Mälardalen University Press Licentiate Thesis
No. 105

Algorithms for Costly Global Optimization

Nils-Hassan Quttineh

September 2009

School of Education, Culture and Communication
Mälardalen University
Västerås, Sweden

This work was funded by the Graduate School in Mathematics and Computing.

Reprinted with corrections, February 2010

Algorithms for Costly Global Optimization

Typeset by the author in LaTeX2e documentation system.

ISSN 1651-9256
ISBN 978-91-86135-29-4
Printed by Mälardalen University, Västerås, Sweden

Mälardalen University Press Licentiate Thesis

No. 105

Algorithms for Costly Global Optimization

Nils-Hassan Quttineh

Akademisk avhandling

som för avläggande av Filosofie licentiatexamen i Matematik/tillämpad matematik vid Akademin för Utbildning, Kultur & Kommunikation, avdelningen för tillämpad matematik, Mälardalens Högskola, kommer att offentligt försvaras Torsdagen, 3 September, 2009, 13:15 i Gamma, Hus U, Högskoleplan 1, Mälardalens Högskola.

Granskare: Prof. Hans Bruun Nielsen, DTU, Danmark

Akademin för Utbildning, Kultur & Kommunikation,
avdelningen för tillämpad matematik
Mälardalens Högskola, Box 883, SE-72123 Västerås

In memory of Simplex,
my beloved cat whom
I truly miss and mourn.

Abstract

There exists many applications with so-called costly problems, which means that the objective function you want to maximize or minimize cannot be described using standard functions and expressions. Instead one considers these objective functions as "black box" where the parameter values are sent in and a function value is returned. This implies in particular that no derivative information is available.

The reason for describing these problems as expensive is that it may take a long time to calculate a single function value. The black box could, for example, solve a large system of differential equations or carrying out a heavy simulation, which can take anywhere from several minutes to several hours!

These very special conditions therefore requires customized algorithms. Common optimization algorithms are based on calculating function values every now and then, which usually can be done instantly. But with an expensive problem, it may take several hours to compute a single function value. Our main objective is therefore to create algorithms that exploit all available information to the limit before a new function value is calculated. Or in other words, we want to find the optimal solution using as few function evaluations as possible.

A good example of real life applications comes from the automotive industry, where the development of new engines utilize advanced models that are governed by a dozen key parameters. The goal is to optimize the model by changing the parameters in such a way that the engine becomes as energy efficient as possible, but still meets all sorts of demands on strength and external constraints.

The thesis consists of three papers, which deal with different parts of the area costly global optimization. The first paper, Paper I, describes and evaluates an algorithm based on "Radial Basis Functions", a special type of basis functions used to create an interpolation surface linking the evaluated parameter combinations (points).

This is the foundation of most costly global optimization algorithms. The idea is to use the points where the function value have been calculated, creating an interpolation surface that (hopefully) reflects the true costly function to be optimized and use the surface to select a new point where the costly function value is calculated.

The second paper, Paper II, deals with the problem to choose a good set of starting points. The algorithms we have developed are all based on the interpolation surface technique, which makes it necessarily to pick an initial set of parameter combinations to start with. This initial set of points have a major impact on the interpolation surface, and thus also a major influence on the success of the algorithm.

Paper III describes implementation details and evaluates a variant of the well known ego algorithm, which is based on a different type of basis functions compared with Paper I.

Acknowledgements

Thanks to my supervisor Kenneth Holmström for introducing me to the field of Costly Global Optimization. I also thank my colleagues at the Division of Applied Mathematics.

The research in this thesis has been carried out with financial support from the Graduate School of Mathematics and Computing (FMB).

Västerås, August 10, 2009

Nils-Hassan Quttineh

Papers

The following papers are appended and will be referred to by their Roman numerals.

I. K. Holmström, N-H. Quttineh, M. M. Edvall, An adaptive radial basis algorithm (ARBF) for expensive black-box mixed-integer constrained global optimization, *Optimization and Engineering* (2008).

II. N-H. Quttineh, K. Holmström, The influence of Experimental Designs on the Performance of Surrogate Model Based Costly Global Optimization Solvers, *Studies in Informatics and Control* (2009).

III. N-H. Quttineh, K. Holmström, Implementation of a One-Stage Efficient Global Optimization (EGO) Algorithm, *Research Report* 2009-2, School of Education, Culture and Communication, Division of Applied Mathematics, Mälardalen University (2009).

Parts of this thesis have been presented at the following international conferences:

1. Nordic MPS, Copenhagen, Denmark, April 20-22, 2006.

2. Euro XXI, Reykavijk, Iceland, July 2-5, 2006.

3. SMSMEO, DTU in Copenhagen, Denmark, November 8-11, 2006.

4. ICCOPT-MOPTA, Hamilton, Canada, August 12-15, 2007.

5. Siam Conference on Optimization, Boston, USA, May 10-13, 2008.

6. Nordic MPS, KTH in Stockholm, Sweden, March 13-14, 2009.

7. ISMP, Chicago, USA, August 23-28, 2009.

Contents

1

Costly Global Optimization

This thesis touches the relative small but important area of Costly Global Optimization (CGO). A problem is considered costly if it is CPU-intensive, i.e time consuming, to evaluate a function value. It could be the result of a complex computer program, e.g. the solution of a PDE system, a huge simulation or a CFD calculation.

