An Adaptive Radial Basis Algorithm (ARBF) for Expensive Black-Box Mixed-Integer Constrained Global Optimization

Kenneth Holmström*, Nils-Hassan Quttineh* and Marcus M. Edvall†

Abstract Response surface methods based on kriging and radial basis function (RBF) interpolation have been successfully applied to solve expensive, i.e. computationally costly, global black-box nonconvex optimization problems. In this paper we describe extensions of these methods to handle linear, nonlinear, and integer constraints. In particular, algorithms for standard RBF and the new adaptive RBF (ARBF) are described. Note, however, while the objective function may be expensive, we assume that any nonlinear constraints are either inexpensive or are incorporated into the objective function via penalty terms. Test results are presented on standard test problems, both nonconvex problems with linear and nonlinear constraints, and mixed-integer nonlinear problems (MINLP). Solvers in the TOMLAB Optimization Environment (http://tomopt.com/tomlab/) have been compared, specifically the three deterministic derivative-free solvers rbfSolve, ARBFMIP and EGO with three derivative-based mixed-integer nonlinear solvers, OQNLP, MINLPBB and MISQP, as well as the GENO solver implementing a stochastic genetic algorithm. Results show that the deterministic derivative-free methods compare well with the derivative-based ones, but the stochastic genetic algorithm solver is several orders of magnitude too slow for practical use. When the objective function for the test problems is costly to evaluate, the performance of the ARBF algorithm proves to be superior.

Keywords: Global optimization, radial basis functions, response surface model, surrogate model, expensive function, CPU-intensive, optimization software, splines, mixed-integer nonlinear programming, nonconvex, derivative-free, black-box, linear constraints, nonlinear constraints.

Abbreviations:

RBF	Radial Basis Function
CGO	Costly Global Optimization
MINLP	Mixed-Integer Nonlinear Programming

*Department of Applied mathematics, Mälardalen University, SE-721 23 Västerås, Sweden.
†Tomlab Optimization Inc., 1260 SE Bishop Blvd Ste E, Pullman, WA 99163-5451, USA.

1 Introduction

Global optimization of continuous black-box functions that are costly (computationally expensive, CPU-intensive) to evaluate is a challenging problem. Several approaches based on response surface techniques, most of which utilize every computed function value, have been developed over the years. In his excellent paper [9], Jones reviews the most important developments. Many methods have been developed based on statistical approaches, called kriging, see e.g. the Efficient Global Optimization (EGO) method in Jones et al. [10]. In this paper we mainly consider methods based on radial basis function interpolation, RBF methods, first discussed in [4] and [13].

Problems that are costly to evaluate are commonly found in engineering design, industrial and financial applications. A function value could be the result of a complex computer program, an advanced simulation, e.g. computational fluid dynamics (CFD), or design optimization. One function value might require the solution of a large system of partial differential equations, and hence consume anything from a few minutes to many hours. In the application areas discussed, derivatives are most often hard to obtain and the algorithms make no use of such information. The practical functions involved are often noisy and nonsmooth; however, the commonly used approximation methods assume smoothness. Another area illustrating the challenges of optimization with expensive function evaluations is space mapping optimization, see e.g. [1]. Instead of one costly function value, in space mapping a vector valued function is the result of each costly evaluation. Companion "coarse" (ideal or low-fidelity) and "fine" (practical or high-fidelity) models of different complexities are intelligently linked together to solve engineering model enhancement and design optimization problems.

Our goal is to develop global optimization algorithms that work in practice and produce reasonably good solutions with a very limited number of function evaluations. From an application perspective there are often restrictions on the variables besides the lower and upper bounds, such as linear, nonlinear or even integer constraints. Henceforth, we seek to solve the complicated problem formulated as follows:

The Mixed-Integer Costly Global Black-Box Nonconvex Problem

$$
\begin{aligned}
\min_{x} \quad & f(x) \\
s/t \quad & \begin{array}{rccccl}
-\infty < & x_L & \leq & x & \leq & x_U \quad < \infty \\
& b_L & \leq & Ax & \leq & b_U \\
& c_L & \leq & c(x) & \leq & c_U \\
& & x_j \in \mathbb{N} & \forall j \in \mathbb{I} & &
\end{array} ,
\end{aligned}
\tag{1}
$$

where $f(x) \in \mathbb{R}$; $x_L,\ x,\ x_U \in \mathbb{R}^d$; the m_1 linear constraints are defined by $A \in \mathbb{R}^{m_1 \times d}$, $b_L,\ b_U \in \mathbb{R}^{m_1}$; and the m_2 nonlinear constraints are defined by $c_L,\ c(x),\ c_U \in \mathbb{R}^{m_2}$. The variables x_I are restricted to be integers, where $\mathbb{I}$ is an index subset of $\{1,\ldots,d\}$. Let $\Omega_C \in \mathbb{R}^d$ be the feasible set defined by all the constraints in (1) and $\Omega \in \mathbb{R}^d$ be the feasible set defined only by the box constraints, the simple bounds. We assume that the function $f(x)$ is continuous with respect to all variables, even though we demand that some variables only take integer values. Otherwise it would not make sense to do surrogate modeling of $f(x)$. Another assumption is that the nonlinear constraints are

cheap to compute compared to the costly $f(x)$. All costly constraints can be treated by adding penalty terms to the objective function in the following way:

$$\min_x \quad p(x) = f(x) + \sum_j w_j \max\left(0,\ c^j(x) - c_U^j,\ \ c_L^j - c^j(x)\right), \tag{2}$$

where weighting parameters w_j have been added. As we have shown in [2] this strategy works in practice for an industrial train set design problem.

The idea of the RBF algorithm by Powell and Gutmann [4] is to use radial basis function interpolation to build an approximating surrogate model and define three utility functions. The next point, where the original objective function should be evaluated, is determined by optimizing one or more of these utility functions. Roughly speaking, the utility functions measure the likelihood that the solution to the problem occurs at a given point with the objective function equal to a certain "target value". Maximizing the utility function therefore provides the point most likely to be a solution to the problem if that the optimal objective equals the target value. Clearly, different target values result in different points being suggested for further search. In the RBF methods of Gutmann and Powell, a non-adaptive (static) scheme is used to select the target values; unfortunately, as we show later, this can lead to the sampling of many points on the boundary of the space that help little to advance the search. To deal with this problem, Holmström [7] proposes a more general adaptive approach to set target values, an Adaptive RBF algorithm (ARBF). Instead of the static choice, a one-dimensional search for a suitable target value is done to improve convergence. This leads to a sequence of global optimization problems to be solved in each iteration. The above mentioned papers only consider a box-bounded region Ω, whereas in this paper the goal is to solve the MINLP problems as defined by (1). The convergence of the ARBF method is discussed in Holmström [7] and is based on the same arguments as for the RBF method, discussed in the thesis of Gutmann [5].

In Section 2 the RBF interpolation method and the extensions of the RBF algorithm for MINLP are described. A detailed presentation of the new Adaptive RBF algorithm is given in Section 3, with some additions compared to Holmström [7] to handle MINLP problems. In Section 4 the implementations in TOMLAB [6, 8] of the given algorithms are described.

The approach to handle mixed-integer constrained problems is validated with tests on a set of standard MINLP problems. Results for these problems and some nonconvex constrained problems are given in Section 5. The same section also compares the results from seven different MINLP solvers in the TOMLAB optimization environment. Section 6 gives some concluding remarks.

2 The RBF method for MINLP

First, the surrogate model used in the RBF method is defined. Given n distinct points $x_1, \ldots, x_n \in \Omega$ with known function values $F_i = f(x_i)$, $i = 1, \ldots, n$, the radial basis function interpolant s_n has the form

$$s_n(x) = \sum_{i=1}^{n} \lambda_i \phi\left(\|x - x_i\|_2\right) + p(x), \tag{3}$$

where $\|\cdot\|$ is the Euclidean norm, $\lambda_1, \ldots, \lambda_n \in \mathbb{R}$ and p is in Π_m^d (the space of polynomials in d variables of degree less than or equal to m). Common choices of radial basis functions ϕ and the corresponding polynomial $p(x)$ and minimal polynomial degree m_ϕ are given in Table 1. When ϕ is either cubic with $\phi(r) = r^3$ or thin plate spline with $\phi(r) = r^2 \log r$, the radial basis function interpolant s_n has the form

$$s_n(x) = \sum_{i=1}^{n} \lambda_i \phi\left(\|x - x_i\|_2\right) + b^T x + a, \tag{4}$$

with $\lambda_1, \ldots, \lambda_n \in \mathbb{R}$, $b \in \mathbb{R}^d$, $a \in \mathbb{R}$. The unknown parameters λ_i, b, a are obtained as the solution of the linear equations

$$\begin{pmatrix} \Phi & P \\ P^T & 0 \end{pmatrix} \begin{pmatrix} \lambda \\ c \end{pmatrix} = \begin{pmatrix} F \\ 0 \end{pmatrix}, \tag{5}$$

where Φ is the $n \times n$ matrix with $\Phi_{ij} = \phi\left(\|x_i - x_j\|_2\right)$ and

$$P = \begin{pmatrix} x_1^T & 1 \\ \vdots & \vdots \\ x_n^T & 1 \end{pmatrix}, \quad \lambda = \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{pmatrix}, \quad c = \begin{pmatrix} b_1 \\ \vdots \\ b_d \\ a \end{pmatrix}, \quad F = \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix}. \tag{6}$$

If $\text{rank}(P) = d + 1$, the matrix $\begin{pmatrix} \Phi & P \\ P^T & 0 \end{pmatrix}$ is nonsingular and system (5) has a unique solution [12]. Thus a unique radial basis function interpolant to f at the points $x_1, \ldots, x_n$ is obtained. After this, one has to consider the question of choosing the next point x_{n+1} to evaluate the objective function for. The idea of the RBF algorithm is to use radial basis function interpolation and a measure of "bumpiness" of a radial function, σ. A target value f_n^* is chosen as an estimate of the global minimum of f. For each $y \notin \{x_1, \ldots, x_n\}$ there exists a radial basis function $s_y(x)$ that satisfies the interpolation conditions

$$\begin{aligned} s_y(x_i) &= f(x_i), \quad i = 1, \ldots, n, \\ s_y(y) &= f_n^*. \end{aligned} \tag{7}$$

The next point x_{n+1} is then calculated as the value of y in the feasible region that minimizes $\sigma(s_y)$. As a surrogate model is used, the function $y \mapsto \sigma(s_y)$ is much cheaper to compute than the original function.

