4] does not need a repair mechanism on the same benchmark because it uses another optimizer (MATLAB's `fmincon`, an interior point optimizer).

But the extra repair effort needed for COBYLA has a payoff: In conjunction with repair method RI-2 we see a faster convergence in Fig. 5, i. e. after 1000 true function evaluations RI-2 reaches a better feasible objective value than Regis' COBRA [4].

The repair method CHO fails on the MOPTA08 benchmark. Why is this the case? – A closer inspection of the results in Fig. 9 shows that CHO often reduces the maximum

Table 1: Comparison of different repair options (RI-2: our algorithm, CHO: Chootinan [5], none) on the G-problem test suite. median(dev): median of the deviation from the optimal objective value in 30 runs, $p_{success}$: probability that a repair attempt results in a feasible solution. Numbers in bold face: significantly better than other options in same row.

repair	RI-2	CHO	none	RI-2	CHO
	median(dev)			$p_{success}$	
G01	7.5E-05	7.5E-05	7.5E-05	**1.000**	0.000
G03	8.5E-02	6.8E-02	**4.1E-02**	**1.000**	0.483
G04	**2.3E-07**	1.4E-05	2.3E-06	**0.976**	0.281
G05	3.0E-04	2.9E-04	3.0E-04	**0.452**	0.296
G06	2.1E-03	2.7E-03	2.1E-03	0.006	**0.103**
G07	2.4E-06	2.4E-06	**5.4E-07**	**0.391**	0.195
G08	**7.1E-07**	1.3E-06	2.3E-06	**0.774**	0.418
G09	4.0E-05	2.1E-05	**6.8E-06**	**0.873**	0.395
G10	**8.7E-02**	**8.7E-02**	3.5E-01	0.196	0.172

Table 2: Wilcoxon rank sum test, paired, one sided, significance level 5%. Repair methods: RI-2, CHO: Chootinan [5], RN: *no repair*.

test	for problem
RI-2 significantly better than CHO	G04, G08
CHO significantly better than RI-2	G03, G09
RN significantly better than RI-2	G07, G09
RN significantly better than CHO	G04, G07, G09
RI-2 significantly better than RN	G04, G08, G10
CHO significantly better than RN	G08, G10

violation largely, but either it leaves a small reminiscent in some of the initially violated constraints or it repairs the violated constraints, but has in the end some other constraints violated which were feasible initially. It may be that method CHO would benefit from the concept of ϵ-feasibility as well, but we did not test this.

As a consequence of the lower probability of successful repair for method CHO, the span between best and worst solution is larger in this case than it is for method RI-2 (Fig. 5).

4.1.2 *G-problem test suite*

On most G-problems we get with repair method RI-2 a significantly higher success rate $p_{success}$ than with CHO (Table 1). But the overall effect of the repair is not that important for the final deviation from the optimum as in the MOPTA08 benchmark: We see cases where the final deviation is nearly the same in all three cases (e. g. G05, G06) and cases where we see higher deviation despite a higher success rate (G09). Sometimes the reason is that in contrast to MOPTA08 only few iterations of the G-problems require a repair. On the other hand, there are also cases where the repair gives a significant improvement (G04, G08, G10).

Table 3: Significant wins and losses on the 5%-level for the G-problem test suite.

method	significant wins	significant losses	delta
RI-2	5	4	1
none	5	5	0
CHO	4	5	-1

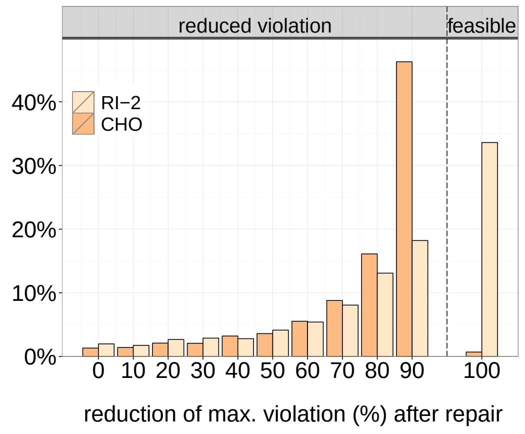

Figure 9: Reduction of maximum constraint violation

Why is it that in some cases the *no repair* method is better than RI-2 or CHO (G03, G07, G09)? – This result, surprising at first sight, is due to the fact that we allow only very few true function evaluations (50–360). If repair does not help to find better feasible objective values, then the fact that a considerable amount of function evaluations is devoted to repair can lead to a deterioration of the overall result. If we increase the number of function evaluations, the winning margin of *no repair* would decrease or even vanish.

A final note: The results on the G-problems with repair method CHO may differ from the results in Chootinan's original work [5]. This is because we use method CHO in the context of COBRA, while [5] has embedded it in a GA optimization scheme using more evaluations on the real function.