From an application perspective there are often restrictions on the variables besides lower and upper bounds, such as linear, nonlinear or even integer constraints. The most general problem formulation is as follows:

The Mixed-Integer Costly Global Nonconvex Problem

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

where $f(x) \in \mathbb{R}$ and $x_L,\ x,\ x_U \in \mathbb{R}^d$. Matrix $A \in \mathbb{R}^{m_1 \times d}$, $b_L,\ b_U \in \mathbb{R}^{m_1}$; defines the m_1 linear constraints and $c_L,\ c(x),\ c_U \in \mathbb{R}^{m_2}$ defines the m_2 nonlinear constraints. The variables x_I are restricted to be integers, where set $\mathbb{I}$ is an index subset of $\{1,\ldots,d\}$. Let $\Omega \in \mathbb{R}^d$ be the feasible set defined only by the simple bounds, the box constraints, and $\Omega_C \in \mathbb{R}^d$ be the feasible set defined by all the constraints in (1).

It is common to treat all such functions as black-box, i.e you send in a set of variables $x \in \mathbb{R}^d$ and out comes a function value $f(x)$. This means that no derivative information is available, and standard optimization algorithms won't suffice. Hence the need of special CGO solvers.

A popular way of handling the costly black-box problems is to utilize a surrogate model, or response surface, to approximate the true (costly) function. In order to perform optimization, surrogate model algorithms iteratively choose new points where the original objective function should be evaluated. This is done by optimizing a less costly utility function, also called merit function.

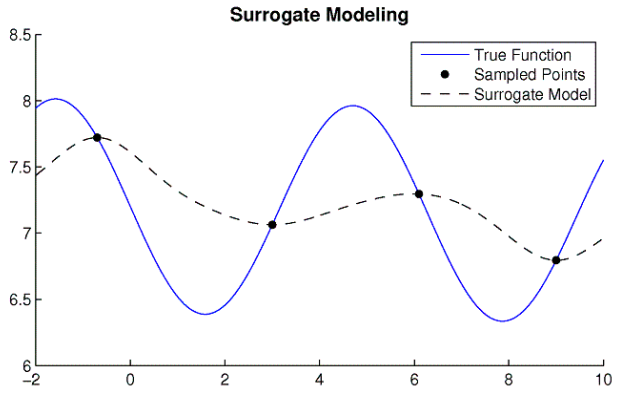


Figure 1: Surrogate modeling.

There exist different surrogate models. Jones et al. [7] introduced the "Efficient Global Optimization" (EGO) algorithm in 1998. It is based on the DACE framework, short for "Design and Analysis of Computer Experiments", and models a function as a realization of random variables, normally distributed with mean μ and variance σ^2.

In 2001, the RBF algorithm was introduced by Powell and Gutmann [1, 12] which use radial basis function interpolation to build an approximating surrogate model. An implementation of this RBF algorithm can be found in the optimization environment TOMLAB [4] by the name `rbfSolve`. Another implementation using radial basis functions is named `ARBFMIP` [3], also available in TOMLAB. This algorithm is described in detail in Paper I.

Different surrogate model algorithms utilize different merit functions, sometimes closely related to the surrogate model used. A thorough description of Surrogate Modeling is presented in Section 1 and some popular merit functions are presented in Section 2.

Common for all surrogate model CGO solvers is the need of an initial sample of points (Experimental Design) to be able to generate the initial surrogate model. In order to create an interpolation surface, one must use at least $n \geq d+1$ points, where d is the dimension of the problem. This is presented more thorough in Section 3 and is also the topic of Paper II.

There exist derivative-free global black-box optimization methods aimed for non-costly problems, such as the DIRECT algorithm by Jones et al. [6]. This algorithm divides the box-bounded space into rectangles, refining only areas of interest. It was later enhanced to handle constraints as well [8]. Generating set search (GSS) is a class of local derivative-free methods that find search directions iteratively and performs polls to locate the optimum.

1 Surrogate modeling

A surrogate model, or response surface, is an interpolation of sampled points and predicts the costly function values for points not yet sampled. Suppose we have evaluated the costly objective function at n distinct points in the sample space. We denote this set of sampled points by $\mathbf{x}$, and the corresponding function values by $\mathbf{y}$. The purpose of building a surrogate model is to provide an inexpensive approximation of the costly black-box function.

To improve the model, iteratively find a new point to sample, denoted x_{n+1}. For a robust and efficient algorithm, something more sophisticated than adding the surface minimum s_{min} of the interpolated surface is needed as this would result in a purely local search. Merit functions are designed to locate promising areas of the design space, suggesting new points to sample. Since merit functions are non-costly, any standard global optimization algorithm can be applied.

This procedure, locating new points, calculate the costly function value and build a new surrogate model, is the core of surrogate model algorithms. In lack of any good convergence criteria, it is common to iterate until some computational budget is exhausted. It could be a given number of function evaluations or a predefined time limit. In Algorithm 1, a pseudo-code for a generic surrogate model algorithm is found.