In [3], a "bumpiness" measure $\sigma(s_n)$ is defined and it is shown that minimizing $\sigma(s_y)$ subject to the interpolation conditions (7) is equivalent to minimizing a utility function $g_n(y)$ defined as

$$g_n(y) = (-1)^{m_\phi + 1} \mu_n(y) \left[s_n(y) - f_n^*\right]^2, \quad y \in \Omega \setminus \{x_1, \ldots, x_n\}. \tag{8}$$

The method of Gutmann and the bumpiness measure is further discussed in the more recent papers [14] and [15] by Regis and Shoemaker. Writing the radial basis function solution to the target value interpolation problem (7) as

$$s_y(x) = s_n(x) + \left[f_n^* - s_n(y)\right] l_n(y, x), \quad x \in \mathbb{R}^d, \tag{9}$$

Table 1: Different choices of Radial Basis Functions.

RBF	$\phi(r) > 0$	$p(x)$	$m_\phi = \text{degree}(p(x))$
cubic	r^3	$b^T \cdot x + a$	1
thin plate spline	$r^2 \log r$	$b^T \cdot x + a$	1
linear	r	a	0
multiquadric	$(r^2 + \gamma^2)^{\frac{1}{2}}, \gamma > 0$	a	0
inverse multiquadric	$1/(r^2 + \gamma^2)^{\frac{1}{2}}, \gamma > 0$	a	0
Gaussian	$\exp(-\gamma r^2), \gamma > 0$	$\{0\}$	-1

$\mu_n(y)$ is the coefficient corresponding to y of the radial basis interpolation function solution $l_n(y, x)$ that satisfies $l_n(y, x_i) = 0$, $i = 1, \ldots, n$ and $l_n(y, y) = 1$. $\mu_n(y)$ can be computed as follows. Φ is extended to

$$\Phi_y = \begin{pmatrix} \Phi & \phi_y \\ \phi_y^T & 0 \end{pmatrix}, \tag{10}$$

where $(\phi_y)_i = \phi(\|y - x_i\|_2)$, $i = 1, \ldots, n$, and P is extended to

$$P_y = \begin{pmatrix} P \\ y^T \quad 1 \end{pmatrix}. \tag{11}$$

Then $\mu_n(y)$ is the $(n+1)$-th component of $v \in \mathbb{R}^{n+d+2}$ that solves the system

$$\begin{pmatrix} \Phi_y & P_y \\ P_y^T & 0 \end{pmatrix} v = \begin{pmatrix} 0_n \\ 1 \\ 0_{d+1} \end{pmatrix}. \tag{12}$$

The notations 0_n and 0_{d+1} are used for column vectors with all entries equal to zero and with dimension n and $(d+1)$, respectively. The computation of $\mu_n(y)$ is done for many different y when minimizing $g_n(y)$. This requires $O(n^3)$ operations if not exploiting the structure of Φ_y and P_y. Hence, it does not make sense to solve the full system each time. A better alternative is to factorize the matrix Φ and then use this stored factorization to speed up the factorization of the matrix on the left hand side of equation 12. An algorithm that requires $O(n^2)$ operations is described in [2].

Note that μ_n and g_n are not defined at $x_1, \ldots, x_n$ and

$$\lim_{y \to x_i} \mu_n(y) = \infty, \quad i = 1, \ldots, n. \tag{13}$$

This will cause problems when μ_n is evaluated at a point close to one of the known points. The function $h_n(x)$ defined by

$$h_n(x) \;=\; \begin{cases} \frac{1}{g_n(x)}, & x \notin \{x_1, \ldots, x_n\} \\ 0, & x \in \{x_1, \ldots, x_n\} \end{cases} \tag{14}$$

is differentiable everywhere on Ω, and is thus a better choice as an objective function. Instead of minimizing $g_n(y)$ in (8), Gutmann [4] suggests to minimize $-h_n(y)$.

The basic RBF algorithm has been discussed in detail in [2, 4] and in the form below in [7]. A discussion on how to expand the RBF algorithm to treat mixed-integer nonlinear (MINLP) problems follows.

In order to handle possible infeasibility due to the linear and nonlinear constraints not being fulfilled, define the following L_1 type merit function

$$\min_x \quad F_{L_1}(x) = f(x) + h_{L_1}(x), \tag{15}$$

where

$$\begin{aligned} h_{L_1}(x) = \quad & \sum_j \max\left(0,\ Ax^j - b_U^j - \epsilon_A,\ \ b_L^j - Ax^j - \epsilon_A\right) + \\ & \sum_j \max\left(0,\ c^j(x) - c_U^j - \epsilon_C,\ \ c_L^j - c^j(x) - \epsilon_C\right). \end{aligned} \tag{16}$$

The linear feasibility tolerance ϵ_A and the nonlinear feasibility tolerance ϵ_C are directly deducted when computing $h_{L_1}(x)$, which means that any numerically feasible point fulfills $h_{L_1}(x) = 0$ and $f(x) = F_{L_1}(x)$. In the Algorithm RBF for MINLP given below, the flag Feasible is used to track if the algorithm has found any feasible point or not. Note that while $h_{L_1}(x)$ is scale dependent it does not influence the behavior of the algorithm RBF for MINLP given below, nor the ARBF algorithm in the next section. This is due to the fact that $h_{L_1}(x)$ is not used in building the interpolation surface, and it is not used in the subproblem solutions. To compute the first RBF interpolation surface, at least $n \geq d + 1$ disjunct sample points are needed. This set is normally found by using a statistical experimental design algorithm, e.g. Latin Hypercube (McKay et al. [11]) or by evaluating some or all the corners of the box defined by Ω. For a constrained or mixed-integer nonlinear problem, the RBF interpolation needs to be a good approximation of $f(x)$ in Ω_C. Assuming that the constraints $c(x)$ are much less time-consuming to compute than the costly $f(x)$, one can try to find a large number of sample points using Latin Hypercube design, then compute $h_{L_1}(x)$ for each of the points and select the first n feasible points found as the initial experimental design. Define this strategy as a *Constrained Latin Hypercube (CLH)* design. In case enough feasible points can not be found, some of the infeasible points are added to get n points. Results using the new CLH method to generate the initial experimental design are compared to standard experimental designs in Section 5. Before turning to the new Adaptive RBF algorithm in the next section, the basic RBF algorithm, expanded to handle MINLP, is given below. This algorithm has been implemented in the TOMLAB solver *rbfSolve* since 2004 and discussed in several conference talks, but not been presented in great detail before.

Algorithm RBF for MINLP:

- Find initial set of $n \geq d + 1$ sample points $x_i \in \Omega_C$ using CLH or $x_i \in \Omega$ using any other experimental design method.
- Compute the n costly function values $f(x_i),\ i = 1, \ldots, n$ and the (non-costly) nonlinear constraints $c(x_i),\ i = 1, \ldots, n$. If using CLH, $c(x_i)$ are already computed.
- Compute $h_{L_1}(x_i), i = 1, \ldots, n$ and $F_{L_1}(x_i), i = 1, \ldots, n$.
- if for any i, $f(x_i) = F_{L_1}(x_i)$ is true, set Feasible = 1, otherwise Feasible = 0.
- if Feasible, find the feasible point with the lowest function value (x_{Min}, f_{Min}) by computing $f_{Min}(x_{Min}) = \min_{i=1,\ldots,n,\ x_i \in \Omega_C} f(x_i)$.
  Otherwise, when no feasible point exists, set $f_{Min}(x_{Min}) = \min_{i=1,\ldots,n} F_{L_1}(x_i)$.

- As an approximation of the function $f(x)$, $x \in \Omega_C$, use the n sample points, $(x_i, f(x_i))$, $x_i \in \Omega$, to build a smooth RBF interpolation model $s_n(x)$ (surrogate model, response surface model) with chosen ϕ and $m \geq m_\phi$ from Table 1.
- **Iteration** until $n \geq n_{Max}$, or a prescribed maximal CPU time (or f_{Goal}, known goal for $f(x)$, achieved with a certain relative tolerance with $x_{Min} \in \Omega_C$).
  1. Find global minimum of the constrained RBF surface, $s_n(x_{s_n}) = \min_{x \in \Omega_C} s_n(x)$[1].
  2. In every iteration in sequence pick one of the $N + 2$ cycle step choices.
     (a) **Cycle step** -1 **(InfStep).**
     Set target value $f_n^* = -\infty$, i.e. solve the global optimization problem
     $$g_n^\infty(x_{g_n}^\infty) = \min_{x \in \Omega_C \setminus \{x_1, \ldots, x_n\}} \mu_n(x), \tag{17}$$
     where $\mu_n(x)$ is computed as described in equation (12). Set $x_{n+1} = x_{g_n}^\infty$.
     (b) **Cycle step** $k = 0, 1, \ldots, N-1$ **(Global search).**
     Define target value $f_n^* \in (-\infty, s_n(x_{s_n})]$ as
     $$f_n^*(k) = s_n(x_{s_n}) - w_k \cdot \left(\max_i f(x_i) - s_n(x_{s_n}) \right),$$
     with $w_k = (1 - k/N)^2$ or $w_k = 1 - k/N$. Solve the global optimization problem
     $$g_n(x_{g_n}^k) = \min_{x \in \Omega_C \setminus \{x_1, \ldots, x_n\}} (-1)^{m_\phi + 1} \mu_n(x) \left[s_n(x) - f_n^*(k) \right]^2 \tag{18}$$
     and set $x_{n+1} = x_{g_n}^k$.
     (c) **Cycle step** N **(Local search).**
     If $s_n(x_{s_n}) < f_{Min} - 10^{-6}|f_{Min}|$, accept x_{s_n} as new search point x_{n+1}. Otherwise set $f_n^*(k) = f_{Min} - 10^{-2}|f_{Min}|$, solve (18) and set $x_{n+1} = x_{g_n}^k$.
  3. If x_{n+1} is not too close to $x_1, \ldots, x_n$, accept x_{n+1} as search point and evaluate $f(x_{n+1})$ and $F_{L_1}(x_{n+1})$.
  4. If $x_{n+1} \in \Omega_C$, set Feasible $= 1$.
  5. Update the point with lowest function value (x_{Min}, f_{Min}): either if Feasible and $f(x_{n+1}) < f_{Min}$ or if not Feasible and $F_{L_1}(x_{n+1}) < f_{Min}$.
  6. Increase n and compute new RBF surface. Start new **Iteration** step.

The InfStep in 2a) is optional, since for most problems, it does not improve the convergence to the global optimum. However, the coefficient $\mu_n(x)$ is always needed in the Global search step, and sometimes in the Local search step as well. Note that Gutmann [4] only considers one special case of the algorithm in which InfStep among others are not included.

The range $\max_i f(x_i) - s_n(x_{s_n})$ in Step 2b) must always be sufficiently positive. In the case of an initial design with an almost flat surface, or with only infeasible initial points, the range could become too small, zero, or negative. Therefore, the implementation in *rbfSolve* safeguards the computation against this issue.

[1] Note that any nonlinear constraints are explicitly used in this subproblem. This is the reason the nonlinear constraints need to be cheap.