5. CONCLUSION

In this contribution we have studied various solvers for constrained optimization problems under severely limited budgets. Regis' COBRA method [4] based on surrogate models was re-implemented in `R` with a new internal optimizer COBYLA. We could confirm that COBRA leads to good results within few function evaluations on a variety of constrained optimization problems. In contrast to [4], we needed a repair method for infeasible solutions in the MOPTA08 benchmark. We proposed a new repair method (RI-2), which performed well on MOPTA08. It led in conjunction with COBYLA to the same asymptotic result as in [4], but with a faster rate of convergence. At the same time, RI-2 reduces the standard deviation as compared to the runs without repair. Thus we can give a positive answer to our research question (H1) on the MOPTA08 benchmark.

In a direct comparison of our new repair mechanism RI-2 with another gradient-based strategy CHO [5], we found both repair methods to perform similarly on the G-problem test-suite, where the number of constraints is relatively small. But on the MOPTA08 benchmark with its large number of constraints, the situation changes: RI-2 has a significantly higher success rate for solving constraint violations (Fig. 8), so that we can give a positive answer to research question (H2). It was shown that both repair operators RI-2 and CHO often decrease the intensity of constraint violations (Fig. 7 and 8). However, RI-2 is capable to transfer an infeasible solution into a feasible solution in more than 30% of the operator calls, whereas CHO has a very low success rate of less than 1% (cf. research question (H3)). Our discussion came to the conclusion that probably the notion of ϵ-feasibility is responsible for this beneficial effect. Since RI-2 performs all calculations on the surrogate models, it does not cost any extra function evaluations. The repair method RI-2 is not restricted to COBRA, it can be applied to any optimization algorithm with surrogate models.

6. REFERENCES

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

[2] T. Runarsson and X. Yao, "Stochastic ranking for constrained evolutionary optimization," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 3, pp. 284–294, 2000.

[3] O. Kramer and H.-P. Schwefel, "On three new approaches to handle constraints within evolution strategies," *Natural Computing*, vol. 5, no. 4, pp. 363–385, 2006.

[4] R. Regis, "Constrained optimization by radial basis function interpolation for high-dimensional expensive black-box problems with infeasible initial points," *Engineering Optimization*, vol. 46, no. 2, pp. 218–243, 2013.

[5] P. Chootinan and A. Chen, "Constraint handling in genetic algorithms using a gradient-based repair method," *Computers & Operations Research*, vol. 33, no. 8, pp. 2263–2281, 2006.

[6] Z. Michalewicz and M. Schoenauer, "Evolutionary algorithms for constrained parameter optimization problems," *Evolutionary Computation*, vol. 4, no. 1, pp. 1–32, 1996.

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

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

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

[10] E. Mezura-Montes and C. Coello Coello, "Constraint-handling in nature-inspired numerical optimization: past, present and future," *Swarm and Evolutionary Computation*, vol. 1, no. 4, pp. 173–194, 2011.

[11] J. Xiao, Z. Michalewicz, L. Zhang, and K. Troja nowski, "Adaptive evolutionary planner/navigator for mobile robots," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 18–28, 1997.

[12] H. Mühlenbein, "Parallel genetic algorithms in combinatorial optimization," in *Computer Science and Operations Research*, O. Balci, R. Sharda, and S. Zenios, Eds. Pergamon, 1992, pp. 441–456.

[13] D. Orvosh and L. Davis, "Using a genetic algorithm to optimize problems with feasibility constraints," in *Proc. 1st IEEE Conference on Evolutionary Computation.* IEEE, 1994, pp. 548–553.

[14] D. Tate and A. Smith, "A genetic approach to the quadratic assignment problem," *Computers & Operations Research*, vol. 22, no. 1, pp. 73–83, 1995.

[15] T. Takahama and S. Sakai, "Constrained optimization by applying the alpha-constrained method to the nonlinear simplex method with mutations," *IEEE Trans. Evolutionary Computation*, vol. 9, no. 5, pp. 437–451, 2005.

[16] ——, "Learning fuzzy control rules by alpha-constrained simplex method," *Systems and Computers in Japan*, vol. 34, no. 6, pp. 80–90, 2003.

[17] T. Runarsson and X. Yao, "Search biases in constrained evolutionary optimization," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 35, no. 2, pp. 233–243, 2005.

[18] A. H. Aguirre, S. B. Rionda, C. A. Coello Coello, G. L. Lizárraga, and E. M. Montes, "Handling constraints using multiobjective optimization concepts," *International Journal for Numerical Methods in Engineering*, vol. 59, no. 15, pp. 1989–2017, 2004.

[19] H.-G. Beyer and S. Finck, "On the design of constraint covariance matrix self-adaptation evolution strategies including a cardinality constraint," *IEEE Trans. Evolutionary Computation*, vol. 16, no. 4, pp. 578–596, 2012.

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

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

[22] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2013. [Online]. Available: http://www.R-project.org/

[23] R. Fisher, "Design of experiments," *British Medical Journal*, vol. 1, no. 3923, 1936.

[24] C. Kelley, *Iterative methods for optimization.* SIAM, 1999, vol. 18.

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