Algorithm 1 Pseudo-code for Surrogate Model Algorithms

1: Find $n \geq d + 1$ initial sample points $\mathbf{x}$ using some Experimental Design.
2: Compute costly $f(x)$ for initial set of n points. Best point (x_{Min}, f_{Min}).
3: **while** n < MAXFUNC **do**
4: Use the sampled points $\mathbf{x}$ to build a response surface model as an approximation of the costly function $f(x)$.
5: Find a new point x_{n+1} to sample, using some merit function.
6: Calculate the costly function value $f(x_{n+1})$.
7: Update best point (x_{Min}, f_{Min}) if $f(x_{n+1}) < f_{Min}$.
8: Update $n := n + 1$.
9: **end while**

In Figure 2 we present a graphical example. The upper left picture is the true costly function $f(x)$ to be optimized. The following pictures show the surrogate model approximation for a certain number of sampled points n, stated in each picture. As the iterations go by, the surrogate model becomes an increasingly better approximation of $f(x)$. At $n = 100$, all main features of the costly function are captured by the surrogate model.

In the following pages, we give a short introduction to the different surrogate models used throughout Papers I - III. First a description of the Radial Basis Function (RBF) interpolation and then the DACE framework.

We also demonstrate how the different interpolation surfaces model the same function by a graphical example found in Figure 3 on page 7.

Radial Basis Functions

Given n distinct points $\mathbf{x} \in \Omega$, with the evaluated function values $\mathbf{y}$, the radial basis function interpolant $s_n(x)$ has the form

$$s_n(\bar{x}) = \sum_{i=1}^{n} \lambda_i \cdot \phi\left(\left\|\mathbf{x}^{(i)} - \bar{x}\right\|_2\right) + b^T x + a, \tag{2}$$

with $\lambda \in \mathbb{R}^n$, $b \in \mathbb{R}^d$, $a \in \mathbb{R}$, where ϕ is either the cubic spline $\phi(r) = r^3$ or the thin plate spline with $\phi(r) = r^2 \log r$. The unknown parameters λ, b and a are obtained as the solution of the system of linear equations

$$\begin{pmatrix} \Phi & P \\ P^T & 0 \end{pmatrix} \begin{pmatrix} \lambda \\ c \end{pmatrix} = \begin{pmatrix} \mathbf{y} \\ 0 \end{pmatrix}, \tag{3}$$

where Φ is the $n \times n$ matrix with $\Phi_{ij} = \phi\left(\left\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\right\|_2\right)$ and

$$P = \begin{pmatrix} x_1^T & 1 \\ x_2^T & 1 \\ \vdots & \vdots \\ x_n^T & 1 \end{pmatrix}, \lambda = \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{pmatrix}, c = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_d \\ a \end{pmatrix}, \mathbf{y} = \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{pmatrix}. \tag{4}$$

If the rank of P is $d+1$, then the matrix $\begin{pmatrix} \Phi & P \\ P^T & 0 \end{pmatrix}$ is nonsingular and the linear system (3) has a unique solution [11]. Thus we have a unique RBF interpolation function $s_n(x)$ to the costly $f(x)$ defined by the points $\mathbf{x}$.