The range may also, for many problems, become too big and lead to unreasonably low target values. Gutmann suggests replacing $f(x_i) > median_i\, f(x_i)$ with $median_i\, f(x_i)$ both when computing the range and in the RBF interpolation. In practice one commonly needs to use some strategy to reduce the range. When large values are replaced by the median in the RBF interpolation, many numerical interpolation problems are avoided, but when additional points are sampled close to a stationary point, the function approximation gets less and less accurate in other parts of the space.

The RBF algorithm in practice is very sensitive to the choice of initial experimental design, especially when using stochastic designs. If the initial steps of the algorithm fail to find some point in the basin of the global optimum, it often starts iterating repeatedly with sample points on the boundaries in the Global search, and only refines a local minima in the Local search.

Define the number of active variables $\alpha(x)$ as the number of elements of x that have components close to the bounds in the box, i.e.

$$\alpha(x) = |\{j \in 1, \ldots, d, j \notin I : |x^j - x_L^j| \leq \epsilon_x \text{ or } |x^j - x_U^j| \leq \epsilon_x\}|. \tag{19}$$

If a point is interior, then obviously $\alpha(x) = 0$. The integer components of x are not considered when determining if a point is interior or not. If studying $\alpha(x)$ during the iterations when running *rbfSolve* for many problems, the algorithm frequently generates points with some components on their bounds, $\alpha(x) > 0$. Doing a systematic study of the solution of (18) for many $f_n^* \in (-\infty, s_n(x_{s_n})]$ on different subproblems during the RBF iterations confirmed that the solution is typically not interior, and hence a careful choice of target value is needed. Solving a large set of problems (18) for different target values should generally be much less time-consuming than computing the costly $f(x)$. In addition computations for different target values are independent and could be done in parallel on different CPUs. By examining solutions for a large set of target values, it should be possible to find good search points in most iteration steps, and only evaluate the costly $f(x)$ for these points. In the next section a new adaptive RBF algorithm suitable for parallel implementation is formulated.

3 The Adaptive Radial Basis Algorithm (ARBF) for MINLP

In this section the main ideas of the new Adaptive Radial Basis Algorithm are discussed and a formalized description is provided. To overcome the limitations of the RBF algorithm, the choice of target values must be made more flexible. The objective function class is very wide and a robust algorithm must adapt to the particular behavior of a function. A few choices of target values based on the function value range as in the RBF algorithm only works for nice well-behaved problems. This observation has been confirmed by practical experience with the RBF algorithm for a large set of real-life user problems over the past six years. Instead, a more adaptive algorithm is proposed, based on evaluating a large set of target values in each iteration is proposed. The approach is similar to two of the algorithms proposed by Jones in [9] to solve kriging problems, named the *Enhanced Method 4* and *Method 7*.

Jones considers several kriging algorithms, e.g. *Method 4*, where the problem in each iteration is to maximize the probability of improvement after setting a target value. The optimal solution found is used as the new search point, and the costly $f(x)$ is evaluated for this search point and a new surrogate model of kriging type is computed. As in the RBF algorithm, it is a major difficulty to set the target value properly in each iteration. To overcome this problem, Jones proposes a new method called the *Enhanced Method 4* that uses a range of target values in each iteration, corresponding to low, medium and high desired improvement. Similarly, in the ARBF algorithm, a set of target values are selected each iteration. However, experience has demonstrated the need to cover the full range from $-\infty$. For each target value the global optimization problem defined by (18) is solved.

Evaluating a large set of target values leads to many candidate points. If all were used, it would lead to several costly function evaluations in each iteration. Jones shows on one-dimensional examples that the optimal solutions tend to cluster in different areas of the parameter space. Similar behavior has been observed for the solutions of (18) with different target values. It is hence natural to apply a clustering algorithm to the set of optimal points $\{\hat{x}_j\}_{j=1}^{M}$ (transformed to the unit cube $[0,1]^d$), and use only one or a few points from each group found. As described below, Jones suggests applying a tailor-made clustering algorithm to the sequence of optimal points in decreasing target value order. The algorithm has been modified by adding steps 7 and 8 later in this section. In the test in step 8 the number of components on bounds is used, defined as in (19). Compared to Holmström [7], the algorithm has also been updated to handle mixed-integer problems by separating treatment of integer and continuous variables, and in addition step 1 is new.

The Jones Cluster Algorithm for MINLP

- Transform the set of optimal points $\{\hat{x}_j\}$ to the unit cube $[0,1]^d$, and compute the distance between two successive optimal points as

$$\Delta_j = \sqrt{\sum_{l=1,\ldots,d, l\notin I} (\hat{x}_j^l - \hat{x}_{j+1}^l)^2/(d-|I)|}.$$

- Compute the number of integer components that are different between two successive solutions as $\Delta_j^I = |\{l \in I : \hat{x}_j^l \neq \hat{x}_{j+1}^l\}|$.
- Assign point 1, $\hat{x}_1$, to group 1.
- Sequentially consider point 2 to M. For each point, if a criterion $C > 12$, a new group is started. The criterion C is computed as follows:
  1. If $\Delta_{j-1}^I > 0$ then set $C = 100$, i.e. start a new group. Then at least one integer component has changed.
  2. If $\Delta_j > 0.1$ and $\Delta_{j-1} > 0.1$ then set $C = 100$, i.e. start a new group.
  3. Otherwise, if $\Delta_j > 0.0005$, then set $C = \Delta_{j-1}/\Delta_j$.
  4. Otherwise, if $j \geq 3$ and $\Delta_{j-1} > 0.0005$, then set $C = \Delta_{j-1}/\max(\Delta_{j-2}, 0.0005)$.
  5. Otherwise, if $j = 2$ and $\Delta_1 > 0.1$ and $\Delta_2 < 0.0005$, then set $C = 100$ to signal the need for a new group.
  6. If none of the above conditions is satisfied, set $C = 0$, i.e. no need to start a new group unless any of the following two criteria are fulfilled.

7. If $j = M$ and $C < 12$ and $\Delta_{M-1} > 0.1$, set $C = 100$, i.e. start a new group for the last point.
8. If $C < 12$ and $\Delta_{j-1} > 0.1$ and $\alpha(\hat{x}_j) = \alpha(\hat{x}_{j-1})$, check if any of the components on the bounds for point $\hat{x}_j$ have at least a 10% difference in the corresponding components in $\hat{x}_{j-1}$. Also test if any of the components on the bounds for point $\hat{x}_{j-1}$ have at least a 10% change in the corresponding components in $\hat{x}_j$. If any of the tests are true, start a new group by setting $C = C + 200$.

Let $f_{Min} = \min_i f(x_i)$ and $f_{Max} = \max_i f(x_i)$. Jones suggests setting the target values using a fixed grid as $f_n^*(j) = s_n(x_{s_n}) - w_j \cdot f_\Delta$, where the range is set to $f_\Delta = f_{Max} - f_{Min}$. The two first rows in Table 2 show the choice of target value factors w_j. In the new algorithm, two extreme values shown in row three have been added. It is then easier to detect if the range of target values is sufficient.

Table 2: Weight factors w_j used in the global grid search

0.0	10^{-4}	10^{-3}	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.10	0.11	0.12	0.13	0.15	0.20	0.25	0.30	0.40	0.50	0.75	1.00
1.50	2.00	3.00	100	∞							

The above choice of f_Δ might become too big if the function varies over a large range. In such cases, as a fixed grid is used, there may be failure to sample important target values that would lead to the region of the global minimum. Since the range can only get larger as the iterations proceed, the algorithm is unlikely to sample these target values in later iterations.

To be more flexible and adaptive, the target values are set as $f_n^*(j) = s_n(x_{s_n}) - \beta \cdot w_j \cdot f_\Delta$, where β is an adaptive factor and the range is

$$f_\Delta = \begin{cases} \min(\max(1, f_{Min}), f_{Max} - f_{Min}), & \text{if } f_{Min} > 0 \\ \min(10 \cdot \max(1, |f_{Min}|), f_{Max} - f_{Min}), & \text{if } f_{Min} \leq 0. \end{cases} \tag{20}$$

If several optimal solutions for different target values are equal or very close, it might be a sign that the target values are too close, so β is iteratively increased by a factor 10 If the optimal point found for the second target value is far from the solution of the first target value, i.e. the minimum of the RBF surface, it is a sign that the target values are too spread out, and β is likewise decreased by a factor 10. In most cases this happens close to a stationary point.

As in the original RBF algorithm, every iteration of the ARBF algorithm starts by finding the global minimum of the RBF surface with respect to all constraints by solving

$$\hat{s}_n := s_n(x_{s_n}) = \min_{x \in \Omega_C} s_n(x).$$

If $\hat{s}_n \ll f_{Min}$ the RBF surface is fluctuating wildly and there is no point in applying a target value strategy. The target values need to be even lower than $\hat{s}_n$, so they might as well be set much lower than the actual global minimum. Applying the target value strategy in such cases generally produces garbage solution points. Instead, the

minimum of the RBF surface is added as a new point repeatedly until the interpolation stabilizes and more reasonable target values can be set. If $\hat{s}_n$ is closer to f_{Min} and the oscillation of the surface is less pronounced, the question is when to rely on the target value strategy. The approach taken is to consider the RBF surface wildy fluctating, i.e. applying the above strategy, down to a relative difference of 10%. This works well in tests so far, but other values could be tried. Currently the following test is used: If $\hat{s}_n < f_{Min} - 0.1|f_{Min}|$ when $f_{Min} \neq 0$, or $\hat{s}_n < f_{Min} - 10v$ when $f_{Min} = 0$, then $\hat{s}_n \ll f_{Min}$ is considered true. v is computed as $v = \min(f(x)), x \in \{x : f(x) > 10^{-7}\}$. For the special case when the set is empty, $v = 10^{-7}$ is used.

For every ARBF iteration, the algorithm is in one of three modes: the *wild mode* described in the previous paragraph, a *global grid search mode*, or a *local grid search* mode. In the global grid search the aim is to sample one or more points from every region of interest. In the local grid search the aim is to find a better approximation of any stationary points close to the best point found so far. Ideally one of these stationary points is also the global minimum. Note that the global grid search also sample points close to the best point found similar to the local grid search. In the wild mode the aim is to proceed with surface minimum points until the interpolation is stable enough to make a global target value grid give reasonable results. The wild mode is entered automatically when the surface is fluctuating wildly, but the switch between global and local grid mode is determined by the algorithm in the following way: Start with global grid mode, and as long as this gives function value reductions, stay in that mode. Every iteration of the global and local grid mode always end by adding the minimum point of the surface (one S-step). This is taken care of by the flag $EndGridMode$. When no reduction is achieved in an iteration of the global mode (or possibly only in the final surface minimum S-step sampling) the algorithm switches to local mode. The same logic applies for local mode; it continues the local grid search until no reductions are achieved, and then switches to global mode. In both the local and global grid mode, one or more points might be selected using the cluster algorithm and some heuristic rules (discussed later in this section). The formal ARBF algorithm description can now be given.

Algorithm ARBF for MINLP:

- Find initial set of $n \geq d + 1$ sample points $x_i \in \Omega_C$ using CLH or $x_i \in \Omega$ using any other experimental design method.
- Compute the n costly function values $f(x_i),\ i = 1, \ldots, n$ and the (non-costly) nonlinear constraints $c(x_i),\ i = 1, \ldots, n$. If using CLH, $c(x_i)$ are already computed.
- Compute $h_{L_1}(x_i), i = 1, \ldots, n$ and $F_{L_1}(x_i), i = 1, \ldots, n$.
- If for any i, $f(x_i) = F_{L_1}(x_i)$ is true, set Feasible = 1, otherwise Feasible = 0.
- if Feasible, find the feasible point with the lowest function value (x_{Min}, f_{Min}) by computing $f_{Min}(x_{Min}) = \min_{i=1,\ldots,n,\ x_i \in \Omega_C} f(x_i)$.
  Otherwise, when no feasible point exists, set $f_{Min}(x_{Min}) = \min_{i=1,\ldots,n} F_{L_1}(x_i)$.
- As an approximation of the function $f(x),\ x \in \Omega_C$, use the n sample points, $(x_i, f(x_i)),\ x_i \in \Omega$, to build a smooth RBF interpolation model $s_n(x)$ with chosen ϕ and $m \geq m_\phi$ from Table 1.
- Set $GlobalProgress = 1$ and $LocalProgress = 0$, making the initial search mode global. Also initialize $EndGridMode = 0$.

- **Iteration** until $n \geq n_{Max}$, or a prescribed maximal CPU time (or f_{Goal}, known goal for $f(x)$, achieved with a certain relative tolerance at $x_{Min} \in \Omega_C$).

  1. Find the global minimum of the RBF surface, $s_n(x_{s_n}) = \min_{x \in \Omega_C} s_n(x)$.
  2. Find a set of new search points $X = \{\bar{x}_j, j = 1, \ldots, k\}$ by applying one of the following three types of search procedures dependent on logical conditions given for each procedure.
     (a) **Wild Mode (S-step).** If $s_n(x_{s_n}) \ll f_{Min}$ or $EndGridMode = 1$, accept the RBF surface minimum x_{s_n} as the new search point, i.e. $X = x_{s_n}$. Set $EndGridMode = 0$.
     (b) **Global Grid Mode. (G-step).** If $GlobalProgress = 1$, define M target values $f_n^* \in (-\infty, s_n(x_{s_n})]$ as $f_n^*(j) = s_n(x_{s_n}) - \beta \cdot w_j \cdot f_\Delta$ with $w_j,\ j = 1, \ldots, M$, a vector of predefined factors in the range $[0, \infty]$, and β an adaptive weight factor in the range $[10^{-3}, 10^3]$, initialized as $\beta = 1$. The function range f_Δ is determined in each step as described in (20).
     For each of the M target values, solve the global optimization problem
     $$g_n(\hat{x}_j) \ = \ \min_{x \in \Omega_C \setminus \{x_1, \ldots, x_n\}} (-1)^{m_\phi + 1} \mu_n(x) \left[s_n(x) - f_n^*(j) \right]^2 \qquad (21)$$
     Then use the Jones Clustering Algorithm on the M optimal solution points $\hat{x}_j$. Apply heuristic rules to determine which of the clustered groups to consider, and in each selected group, which of the points to include in the new set of search points X; see the *Point Selection Algorithm* later in this section. Set $EndGridMode = 1$.
     (c) **Local Grid Mode. (L-step).** If $LocalProgress = 1$, define M_L target values using the same factors w_j as in the G-step together with some additional small factors.
     Solve (21) and apply the Jones Clustering Algorithm to the M_L optimal solutions $\hat{x}_j$.
     Apply the heuristic rules described in the *Point Selection Algorithm* to determine which points in the first cluster group should be included in set X. Set $EndGridMode = 1$.
  3. Check the set of new search points $X = \{\bar{x}_j, j = 1, \ldots, k\}$, deleting any point too close (normalized distance less than 10^{-4}) to any previous point in X; or too close to any sample point $x_1, \ldots, x_n$ (distance less than 10^{-8}).
  4. Set $x_{n+j} = \bar{x}_j, j = 1, \ldots, k$ and evaluate $f(x_{n+j}),\ c(x_{n+j}),\ j = 1, \ldots, k$ and $F_{L_1}(x_{n+j}),\ j = 1, \ldots, k$
  5. If any $x_{n+j} \in \Omega_C,\ j = 1, \ldots, k$, set Feasible = 1.
  6. If Feasible and $\min_{j=1,\ldots,k} f(x_{n+j}) < f_{Min}$ or
     if not Feasible and $\min_{j=1,\ldots,k} F_{L_1}(x_{n+j}) < f_{Min}$
     - Update the point with lowest function value (x_{Min}, f_{Min}).
     - Set $LocalProgress = 1$ (if L-step).
     - Set $GlobalProgress = 1$ (if G-step).

     else
     - Set $LocalProgress = 0$ (if L-step).

- Set $GlobalProgress = 0$ (if G-step).

7. Increase n by k; compute new RBF surface. Start new **Iteration** step.

The selection of trial points utilizing the result of the clustering process applied to the set of optimal solutions computed from the target values is one of the heuristics. For kriging algorithms, Jones suggests picking the last member of each of the groups formed by the clustering algorithm as a new candidate point, i.e. the one with the smallest target value in each group. This selection criteria has been found a bit crude when applied to RBF algorithms. Therefore a *Point Selection Algorithm* has been developed, which describes how to generate new trial points based on the results from the Jones Cluster Algorithm. The algorithm is described in detail in Holmström [7].

The main idea is to only select points from the cluster groups with least number of components on bounds, found by computing $\alpha(x)$ in (19) for every optimal solution. The number of points selected in each group depends on the distance between the optimal points in the group. The first group, with optimal points close to the current RBF surface minimum, and the last group, with lowest target values including the $-\infty$ target value, are treated separately.

4 Implementation of the RBF and ARBF for MINLP

The MINLP algorithms described are available in MATLAB using the TOMLAB Optimization Environment (http://tomopt.com/tomlab/). The Algorithm ARBF for MINLP in Section 3 is implemented in the TOMLAB solver *ARBFMIP*, while the Algorithm RBF for MINLP in Section 2 is found in the solver *rbfSolve*. Similar ideas were used to implement a MINLP version of the Efficient Global Optimization (EGO) method described by Jones et al. [10]. All three solvers are part of the TOMLAB/CGO toolbox for costly nonconvex black-box mixed-integer optimization.

The implementations rely on robust solutions of the non-costly global MINLP optimization problems described as part of the algorithms, equations (3), (18), (17) and (21). Subsolvers are required for all three costly MINLP solvers. Any standard MINLP solver in TOMLAB can be used; currently there are nine choices. In order to make the CGO solvers more robust, if the subsolver returns an infeasible solution to the MINLP subproblem, one or two alternative solvers will try to find a feasible solution.

By default and in the numerical tests, the global MINLP solver *glcCluster* is used. It uses a mixed-integer constrained DIRECT solver (*glcDirect*, *glcFast* or *glcSolve*) as the initial step, then applies an adaptive clustering algorithm to all points sampled by the DIRECT algorithm, and finally repeats local optimization with fixed integer variables. The starting points in the local optimizations are set as the best point found in each cluster. As local solver, any nonlinear programming solver in TOMLAB is suitable; by default *NPSOL* and *SNOPT* are used. TOMLAB also has four derivative-based mixed-integer nonlinear solvers, MULTIMIN, OQNLP, MINLPBB and MISQP. If derivatives are estimated numerically, these four solvers can be used as subsolvers. In the tests, OQNLP and MULTIMIN were used as alternative solvers, whenever *glcCluster* failed to find a feasible solution for a subproblem.

5 Numerical Results

In this section the results from a set of standard MINLP test problems and a set of constrained global optimization test problems are reported. Each problem set was solved using *ARBFMIP*, *rbfSolve*, *EGO*, *OQNLP*, *MINLPBB*, *MISQP* and *GENO*.

For the black-box CGO solvers *ARBFMIP*, *rbfSolve* and *EGO*, different settings and experimental designs were evaluated to see if any combination outperforms the rest. Two types of radial basis functions were used for *ARBFMIP* and *rbfSolve*: the thin plate spline $\phi(r) = r^2 \log r$ (TPS) and the cubic spline $\phi(r) = r^3$ (Cubic). Three different experimental designs are used: The standard Latin Hypercube (LH), the Constrained Latin Hypercube (CLH) as defined in Section 2 and a corner strategy we denote LAC. LAC picks the lower left corner and its d adjacent corners, i.e. the following $d+1$ corners of Ω:

$$\{x_L, x_L + e_i, i = 1, \ldots, d, \text{ with } e_i = 0 \in R^d, \text{ except component j} : e_i^j = x_U^j - x_L^j\}.$$

In the tests the midpoint of the box, $(x_L + x_U)/2$, is always added as point $d+2$ in this design strategy. In the tests of the CLH design, the minimal number of points, $n = d + 1$, was used. Slightly more robust results would probably be obtained if n were increased somewhat. In all runs with the CGO solvers, the maximum number of function evaluations were set to 250 ($n_{Max} = 250$ in Algorithm *RBF* and *ARBF* for MINLP).

In the tables, row *ExD* gives the experimental design used, row *RBF* gives the radial basis function used, and in row *Scale* the On/Off switch indicates whether variable scaling to the unit cube was used or not. For the MINLP problems, scaling was not used. Column *IP* specifies the number of initial points from the experimental design. For *rbfSolve* the option to replace all function values $f(x_i) > median_i f(x_i)$ with the $median_i f(x_i)$ is used if row *Repl* is set to *Yes*. The function values are replaced both when computing the range and in the RBF interpolation. In *ARBFMIP* this choice of replacement strategy is avoided. In each iteration, for the radial basis interpolation, all function values in the range $[f_{Min}, \max(0, f_{Min}) + 10^5]$ are untouched and values $f(x) > \max(0, f_{Min}) + 10^5$ are replaced by $\max(0, f_{Min}) + 10^5 + \log_{10}(f(x) - \max(0, f_{Min}) - 10^5)$. Thereby, huge scale differences in the linear equation system (5) are avoided. The other possible source of numerical difficulties in solving (5) is a very badly scaled domain $[x_L, x_U]$. Such a problem is easily avoided by using the scale option described above.

The solvers *OQNLP*, *MINLPBB* and *MISQP* are derivative-based MINLP solvers, but were forced to estimate derivatives numerically in the experiments. It is easy to derive analytical derivatives for the test problems, but the aim is to test the ability to solve black-box problems. *OQNLP* implements a stochastic algorithm to find starting points, and the random point generation depends on a seed parameter. In the tests only one seed parameter value was set. The other two solvers are deterministic, and the result is only dependent on the starting point given. In the tests, each problem is solved with 100 random starting points created inside the box defined by the simple bounds.