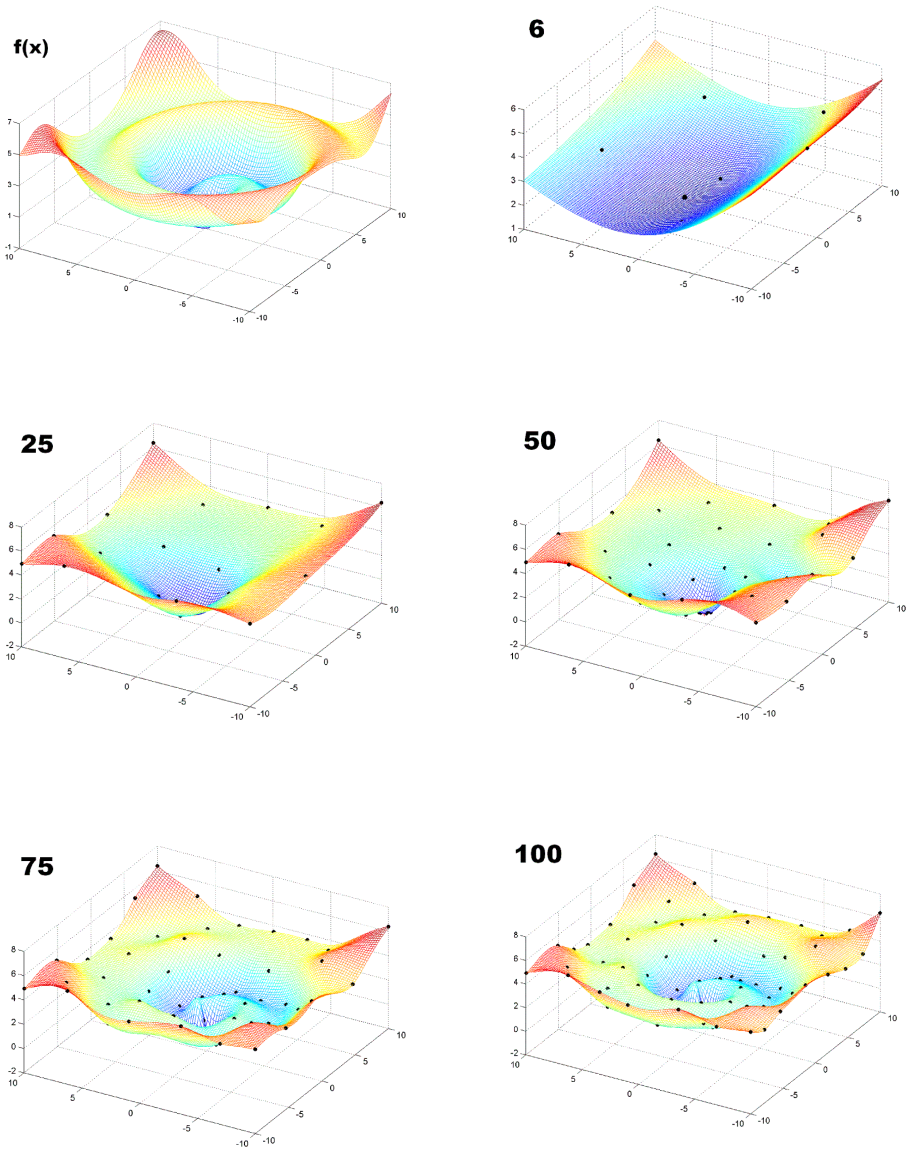


Figure 2: The top left picture is the true costly function $f(x)$ to be optimized. The following pictures are surrogate models, based on the number of sampling points n in bold face. As the iterations go by, the surrogate model becomes an increasingly more correct description of $f(x)$.

DACE Framework

As mentioned earlier, DACE models a function as a realization of random variables, normally distributed with mean μ and variance σ^2. Estimates of μ and σ^2 are found using Maximum Likelihood Estimation (MLE) with respect to the n sampled points $\mathbf{x}$ and their corresponding function values $\mathbf{y}$.

Using a matrix of correlation values $\mathbf{R}$, the DACE interpolant is defined by

$$y(\bar{x}) \;=\; \mu + \mathbf{r'}\mathbf{R^{-1}}(\mathbf{y} - \mathbf{1}\mu) \tag{5}$$

where $\mathbf{r}$ is the vector of correlations between $\bar{x}$ and $\mathbf{x}$. The first term μ is the estimated mean, and the second term represents the adjustment to this prediction based on the correlation of sampled points $\mathbf{x}$.

The correlation function is defined as

$$Corr\left[\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right] \;=\; e^{-D\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right)} \tag{6}$$

with respect to some distance formula. Compared with Euclidean distance, where every variable is weighted equally, DACE utilize the distance formula

$$D\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) \;=\; \sum_{k=1}^{d} \theta_k \cdot \left|x_k^{(i)} - x_k^{(j)}\right|^{p_k} \qquad \theta_k > 0,\ p_k \in [1, 2] \tag{7}$$

which is designed to capture functions more precise. The exponent p_k is related to the smoothness of the function in the kth dimension. Values of p_k near 2 corresponds to smooth functions and values near 1 to less smoothness. Parameter θ_k controls the impact of changes in variable x_k.

RBF versus DACE

Both interpolation techniques produce surrogate models, and the results are often very similar. The main features are practically the same, only small details might differ significantly. In Figure 3 we show an example of RBF and DACE interpolation, comparing the different techniques by approximating the same costly function using only $n = 9$ sampled points.

In the top left picture, the costly function to be optimized. The top right picture illustrates the DACE interpolation surface where the parameters are found using MLE. The two pictures in the bottom are RBF interpolation surfaces, the left one using Thin Plate Splines and the right one using Cubic Splines. There are some small differences between them, but almost not noticeable in the pictures. DACE is the only one that reflects different scaling in the different directions, and this is due to the individual weight factors θ_k.