The *GENO* solver implements a constrained mixed-integer nonlinear stochastic genetic algorithm. This type of algorithm is popular in practice, although many function evaluations are needed to reach a global minimum.

Each test problem was solved with 50 random starting points (using the same starting points as in the experiments with *OQNLP*, *MINLPBB* and *MISQP*, but only the first 50). It is possible to specify a random seed in *GENO*. The result of a run with a genetic algorithm is highly dependent on the set of random points generated in the run. Therefore, each problem was solved using 5 different seeds. Thus, each problem was solved 250 times. A time limit was defined for all runs with *GENO*, in order to avoid excessive run-times. The limit was set so that *GENO* could perform at least 10000 function evaluations. Most of the runs do not find any good solutions.

Throughout this section, all tables come in pairs and present the number of function evaluations needed to achieve a function value with relative error of 1% and 0.01% respectively. The relative error is defined as

$$E = \frac{f_{Min} - f_{global}}{|f_{global}|}, \tag{22}$$

where f_{Min} is the current best function value and f_{global} is the known global optimum (which is nonzero for all the problems). A $-$ sign indicates failure.

5.1 Numerical Results for mixed-integer nonlinear programs

Table 3 gives a compact description for the MINLP test functions, including the abbreviations used. Column d is the number of variables, x_I the number of integer variables, Ax the number of linear inequality constraints, Ax with $=$ on the row below the number of linear equality constraints, $c(x)$ the number of nonlinear inequality constraints and $c(x)$ with $=$ on the row below the number of nonlinear equality constraints. The Domain column shows the lower and upper bounds for all variables.

Table 3: Names and descriptions of the MINLP test problems

Problem	Abbrev.	d	x_I	Ax	Ax $=$	$c(x)$	$c(x)$ $=$	Domain
Kocis & Grossmann 1998	KG98	5	3	3	0	2	2	$[0, 10^{-8}, 0, 0, 0]$ $-$ $[10^8, 10^8, 1, 1, 1]$
Floudas 1995 6.6.5	FL95	3	1	2	0	1	0	$[0.2, -2.22554, 0]$ $-$ $[1, -1, 1]$
Pörn et al. 1997	PÖ97	2	2	3	0	1	0	$[1, 1]$ $-$ $[5, 5]$
Kocis & Grossmann 1989	KG89	4	2	1	0	4	0	$[0, 0, 0, 0]$ $-$ $[10, 20, 1, 1]$
Kesavan et al. 2004 D	KE04	5	3	3	1	1	0	$[0, 0, 0, 1, 1]$ $-$ $[1, 1, 1, 10, 6]$
Floudas-Pardalos 3.4TP3	FP1	6	2	3	0	2	0	$[0, 0, 1, 0, 1, 0]$ $-$ $[6, 6, 5, 6, 5, 10]$
Floudas-Pardalos 12.2TP1	FP2	5	3	3	0	2	2	$[0]^5$ $-$ $[1, 1, 1, 1.5, 1.6]$
Floudas-Pardalos 12.2TP3	FP3	7	4	5	0	4	0	$[0]^7$ $-$ $[1.2, 1.8, 2.5, 1, 1, 1, 1]$
Floudas-Pardalos 12.2TP4	FP4	11	8	4	0	3	3	$[0]^{11}$ $-$ $[1]^{11}$
Floudas-Pardalos 12.2TP5	FP5	2	2	3	0	1	0	$[1, 1]$ $-$ $[5, 5]$
Floudas-Pardalos 12.2TP6	FP6	2	1	2	0	1	0	$[1, 1]$ $-$ $[10, 6]$
Floudas-Pardalos 12.2TP2	FP7	3	1	2	2	1	0	$[0, 0.2, -2.22554]$ $-$ $[1, 1, -1]$

Table 4 presents the results for *rbfSolve* with different settings. The number of function values needed to get close to the optimal value with requested accuracy is very low in all cases. The results are very similar and practically independent of the parameter settings used for the two experimental designs tested, LAC and CLH. There are a few failures to converge, possibly avoided if using a larger number of initial points in the experimental design. In four cases the failures are avoided if the median replacement option is used, but this option often leads to slower convergence.

Table 4: Number of function evaluations to get within 1% and 0.01% of the optimal value for *rbfSolve* on the MINLP test problems.

	$E \leq 10^{-2}$										$E \leq 10^{-4}$							
ExD	LAC					CLH					LAC				CLH			
RBF	TPS		Cubic			TPS		Cubic			TPS		Cubic		TPS		Cubic	
Scale	Off		Off			Off		Off			Off		Off		Off		Off	
Repl	IP	Yes	No	Yes	No	IP	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No
KG98	7	9	8	9	8	6	10	10	10	10	9	8	9	8	10	10	10	10
FL95	5	8	6	8	6	4	5	5	5	5	8	6	8	6	7	5	5	5
PÖ97	3	5	5	5	5	3	5	5	5	5	5	5	5	5	5	5	5	5
KG89	6	13	8	16	11	5	11	8	8	8	19	-	-	-	14	14	17	20
KE04	7	8	8	8	8	6	8	7	8	7	8	8	8	8	8	7	8	7
FP1	8	24	-	10	19	7	24	14	21	16	24	-	10	-	24	14	21	16
FP2	7	8	8	8	8	6	7	7	7	7	8	8	8	8	7	7	7	7
FP3	9	10	10	10	10	8	-	-	-	-	10	10	10	10	-	-	-	-
FP4	13	21	21	15	15	12	23	20	20	16	21	21	15	15	23	20	20	16
FP5	3	5	5	5	5	3	5	5	5	5	5	5	5	5	5	5	5	5
FP6	4	5	5	5	5	3	4	4	4	4	5	5	5	5	4	4	4	4
FP7	5	7	7	7	7	4	7	7	7	7	-	-	-	-	-	-	-	-

Table 5 and Table 6 present the results for *ARBFMIP* and *EGO* with different settings.

The results for *ARBFMIP* are in general excellent, with a very low number of function evaluations needed to converge and few failures. The reason for most failures is that the target value search grid needs to be made more dense in the Point Selection Algorithm when close to the global optimum. This problem is easy to detect and an adaptive strategy to overcome the problem is in development. *EGO* also shows good results for the CLH design, when feasible points are obtained in the experimental design. It was not tested using LAC as initial design, since it needs more initial points to work properly.

Table 7 and Table 8 present the results for *OQNLP*, *MINLPBB* and *MISQP*. The number of failures (in %) and the mean, min and max values for the successful runs out of the total 100 for each problem are reported, as well as the number of constraint evaluations needed. For example, 166 *123* represents 166 function evaluations and 123 constraint evaluations. The deterministic solvers are more sensitive to the initial starting points. *MISQP* had two cases that did not converge, and seems less robust than the other two. The number of function evaluations needed is an order of magnitude higher than for the CGO solvers. The solvers also often had trouble obtaining higher accuracy solutions.

Table 5: Number of function evaluations to get within 1% of the optimal value for *ARBFMIP* and *EGO* on the MINLP test problems. No variable scaling was used.

Solver	ARBFMIP									EGO			
ExD	LAC			CLH			LH			CLH		LH	
RBF	IP	TPS	Cubic	IP	TPS	Cubic	IP	TPS	Cubic	IP		IP	
KG98	7	8	8	6	7	7	51	52	52	6	11	51	-
FL95	5	6	6	4	6	6	33	34	35	4	-	33	37
PÖ97	3	5	5	3	4	4	14	14	14	3	5	14	14
KG89	6	8	11	5	6	6	41	48	42	5	18	41	-
KE04	7	8	8	6	7	7	51	52	52	6	-	51	-
FP1	8	-	15	7	-	-	65	66	66	7	164	65	-
FP2	7	8	8	6	7	7	19	52	52	6	8		
FP3	9	10	10	8	13	-	65	66	66	8	62	65	-
FP4	13	19	15	12	23	18	65	102	-	12	42		
FP5	3	5	5	3	4	4	14	14	14	3	4	14	14
FP6	4	5	5	3	4	4	21	22	22	3	5	21	-
FP7	5	6	6	4	7	7	33	34	34	4	13	33	-

Table 6: Number of function evaluations to get within 0.01% of the optimal value for *ARBFMIP* and *EGO* on the MINLP test problems. No variable scaling was used.

Solver	ARBFMIP									EGO			
ExD	LAC			CLH			LH			CLH		LH	
RBF	IP	TPS	Cubic	IP	TPS	Cubic	IP	TPS	Cubic	IP		IP	
KG98	7	8	8	6	7	7	51	52	52	6	11	51	-
FL95	5	6	6	4	6	6	33	34	35	4	-	33	37
PÖ97	3	5	5	3	4	4	14	14	14	3	5	14	14
KG89	6	10	12	5	6	7	41	52	52	5	-	41	-
KE04	7	8	8	6	7	7	51	52	52	6	-	51	-
FP1	8	-	15	7	-	-	65	66	66	7	164	65	-
FP2	7	8	8	6	7	7	19	52	52	6	8		
FP3	9	10	10	8	13	-	65	66	66	8	-	65	-
FP4	13	19	15	12	23	18	65	102	-	12	42		
FP5	3	5	5	3	4	4	14	14	14	3	4	14	14
FP6	4	5	5	3	4	4	21	22	22	3	5	21	-
FP7	5	-	-	4	-	-	33	-	-	4	13	33	-

Table 7: Number of function evaluations to get within 1% of the optimal value for *OQNLP*, *MINLPBB* and *MISQP* on the MINLP test problems.

Solver	%	OQNLP			%	MINLPBB			%	MISQP		
	Fail	mean	min	max	Fail	mean	min	max	Fail	mean	min	max
KG98	75	7529 *5943*	166 *124*	26335 *19247*	50	119 *61*	119 *61*	119 *61*	53	199 *177*	91 *81*	244 *217*
FL95	0	171 *158*	18 *12*	297 *279*	0	137 *95*	121 *81*	138 *96*	48	25 *25*	6 *6*	41 *41*
PÖ97	0	8 *8*	1 *1*	8 *8*	0	52 *34*	1 *1*	80 *60*	20	10 *10*	1 *1*	19 *19*
KG89	0	674 *499*	19 *12*	2882 *961*	0	3329 *2662*	1887 *1533*	4808 *3820*	100	-	-	-
KE04	0	5430 *3808*	30 *18*	21867 *15378*	0	539 *160*	45 *7*	1194 *387*	63	24 *16*	10 *7*	64 *43*
FP1	1	2088 *1547*	290 *259*	8621 *6325*	92	110 *52*	53 *8*	205 *72*	87	45 *45*	31 *31*	59 *59*
FP2	35	4595 *2751*	27 *15*	44342 *26287*	46	119 *96*	77 *42*	158 *211*	56	40 *40*	19 *19*	127 *127*
FP3	0	645 *501*	50 *26*	6576 *5341*	0	319 *222*	263 *199*	328 *262*	98	58 *58*	54 *54*	61 *61*
FP4	13	24059 *8612*	76 *32*	95500 *36191*	0	802 *690*	298 *30*	1188 *1169*	100	-	-	-
FP5	0	8 *8*	1 *1*	8 *8*	0	73 *47*	1 *1*	197 *141*	12	14 *14*	1 *1*	23 *23*
FP6	0	111 *102*	9 *5*	251 *237*	1	81 *57*	23 *12*	128 *91*	58	21 *21*	5 *5*	113 *113*
FP7	0	142 *133*	16 *10*	290 *277*	0	151 *100*	148 *96*	151 *100*	40	25 *25*	6 *6*	41 *41*

Table 8: Number of function evaluations to get within 0.01% of the optimal value for *OQNLP*, *MINLPBB* and *MISQP* on the MINLP test problems.

Solver	%	OQNLP			%	MINLPBB			%	MISQP		
	Fail	mean	min	max	Fail	mean	min	max	Fail	mean	min	max
KG98	75	7529 *5943*	166 *124*	26335 *19247*	50	119 *61*	119 *61*	119 *61*	53	199 *177*	91 *81*	244 *217*
FL95	0	178 *165*	18 *12*	355 *337*	0	154 *110*	138 *96*	155 *111*	50	29 *29*	6 *6*	46 *46*
PÖ97	0	8 *8*	1 *1*	8 *8*	0	64 *44*	1 *1*	116 *83*	20	10 *10*	1 *1*	19 *19*
KG89	0	-	-	-	0	-	-	-	100	-	-	-
KE04	0	22633 *16117*	44 *29*	63230 *44518*	0	555 *164*	58 *10*	1231 *396*	63	24 *16*	10 *7*	64 *43*
FP1	1	2088 *1548*	291 *259*	8622 *6326*	92	110 *52*	53 *8*	205 *72*	87	47 *47*	31 *31*	68 *68*
FP2	35	4595 *2751*	27 *15*	44342 *26287*	46	119 *96*	77 *42*	158 *211*	56	40 *40*	19 *19*	127 *127*
FP3	0	1018 *772*	52 *28*	6587 *5348*	0	777 *662*	721 *639*	786 *702*	100	-	-	-
FP4	13	24059 *8612*	76 *32*	95500 *36191*	0	4571 *4417*	3924 *3731*	5255 *5291*	100	-	-	-
FP5	0	8 *8*	1 *1*	8 *8*	0	219 *153*	1 *1*	248 *176*	12	14 *14*	1 *1*	23 *23*
FP6	0	238 *228*	11 *7*	257 *245*	1	84 *60*	23 *12*	132 *94*	58	22 *22*	5 *5*	113 *113*
FP7	100	-	-	-	100	-	-	-	100	-	-	-

Table 9 presents the results for the *GENO* solver running each problem $50 \times 5 = 250$ times. Almost every problem fails to be solved within 10000 function evaluations and the range of failures is between 84 and 100 percent. A stochastic genetic algorithm solver like *GENO* is clearly not suitable for costly black-box MINLP problems.

Table 9: Number of function evaluations to get within 1% and 0.01% of the optimal value for *GENO* on the MINLP test problems.

	%	$E \leq 10^{-2}$			%	$E \leq 10^{-4}$		
	Fail	mean	min	max	Fail	mean	min	max
KG98	100	-	-	-	100	-	-	-
FL95	94	5761 *3048*	3711 *844*	7785 *4588*	94	11831 *7186*	9569 *5902*	14810 *8412*
PÖ97	100	-	-	-	100	-	-	-
KG89	99	6529 *4614*	5459 *3261*	7598 *5967*	100	-	-	-
KE04	100	-	-	-	100	-	-	-
FP1	100	-	-	-	100	-	-	-
FP2	94	9094 *5550*	8538 *5028*	10165 *6005*	94	9094 *5550*	8538 *5028*	10165 *6005*
FP3	96	5343 *3277*	4398 *2765*	5818 *4029*	100	-	-	-
FP4	100	-	-	-	100	-	-	-
FP5	100	-	-	-	100	-	-	-
FP6	100	-	-	-	100	-	-	-
FP7	84	6538 *3229*	1977 *907*	8858 *5403*	100	-	-	-

5.2 Numerical Results for Constrained Global Optimization Problems

In this section, evaluations performed on a set of Constrained global optimization test problems are presented. Table 10 gives the names of the problems and the abbreviations used and Table 11 gives a compact description of the test functions with the same notation as in Table 3.

Table 12 and Table 13 present the results for *rbfSolve*. In this case runs with and without variable scaling to the unit cube are reported. As for the MINLP problems, the results are in most cases very good.

Table 14 and Table 15 present the results for *ARBFMIP* for all three experimental designs. Results are excellent, similar to the results for the MINLP problems in Section 5.1. Comparing the results for the CLH and LH experimental designs, clearly much fewer function evaluations are needed using the CLH design and more general failures are avoided. Creating an experimental design with feasible points, as in the CLH design method, seems to be advantageous and helps the progress of RBF-type algorithms. Figures 1 and 2 show a comparison between the results for *ARBFMIP* and *rbfSolve* for the CLH and LAC initial designs. It is easy to see that *ARBFMIP* produces improved results since most points cluster in the right side of the diagrams.

Table 10: Names and abbreviations for the constrained global optimization test problems

Abbrev	Problem Name	Abbrev	Problem Name	Abbrev	Problem Name
P1	Gomez 2	P7	Schittkowski 234	P13	Schittkowski 343
P2	Gomez 3	P8	Schittkowski 236	P14	Floudas-Pardalos 3.2 TP 1
P3	Hock-Schittkowski 59	P9	Schittkowski 237	P15	Floudas-Pardalos 3.3 TP 2
P4	Hock-Schittkowski 65	P10	Schittkowski 239	P17	Floudas-Pardalos 3.5 TP 4
P5	Hock-Schittkowski 104	P11	Schittkowski 330	P18	Floudas-Pardalos 4.10 TP 9
P6	Hock-Schittkowski 105	P12	Schittkowski 332	P28	Zimmerman

Table 11: Description of the constrained global optimization test problems

Problem Nr.	d	Ax	Ax =	$c(x)$	$c(x)$ =	Domain
P1	2	0	0	1	0	$[-1,-1] - [1,1]$
P2	2	0	0	1	0	$[-1,-1] - [1,1]$
P3	2	0	0	3	0	$[0,0] - [75,65]$
P4	3	0	0	1	0	$[-4.5,-4.5,-5] - [4.5,4.5,5]$
P5	8	0	0	6	0	$[0.1]^8 - [10]^8$
P6	6	1	0	0	0	$[0,0,1,0,1,0] - [6,6,5,6,5,10]$
P7	2	0	0	1	0	$[0.2,0.2] - [2,2]$
P8	2	0	0	2	0	$[0,0] - [75,65]$
P9	2	0	0	3	0	$[54,0] - [75,65]$
P10	2	0	0	1	0	$[0,0] - [75,65]$
P11	2	0	0	1	0	$[10^{-10},10^{-10}] - [5,5]$
P12	2	0	0	2	0	$[0,0] - [1.5,1.5]$
P13	3	0	0	2	0	$[0,0,0] - [36,5,125]$
P14	8	3	0	3	0	$10 \times [10,100,200,1,1,1,1,1] - 500 \times [2,4,12,1,1,1,1,1]$
P15	5	0	0	6	0	$[78,33,27,27,27] - [102,45,45,45,45]$
P17	3	2	0	1	0	$[0,0,0] - [2,2,3]$
P18	2	0	0	2	0	$[0,0] - [3,4]$
P28	2	0	0	2	0	$[0,0] - [100,100]$

Table 16 and Table 17 illustrate the results for *OQNLP*, *MINLPBB* and *MISQP*. The same settings as for the MINLP-problems were used. In general *OQNLP* performed much better for this test set than for the MINLP problems. The deterministic solvers have rather many failures that are due to bad starting points. The function evaluations needed are less than for the MINLP problems, but still much higher than for the CGO solvers.

Table 18 presents the results for the *GENO* solver running each problem $50 \times 5 = 250$ times. The results are slightly better than for MINLP problems, and a few problems are always solved in less than 10000 function evaluations. Still, six problems are never solved, and many others are solved only in very few runs. The number of function evaluations needed to achieve the required accuracy is several orders of magnitude larger than for the other solvers used. We conclude that, as for the MINLP tests, a stochastic genetic algorithm is not suitable for any form of costly optimization.

Table 12: Number of function evaluations to get within 1% of the optimal value for *rbfSolve* on the constrained global test problems.

ExD		LAC									CLH							
RBF		TPS				Cubic					TPS				Cubic			
Scale		On		Off		On		Off			On		Off		On		Off	
Repl	IP	Yes	No	Yes	No	Yes	No	Yes	No	IP	Yes	No	Yes	No	Yes	No	Yes	No
P1	4	5	5	5	5	5	5	5	5	3	9	9	7	8	6	6	5	5
P2	4	31	28	31	28	13	22	10	22	3	12	12	15	18	15	12	12	6
P3	4	16	9	16	9	9	10	9	10	3	21	12	22	23	12	12	18	12
P4	5	20	23	14	23	23	20	14	20	4	25	25	22	13	22	13	22	16
P5	10	64	46	126	54	84	46	63	34	9	36	24	18	36	36	36	48	27
P6	10	-	-	-	-	-	-	-	-	9	72	105	-	-	195	93	-	-
P7	4	5	5	5	5	5	5	5	5	3	5	4	7	4	5	4	7	4
P8	4	11	5	11	5	5	5	5	5	3	6	4	6	4	7	4	7	4
P9	4	5	5	5	5	5	5	5	5	3	4	4	4	5	4	4	4	5
P10	4	11	5	11	5	5	5	5	5	3	6	4	6	4	7	4	7	4
P11	4	-	-	-	-	-	-	-	-	3	36	6	14	6	23	6	12	6
P12	4	13	48	16	13	61	6	16	6	3	18	36	15	21	15	9	21	36
P13	5	15	14	10	10	7	7	10	10	4	13	10	38	39	7	10	17	17
P14	10	19	13	26	11	25	13	22	12	9	21	12	33	11	15	12	21	18
P15	7	13	9	13	9	13	9	12	10	6	7	8	7	9	7	7	9	9
P17	5	6	6	6	6	6	6	6	6	4	7	47	5	5	7	55	5	5
P18	4	5	5	5	5	5	5	5	5	3	67	117	5	5	61	15	5	5
P28	4	5	5	5	5	5	5	5	5	3	55	11	4	4	6	11	4	4

Table 13: Number of function evaluations to get within 0.01% of the optimal value for *rbfSolve* on the constrained global test problems.