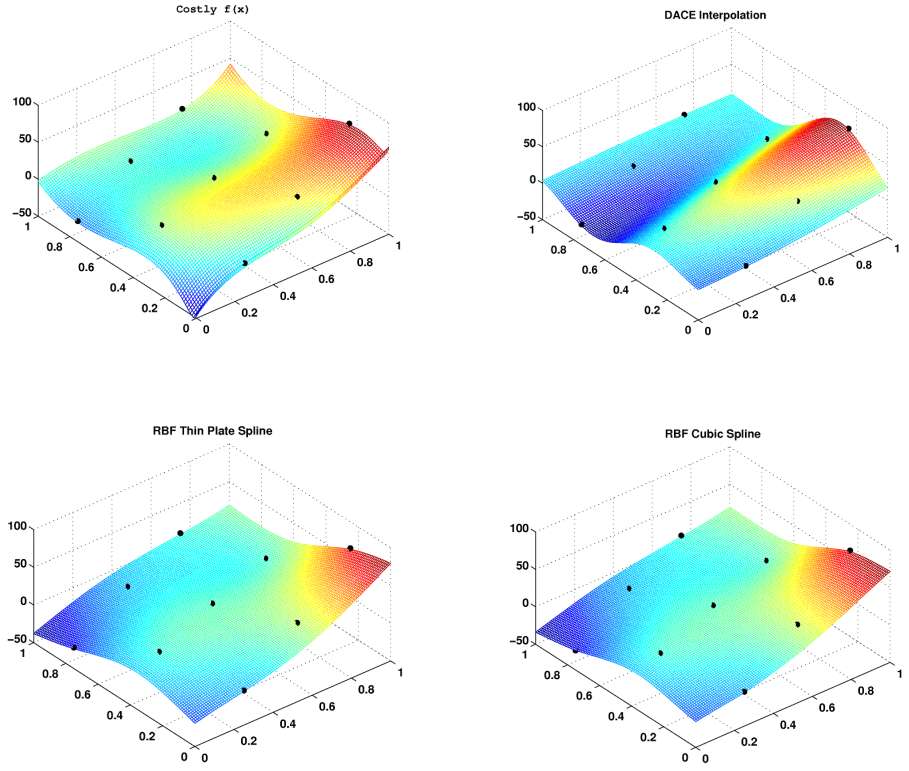


Figure 3: Surrogate models of the same function using different interpolation methods. The top left picture is the costly function to be optimized and to its right the DACE interpolation model. The bottom pictures are RBF interpolation models using Thin Plate Splines and Cubic splines respectively.

2 Merit functions

Merit functions are used to decide upon a new point, so far not sampled, where the costly objective function should be evaluated. It is important to clarify the advantage of merit functions; they are not expensive to evaluate compared to the costly black-box function.

As stated before, merit functions are not unique. The only qualifications needed are the ability to locate unexplored regions and/or investigate promising areas of the parameter space. A purely global merit function would be to find the point most distant from all sampled points, i.e. maximizing the minimal distance to sampled points.

The other extreme would be to find the global minimum of the surrogate model, denoted s_{min}, and choose this as the new point. Note that this is not an expensive problem and hence any global optimization algorithm can be used to find s_{min}.

The key to success is to balance these ambiguous goals, to somehow find both local and global points. The global search is necessary in order to avoid get stuck in a local minima, but will not likely be able to find an optimal solution with many digits of accuracy. A pure global search will converge only in the limit, clearly not preferable since function evaluations are costly.

One-stage/two-stage methods

In 2002, Jones wrote a paper [9] in which he summarized the area of costly global optimization and characterized popular algorithms and implementations. A surrogate model based method can be classified as either a two-stage procedure or a one-stage procedure, defined by the process of selecting new points to sample.

In its first stage, a two-stage method fit a response surface to the sampled points, estimating the parameters required to define a surrogate model. Then, in the second stage, these parameter estimates are considered true in order to utilize the surface to find new search points.

But considering the estimates as true is a potential source of error. The response surface might not be a good approximation of the true function, hence misleading the search for promising new points to sample.

One-stage methods do not separate the model fitting and the search for new sample points. Instead, using some measure of credibility, one seek the location of a new point x^* with function value f^* at the same time as fitting the surrogate model to already sampled data points.

The EGO algorithm traditionally utilize a two-stage procedure, first estimating parameters and then evaluating some merit function. It is possible though to construct one-stage procedures, which is explored in Paper III.

In RBF-based algorithms, it is common to use a one-stage procedure, where the merit function includes a target value f^* which is a number below the currently best found solution. A new point to sample is found by optimizing a measure of credibility of the hypothesis that the surface passes through the sampled points $\mathbf{x}$ and additionally the new point x^* with function value f^*.

In the upcoming sections, we present some different merit functions proposed by authors over the years.

Target Values

Given a set of sampled points $\mathbf{x} = \{x_1, \ldots, x_n\}$ and a target value f^*, find the point x^* which most probably has function value f^*. One should always use a target value lower than the minimum of the surrogate model, i.e. $f^* < s_{min}$.

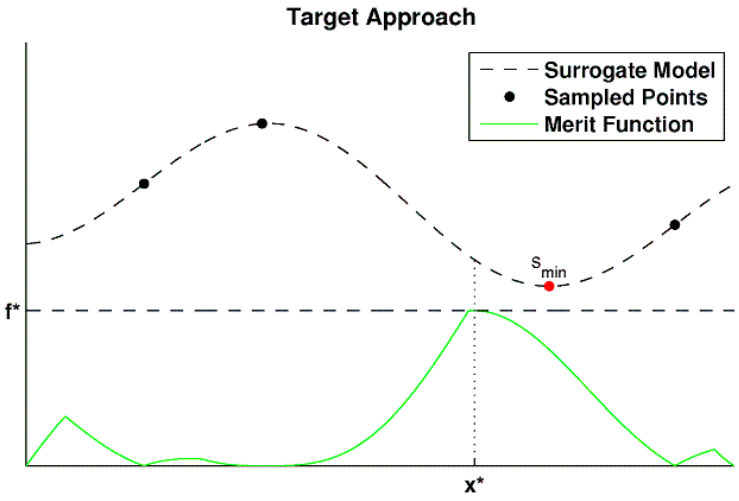


Figure 4: A merit function.

Define $\Delta = s_{min} - f^*$. If Δ is small, a modest improvement will do, i.e. a local search. A large value of Δ aims for a big improvement, i.e. global search.

The RBF algorithm utilize radial basis function interpolation and σ, a measure of 'bumpiness' of a radial function. The target value f_n^* is chosen as an estimate of the global minimum of f. For each $\bar{x} \notin \{x_1, \ldots, x_n\}$ there exists a radial basis function $s_n(x)$ that satisfies the interpolation conditions

$$\begin{aligned} s_n(x_i) &= f(x_i), \quad i = 1, \ldots, n, \\ s_n(\bar{x}) &= f_n^*. \end{aligned} \tag{8}$$

The new point x^* is then calculated as the value of x in the feasible region that minimizes $\sigma(s_n)$. In [2], a 'bumpiness' measure $\sigma(s_n)$ is defined and it is shown that minimizing $\sigma(s_n(x^*))$ subject to the interpolation conditions (8), is equivalent to minimizing a utility function $g_n(x^*)$ defined as

$$g_n(x^*) = (-1)^{m_\phi+1} \mu_n(x^*) \left[s_n(x^*) - f_n^*\right]^2, \quad x^* \in \Omega \setminus \{x_1, \ldots, x_n\}, \tag{9}$$

where $\mu_n(x^*)$ is the coefficient corresponding to x^* of the Lagrangian function L that satisfies $L(x_i) = 0$, $i = 1, \ldots, n$ and $L(x^*) = 1$.

A range of target values

Instead of using one target value f^* in each iteration, defined by some cyclic scheme to balance local and global search, it is possible to solve the utility function (9) multiple times for a range of target values.

This range of target values in which f^* lies, denoted here by F, depends on the surface minimum s_{min}. Like before, values slightly less than s_{min} means local search, and a very big value of Δ will result in a point most distant from already sampled points, i.e. global search.

Each target value $f_k^* \in F$ results in a candidate x_k^* as (9) is solved. It is not reasonable to continue with all candidates, the range F might contain as many as 40-60 values, each one connected with a costly function evaluation if utilized.

Fortunately the x_k^* candidates tend to cluster, and by applying a clustering process it is possible to proceed with a more moderate number of promising candidates, covering both local and global search. Details on how to perform the clustering is found in Paper I.

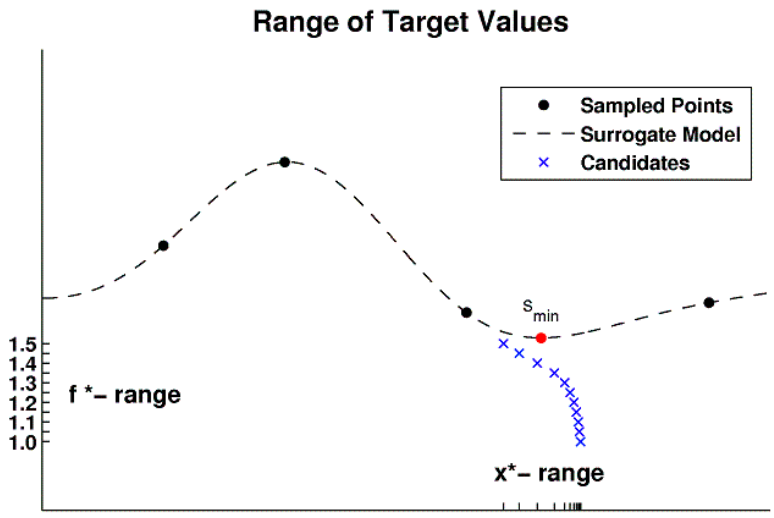


Figure 5: A range of target values.

In Figure 5 we see an example of x_k^* candidates, found by the different target values f_k^*, and how they immediately begin to cluster. This idea is implemented in the RBF-based solver `ARBFMIP`, presented in [3] and Paper I. It is also used in the one-stage EGO algorithm presented in Paper III, implemented in TOMLAB as `osEGO`.

Expected Improvement

A popular choice of merit function for the EGO algorithm it is the Expected Improvement, a merit function designed to find the point most probable to have a lower function value than f_{min}, the best one found so far. To simplify notations, we define

$$z(\bar{x}) = \frac{f_{min} - y(\bar{x})}{\sigma}$$

where σ^2 is the variance and y is the DACE interpolation model value at $\bar{x}$. The improvement over f_{min} is defined as $I = \max\{0, f_{min} - y\}$. The expected value of the improvement (ExpI) is computed as

$$ExpI(\bar{x}) \;=\; \begin{cases} (f_{min} - y) \cdot \Phi(z) + \sigma \cdot \phi(z) & \text{if} \quad \sigma > 0 \\ 0 & \qquad\quad \sigma = 0 \end{cases} \tag{10}$$

where $\phi(\cdot)$ and $\Phi(\cdot)$ denote the probability density function and cumulative density function of the standard normal distribution. The expression can be rewritten as $\sigma \cdot (z \cdot \Phi(z) + \phi(z))$. This two-stage method is connected to the DACE framework since the estimated values of parameters σ and μ are needed in order to build the response surface.