ExD		LAC									CLH							
RBF		TPS				Cubic					TPS				Cubic			
Scale		On		Off		On		Off			On		Off		On		Off	
Repl	IP	Yes	No	Yes	No	Yes	No	Yes	No	IP	Yes	No	Yes	No	Yes	No	Yes	No
P1	4	5	5	5	5	5	5	5	5	3	18	15	8	15	9	12	6	5
P2	4	55	28	52	40	19	25	19	34	3	30	15	30	30	18	18	12	12
P3	4	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-
P4	5	32	44	35	38	29	20	23	20	4	43	43	37	37	37	22	31	22
P5	10	175	115	-	124	208	100	175	166	9	-	-	183	168	102	156	207	48
P6	10	-	-	-	-	-	-	-	-	9	-	-	-	-	-	-	-	-
P7	4	5	5	5	5	5	5	5	5	3	5	4	7	4	5	4	7	4
P8	4	11	5	11	5	5	5	5	5	3	6	4	6	4	7	4	7	4
P9	4	5	5	5	5	5	5	5	5	3	4	4	4	5	4	4	4	5
P10	4	11	5	11	5	5	5	5	5	3	6	4	6	4	7	4	7	4
P11	4	-	-	-	-	-	-	-	-	3	39	6	18	6	30	21	18	24
P12	4	13	49	16	22	61	6	16	19	3	18	39	15	24	21	9	21	36
P13	5	15	16	10	13	7	7	10	13	4	13	10	38	39	10	10	17	17
P14	10	31	13	73	13	43	13	34	13	9	63	12	75	12	29	12	21	18
P15	7	13	10	13	10	13	10	12	10	6	9	9	7	9	9	9	9	9
P17	5	6	6	6	6	6	6	6	6	4	7	47	5	5	7	57	5	5
P18	4	5	5	5	5	5	5	5	5	3	67	117	5	5	61	15	5	5
P28	4	7	7	7	5	7	7	7	5	3	55	11	4	4	6	11	4	4

Table 14: Number of function evaluations to get within 1% of the optimal value for *ARBFMIP* on the constrained global test problems.

ExD		LAC					CLH					LH				
RBF		TPS		Cubic			TPS		Cubic			TPS		Cubic		
Scale	IP	On	Off	On	Off	IP	On	Off	On	Off	IP	On	Off	On	Off	
P1	4	5	5	5	5	3	6	5	5	5	21	22	22	22	22	
P2	4	19	81	10	-	3	11	9	13	9	21	52	35	-	28	
P3	4	14	12	8	8	3	53	22	26	39	21	38	42	41	28	
P4	5	56	20	20	18	4	50	20	23	20	33	82	50	42	40	
P5	10	53	33	31	44	9	25	20	37	20	65	66	71	66	77	
P6	10	66	-	-	-	9	45	-	-	-	65	206	-	212	-	
P7	4	5	5	5	5	3	4	4	4	4	21	22	22	22	22	
P8	4	13	19	9	9	3	5	7	5	7	21	25	26	22	22	
P9	4	5	7	5	7	3	5	5	5	5	21	31	22	23	23	
P10	4	16	11	9	9	3	5	7	5	7	21	25	26	22	22	
P11	4	70	-	62	84	3	5	9	6	9	21	173	156	65	60	
P12	4	8	9	7	7	3	19	-	-	-	21	-	23	24	25	
P13	5	10	8	12	6	4	5	8	5	8	33	35	35	35	34	
P14	10	11	11	11	11	9	10	10	10	10	65	-	-	-	93	
P15	7	8	8	8	8	6	11	9	12	9	51	59	55	54	53	
P17	5	6	5	6	6	4	5	5	5	5	33	34	34	34	34	
P18	4	5	4	5	5	3	4	4	4	4	21	22	22	22	22	
P28	4	5	4	5	5	3	4	4	4	4	21	22	22	22	22	

Table 15: Number of function evaluations to get within 0.01% of the optimal value for *ARBFMIP* on the constrained global test problems.

ExD		LAC					CLH					LH				
RBF		TPS		Cubic			TPS		Cubic			TPS		Cubic		
Scale	IP	On	Off	On	Off	IP	On	Off	On	Off	IP	On	Off	On	Off	
P1	4	5	5	5	5	3	10	6	6	6	21	22	22	22	22	
P2	4	21	89	14	-	3	24	13	15	13	21	79	49	-	29	
P3	4	-	-	-	-	3	54	-	-	-	21	-	-	-	-	
P4	5	73	27	41	22	4	103	26	37	26	33	95	75	47	40	
P5	10	178	129	132	78	9	87	76	133	76	65	164	222	133	133	
P6	10	-	-	-	-	9	-	-	-	-	65	-	-	-	-	
P7	4	5	5	5	5	3	4	4	4	4	21	36	34	22	22	
P8	4	14	20	9	9	3	5	7	5	7	21	26	26	22	22	
P9	4	6	7	8	7	3	5	5	5	5	21	34	22	26	23	
P10	4	17	11	9	9	3	5	7	5	7	21	27	26	22	22	
P11	4	106	-	62	-	3	18	26	-	26	21	-	200	209	-	
P12	4	8	9	7	50	3	19	-	-	-	21	-	23	24	25	
P13	5	10	11	12	6	4	5	8	5	8	33	35	35	35	34	
P14	10	11	11	11	11	9	10	10	10	10	65	-	-	-	93	
P15	7	8	8	8	8	6	22	9	13	9	51	63	55	62	53	
P17	5	6	5	6	6	4	5	5	5	5	33	34	34	34	34	
P18	4	5	4	5	5	3	4	4	4	4	21	22	22	22	22	
P28	4	5	4	5	5	3	4	4	4	4	21	22	22	22	22	

Table 16: Number of function evaluations to get within 1% of the optimal value for *OQNLP*, *MINLPBB* and *MISQP* on the constrained global test problems.

Solver	%	OQNLP			%	MINLPBB			%	MISQP		
	Fail	mean	min	max	Fail	mean	min	max	Fail	mean	min	max
P1	0	139 *133*	1 *1*	294 *287*	33	23 *10*	1 *1*	66 *42*	78	14 *14*	1 *1*	63 *63*
P2	0	706 *690*	13 *10*	2629 *2567*	87	82 *54*	43 *28*	167 *111*	91	31 *31*	12 *12*	46 *46*
P3	0	286 *279*	20 *17*	1061 *1042*	69	43 *27*	23 *12*	81 *54*	67	26 *26*	10 *10*	49 *49*
P4	0	76 *72*	51 *47*	94 *90*	0	71 *39*	71 *39*	71 *50*	0	28 *28*	21 *21*	38 *38*
P5	0	286 *275*	88 *79*	710 *690*	0	940 *891*	577 *490*	1904 *2069*	0	142 *142*	82 *82*	227 *227*
P6	0	114 *1*	23 *1*	394 *1*	7	284 *1*	86 *1*	665 *1*	2	93 *11*	23 *3*	280 *32*
P7	0	18 *15*	9 *6*	31 *28*	0	24 *7*	13 *4*	53 *21*	0	10 *10*	4 *4*	26 *26*
P8	0	31 *28*	9 *6*	75 *70*	15	31 *13*	13 *4*	47 *56*	18	26 *26*	7 *7*	49 *49*
P9	0	168 *164*	8 *5*	488 *483*	0	88 *49*	13 *4*	136 *82*	0	29 *29*	7 *7*	52 *52*
P10	0	52 *48*	10 *7*	259 *254*	19	31 *13*	13 *4*	44 *40*	26	26 *26*	10 *10*	52 *52*
P11	0	71 *68*	14 *11*	106 *103*	0	76 *53*	35 *17*	287 *123*	0	30 *30*	14 *14*	73 *73*
P12	0	149 *141*	17 *14*	645 *617*	13	335 *242*	127 *88*	1355 *1056*	100	-	-	-
P13	0	163 *159*	17 *13*	366 *362*	0	105 *76*	21 *6*	185 *140*	12	43 *43*	10 *10*	89 *89*
P14	0	423 *413*	118 *109*	1535 *1474*	0	557 *852*	176 *170*	1069 *1535*	16	3610 *3609*	47 *847*	24811 *24806*
P15	0	106 *99*	25 *19*	418 *408*	0	98 *59*	40 *7*	151 *112*	100	-	-	-
P17	0	102 *97*	21 *17*	305 *295*	2	69 *50*	20 *5*	142 *126*	18	25 *25*	5 *5*	57 *57*
P18	0	254 *247*	14 *11*	847 *822*	64	39 *25*	23 *12*	65 *51*	48	15 *15*	7 *7*	23 *23*
P28	0	888 *872*	22 *19*	5650 *5543*	45	50 *33*	33 *20*	106 *74*	46	17 *17*	10 *10*	37 *37*

Table 17: Number of function evaluations to get within 0.01% of the optimal value for *OQNLP*, *MINLPBB* and *MISQP* on the constrained global test problems.

Solver	%	OQNLP			%	MINLPBB			%	MISQP		
	Fail	mean	min	max	Fail	mean	min	max	Fail	mean	min	max
P1	0	172 *165*	9 *6*	331 *324*	33	24 *10*	13 *4*	66 *42*	87	13 *13*	7 *7*	28 *28*
P2	0	847 *826*	54 *51*	2670 *2608*	87	86 *57*	52 *31*	167 *111*	91	34 *34*	26 *26*	46 *46*
P3	100	-	-	-	100	-	-	-	100	-	-	-
P4	0	84 *80*	59 *55*	103 *99*	0	71 *39*	71 *39*	71 *50*	0	28 *28*	21 *21*	38 *38*
P5	0	457 *443*	132 *123*	1175 *1140*	0	940 *891*	577 *490*	1904 *2069*	0	156 *156*	100 *100*	244 *244*
P6	100	-	-	-	100	-	-	-	100	-	-	-
P7	0	21 *18*	9 *6*	32 *29*	0	37 *11*	13 *4*	73 *27*	4	17 *17*	4 *4*	38 *38*
P8	0	31 *28*	9 *6*	75 *70*	15	32 *13*	13 *4*	47 *56*	18	26 *26*	10 *10*	49 *49*
P9	0	181 *177*	8 *5*	491 *483*	0	88 *49*	13 *4*	136 *82*	0	29 *29*	7 *7*	52 *52*
P10	0	52 *48*	10 *7*	259 *254*	19	31 *13*	13 *4*	44 *40*	26	27 *27*	10 *10*	52 *52*
P11	0	80 *77*	28 *25*	112 *109*	0	76 *53*	35 *17*	287 *123*	0	30 *30*	14 *14*	76 *76*
P12	0	191 *182*	51 *48*	668 *640*	91	763 *626*	401 *314*	1177 *991*	100	-	-	-
P13	0	207 *203*	27 *23*	606 *600*	0	127 *95*	38 *21*	228 *178*	12	45 *46*	10 *10*	92 *92*
P14	0	509 *498*	143 *134*	1685 *1624*	0	585 *880*	392 *499*	1069 *1535*	19	3740 *3739*	973 *973*	24811 *24806*
P15	0	209 *202*	43 *37*	799 *781*	0	106 *66*	77 *42*	151 *112*	100	-	-	-
P17	0	180 *174*	24 *20*	917 *895*	2	86 *65*	37 *20*	159 *141*	18	30 *30*	9 *9*	61 *61*
P18	0	317 *309*	19 *15*	1028 *999*	64	39 *25*	23 *12*	65 *51*	48	15 *15*	7 *7*	23 *23*
P28	1	1898 *867*	24 *21*	11501 *11322*	45	50 *33*	33 *20*	106 *74*	46	17 *17*	10 *10*	37 *37*