The CORS method

In 2005, Regis and Shoemaker [13] presented an RBF method for expensive black-box functions. The merit function used in this CORS (*Constrained Optimization using Response Surfaces*) method is to find the global minimum of the response surface s_{min}, but with constraints on the distance from previously evaluated points. Define the maximin distance from the n sampled points within the feasible region:

$$\Delta_n = \max_{x \in \Omega_C} \min_{1 \le i \le n} ||x - x_i||.$$

In each iteration, find the value of Δ_n and minimize the surrogate model $s_n(x)$ with respect to the additional distance constraints:

$$\begin{aligned} \min \quad & s_n(x) \\ s.t. \quad & ||x - x_i|| \;\geq\; \beta \cdot \Delta_n \qquad i = 1, \ldots, n \\ & x \in \Omega_C \end{aligned} \tag{11}$$

where $0 \le \beta \le 1$ is a scalar parameter to be specified.

This two-stage procedure is more advanced than the naive approach of just choosing the surface min s_{min} in each iteration. By varying the parameter β, using a cyclic scheme starting with values close to 1 (global search) and ending with $\beta = 0$ (local search), this merit function both explores the sample space and improves on promising areas.

The Quality function

In a recent paper, Jakobsson et al. [5] introduce a Quality function to be maximized. The merit function seeks to minimize the uncertainty of the sample space, but only in promising areas where the surrogate model predicts low function values. The uncertainty increases with distance to evaluated points $\mathbf{x}$, and the measure of uncertainty at a point $\bar{x}$ is defined as:

$$U_{\mathbf{x}}(\bar{x}) = \min_{x_i \in \mathbf{x}} ||x_i - \bar{x}||.$$

This uncertainty should be weighted against function value though, giving points with low surrogate value $s_n(\bar{x})$ a high weight. A weight function $w(s_n(\bar{x}))$ with such features can be constructed in many ways.

The quality function measures the improvement in certainty gained weighted against surrogate function value, and is given by:

$$Q(\bar{x}) = \int_{\Omega} (U_{\mathbf{x}}(x) - U_{\mathbf{x}\cup\bar{x}}(x)) \cdot w(s_n(x))\ dx. \tag{12}$$

To find a new point to evaluate, solve the following optimization problem using some standard solver:

$$\arg\max_{\bar{x}\in\Omega_C} Q(\bar{x}).$$

In order to evaluate the quality function, numerical integration is necessary, which makes this method suited only for problems in lower dimensions. To overcome this drawback, Lindström and Eriksson [10] introduced a simplified version of the quality function, avoiding the numerical integration.

Although implemented using RBF interpolation, the Quality function can be used for any kind of surrogate model. It is a two-stage process since the interpolation model is used in the calculations.

3 Experimental Designs

All surrogate based algorithms need an initial set of sample points in order to get going. To build the first interpolation surface, a minimum of $n > d + 1$ sample points is required where d is the dimension of the problem to be solved. So how should one choose these initial points?

The procedure of choosing this initial set is often referred to as Experimental Design (ExD), or sometimes Design of Experiments (DoE). We prefer the former, and hence ExD will be used throughout this thesis.

There are no general rules for an ExD, but there are some attractive features one like to achieve. Since the objective function is considered black-box, the ExD should preferably have some kind of spacefilling ability, i.e. not choose all sample point from a small part of the sample space.

Corner Point Strategy

CGO solvers tend to sample points on the boundary of the box constraints, the region of highest uncertainty of the costly function. Boundary points seldom contribute with as much information as interior points do to the interpolation surface, a problem discussed by Holmström in [3]. Sampling all corner points of the box constraints Ω, and additionally the midpoint of the box, has proven to increase the chances of generating interior points.

For this Corner Point Strategy (CPS) to perform at its best, the midpoint has to be the point with lowest function value, otherwise the initial interpolation surface will have its minimum somewhere along the boundary and hence the CGO solver will generate a boundary point. To avoid this we propose additionally sampling the corner points of half the bounding box as well, centered around the original midpoint, until we find a point with lowest function value so far. The idea is demonstrated in Figure 6.

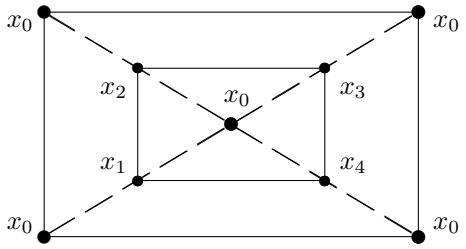


Figure 6: A Corner Point Strategy example. First sample the corner points and the midpoint, denoted by x_0. If the midpoint is not of lowest value, proceed with the inner corner points x_1, x_2, x_3 and x_4 until found.

The number of corner points $N = 2^d$ grow exponentially, which becomes an issue for problems in higher dimensions d. A possible remedy is to sample only a subset of corner points, for example only the lower left corner point of the bounding box plus all its adjacent corner points. This yields a more moderate number of initial sample points $N = d + 1$. This is also the minimum number of initial points needed for a surrogate model algorithm to get started.

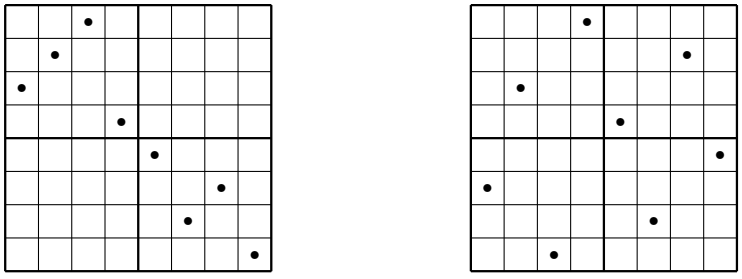
A generalization of the previous idea is to choose both the lower left and the upper right corner points, plus all adjacent corner points. This gives an initial sample of size $N = 2 \cdot (d+1)$ if $d > 2$. In two and three dimensions, the strategy is equivalent to sampling all corner points.