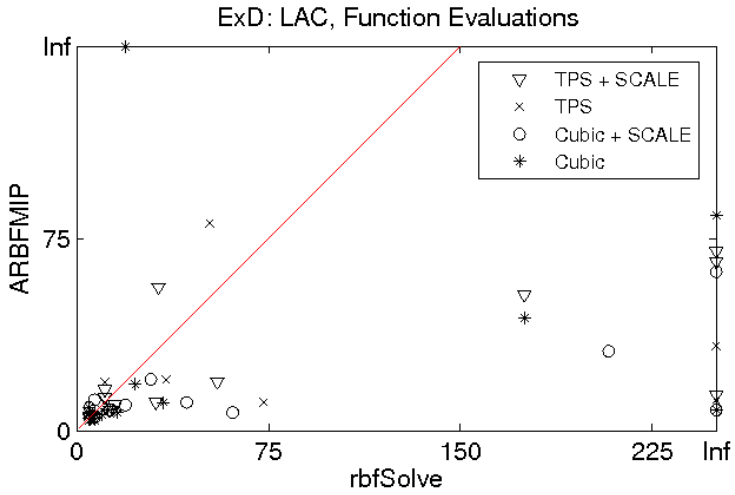

Figure 1: Number of function evaluations for ARBFMIP vs. rbfSolve for Table 13 and Table 15. Values for the choice Repl = Yes in Table 13 are used. In total there are 13 points of each kind. Four points were removed since both solvers used the maximum number of function evaluations.

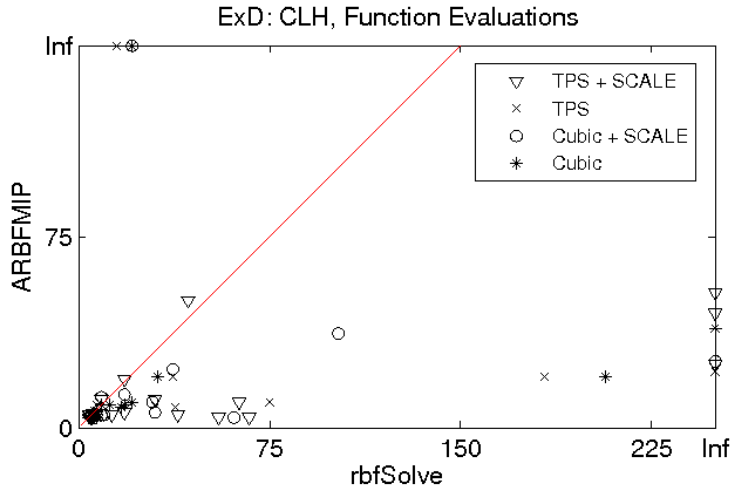

Figure 2: Number of function evaluations for ARBFMIP vs. rbfSolve for Table 13 and Table 15. Values for the choice Repl = Yes in Table 13 are used. In total there are 13 points of each kind. Three points were removed since both solvers used the maximum number of function evaluations.

Table 18: Number of function evaluations to get within 1% and 0.01% of the optimal value for *GENO* on the constrained global test problems.

	%	$E \leq 10^{-2}$			%	$E \leq 10^{-4}$		
	Fail	mean	min	max	Fail	mean	min	max
P1	0	100 *81*	1 *1*	1380 *725*	0	3091 *1539*	7 *7*	5438 *2780*
P2	44	4712 *2280*	1 *1*	15105 *9208*	45	10342 *5510*	7335 *2605*	15561 *9607*
P3	86	2116 *1363*	93 *92*	11920 *6126*	100	-	-	-
P4	87	5377 *3711*	3661 *2121*	11210 *8348*	98	14180 *10434*	9467 *6536*	15226 *11962*
P5	96	7564 *7071*	4149 *3826*	12025 *11151*	100	-	-	-
P6	100	-	-	-	100	-	-	-
P7	100	5106 *2954*	5106 *2954*	5106 *2954*	100	-	-	-
P8	0	312 *302*	2 *2*	968 *904*	0	5710 *3438*	1169 *936*	8004 *4946*
P9	100	-	-	-	100	-	-	-
P10	0	298 *287*	1 *1*	1105 *1059*	0	5341 *3259*	3448 *1926*	8046 *4823*
P11	100	-	-	-	100	-	-	-
P12	98	7365 *2974*	7280 *2794*	7492 *3249*	100	-	-	-
P13	70	2566 *1978*	271 *268*	12527 *8965*	71	9047 *6605*	5099 *3558*	14256 *11590*
P14	100	-	-	-	100	-	-	-
P15	100	-	-	-	100	-	-	-
P17	98	7379 *3495*	7227 *3117*	7530 *3837*	100	-	-	-
P18	44	4356 *2126*	3425 *1103*	7969 *4180*	46	11872 *6803*	7552 *3363*	14705 *9041*
P28	90	6366 *3816*	3531 *2131*	9368 *5983*	97	13564 *8429*	11384 *6727*	14332 *9302*

6 Conclusions

Methods based on radial basis interpolation are powerful tools for solving expensive black-box optimization problems. The paper presents extensions of two algorithms based on RBF interpolation to handle black-box mixed-integer nonlinear programs and nonconvex nonlinear programs. Algorithms have been discussed in detail as well as the MATLAB implementations of two solvers, *rbfSolve* and *ARBFMIP*, in the TOMLAB optimization environment.

A large number of numerical tests on black-box MINLP and nonlinear programs are presented that compare seven different MINLP solvers. The results show that the dedicated costly global black-box optimization solvers, including a third solver, *EGO*, outperform other approaches. The use of derivative-based solvers is possible, but an order of magnitude higher number of function evaluations are normally needed. Starting values need to be carefully set to avoid failues. Stochastic black-box solvers, like the tested *GENO* solver, should definitely be avoided in this context.

The RBF method uses a static selection of target values and is dependent on the scaling of the problem. The global target value optimization problem often does not produce interior points, and the search points computed do not help the practical convergence of the algorithm.

The details of a new Adaptive RBF (ARBF) method extended to MINLP have been presented. In every iteration, the method does an extensive search for target values to produce a suitable selection of search points. In general, the computational overhead is substantial. However, since the global subsolvers in TOMLAB are very efficient any problem considered expensive will benefit from the new ARBF algorithm. It is possible to parallelize the global target value optimization as well as the costly function evaluations. These two tasks are the most CPU-intensive parts of the algorithm. The conclusion is that the ARBF approach is the best. Further research is needed to implement a parallel algorithm. Some more algorithmic work is also needed to make the ARBF algorithm and the solver robust for more types of problems. There may be some sensitivity to additive scaling in parts of the algorithms, especially in the choice of f_Δ. This sensitivity could easily be removed, but time did not permit testing and re-solving. However, we are working on an enhanced ARBF algorithm that will be much less sensitive to the choice of f_Δ and β.

References

[1] M. H. Bakr, J. W. Bandler, K. Madsen, and J. Sondergaard: 2000, 'Review of the Space Mapping Approach to Engineering Optimization and Modeling'. *Optimization and Engineering* **1**(3), 241–276.

[2] M. Björkman and K. Holmström: 2000, 'Global optimization of costly nonconvex functions using radial basis functions'. *Optimization and Engineering* **1**(4), 373–397.

[3] H.-M. Gutmann: 1999, 'A radial basis function method for global optimization'. Technical Report DAMTP 1999/NA22, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, England.

[4] H.-M. Gutmann: 2001a, 'A radial basis function method for global optimization'. *Journal of Global Optimization* **19**, 201–227.

[5] H.-M. Gutmann: 2001b, 'Radial Basis Function Methods for Global Optimization'. Doctoral Thesis, Department of Numerical Analysis, Cambridge University, Cambridge, UK.

[6] K. Holmström: 1999, 'The TOMLAB optimization environment in Matlab'. *Advanced Modeling and Optimization* **1**(1), 47–69.

[7] K. Holmström: 2007, 'An adaptive radial basis algorithm (ARBF) for expensive black-box global optimization'. *Journal of Global Optimization*, DOI 10.1007/s10898-007-9256-8, ISSN 0925-5001 (Print) 1573-2916 (Online).

[8] K. Holmström and M. M. Edvall: January 2004, 'CHAPTER 19: THE TOMLAB OPTIMIZATION ENVIRONMENT'. In: L. G. Josef Kallrath, BASF AB (ed.): *Modeling Languages in Mathematical Optimization*. Boston/Dordrecht/London.

[9] D. R. Jones: 2002, 'A taxonomy of global optimization methods based on response surfaces'. *Journal of Global Optimization* **21**, 345–383.

[10] D. R. Jones, M. Schonlau, and W. J. Welch: 1998, 'Efficient global optimization of expensive black-box functions'. *Journal of Global Optimization* **13**, 455–492.

[11] M. McKay, R. Beckman, and W. Conover: 1979, 'A comparison of three methods for selecting values of input variables in the analysis of output from a computer code'. *Technometrics* **21**, 239–246.

[12] M. J. D. Powell: 1992, 'The Theory of Radial Basis Function Approximation in 1990'. In: W. Light (ed.): *Advances in Numerical Analysis, Volume 2: Wavelets, Subdivision Algorithms and Radial Basis Functions*. Oxford University Press, pp. 105–210.

[13] M. J. D. Powell: 1999, 'Recent Research at Cambridge on Radial Basis Functions'. In: M. D. Buhmann, M. Felten, D. Mache, and M. W. Müller (eds.): *New Developments in Approximation Theory*. Basel: Birkhäuser, pp. 215–232.

[14] R. G. Regis and C. A. Shoemaker: 2005, 'Constrained global optimization of expensive black box functions using radial basis functions'. *Journal of Global Optimization* **31**(1), 153–171.

[15] R. G. Regis and C. A. Shoemaker: 2007, 'Improved Strategies for Radial Basis Function Methods for Global Optimization'. *Journal of Global Optimization* **37**(1), 113–135.