Latin Hypercube Designs

Latin Hypercube Designs (LHD) is a popular choice of experimental design. The structure of LHDs ensure that the sampled points cover the sampling space in a good way. They also have a non-collapsing feature, i.e. no points ever share the same value in any dimension. It is also extremely easy to generate a LHD. Suppose we need to find n sample points $x \in \mathbb{R}^d$, then simply permute the numbers $1, \ldots, n$, once for each dimension d.

Maximin LHDs give an even better design, as the points not only fulfill the structural properties of LHD designs, but also separate as much as possible in a given norm, e.g. the standard Euclidean norm. They are much harder to generate though, except for some special cases in 2 dimensions, using the 1-norm and ∞-norm, described in [15].

To clarify the limitations of a standard LHD, Figure 7 shows the sampling space divided into 4 subspaces. The random permutation approach could result in the situation seen to the left, not covering large pieces of the sampling space at all, although a valid LHD.



Latin Hypercube Design Maximin LHD

Figure 7: Different LHD sampling techniques. To the left, a LHD generated by random permutation of the main diagonal. To the right, a maximin LHD where the sample points are guaranteed to spread out.

In the right picture, a maximin LHD where the minimal Euclidian distance between each pair of sample points is maximized, at the same time fulfilling the special structure of a LHD. The maximin LHDs are clearly preferable.

It is possible to use any norm when generating these designs, but most common are the 1-norm and ∞-norm besides the standard Euclidian norm (or 2-norm). A large collection of maximin LHDs and other spacefilling designs are available at `http://www.spacefillingdesigns.nl` together with state-of-the-art articles in the area.

Random Sampling Designs

Randomly choose a point $x \in \Omega$, or optionally $x \in \Omega_C$ to handle constraints, and forbid points inside a circle with center x and radius r. Randomly select a new point, reject it if too close to any previous points. Repeat until n points are found. The result is not necessarily non-collapsing, as hinted by Figure 8.

At a closer look, we see that the radii of points 11 and 12 do overlap some of their adjacent circles. It is simply not possible to fit the last two circles inside the box, given the randomly chosen locations of the previous 10 points.

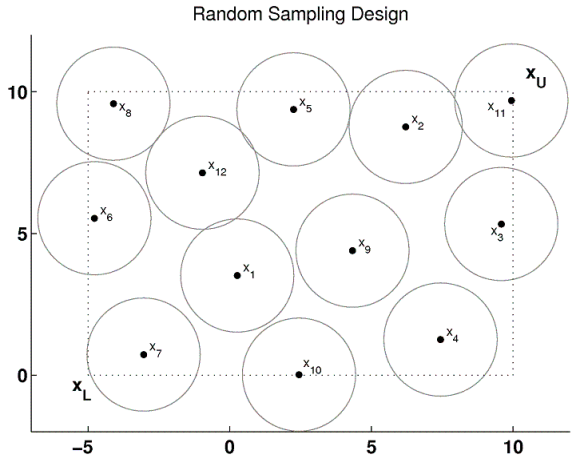


Figure 8: Example of a Random Sampling Design. For this 2-dimensional problem, 12 points have been sampled iteratively to fit inside the bounding box, marked with a dotted line.

The Random Sampling Design (RSD) algorithm generates the n sample points one at a time by randomly choosing a point inside the bounding box. If the circle does not overlap any other circles, the point is saved. If the circle do overlap another circle, the point is rejected and a new point is randomly generated. This is repeated until n sample points have been found.

To handle the situation seen in Figure 8, where it is not possible to fit the last two circles, the RSD algorithm use a specified finite number of points to reject. For each rejected point, a violation measure is calculated, and the least infeasible point is always saved. If the limit is reached, still without a feasible point, the least infeasible point is used instead.

This method is a good substitute for a maximin LHD, at least for sufficiently large values of r. Finding a maximin LHD is not easy in general, especially in higher dimensions, hence we recommend using a RSD instead if no suitable maximin LHD is available.

Optimize maximin distance

Suppose we like to sample n initial points $\mathbf{x} \in \Omega_C$ in some way, preferably with a good spacefilling ability. This can be formulated as an optimization problem, where the minimum distance between each pair of sample points is to be maximized:

$$\begin{array}{lrcll} \max\limits_{x} & d_{min} & & & \\ s.t. & d_{min} & \leq & \left\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\right\| & 1 \leq i < j \leq n \\ & \mathbf{x}^{(i)} & \in & \Omega_C & i = 1, \ldots, n \end{array} \tag{13}$$

In a paper from 2003, Stinstra et al. [14] discuss efficient methods for solving problem (13). By sequentially fixating all points but one, they solve a series of smaller problems which converges quickly. This approach also improves the minimum distances between all pairs, something often neglected in the search for the overall maximin distance.

Solutions tend to be collapsing for regular and standard regions, e.g. the optimal solution for 4 points in a square surface is always the corner points.

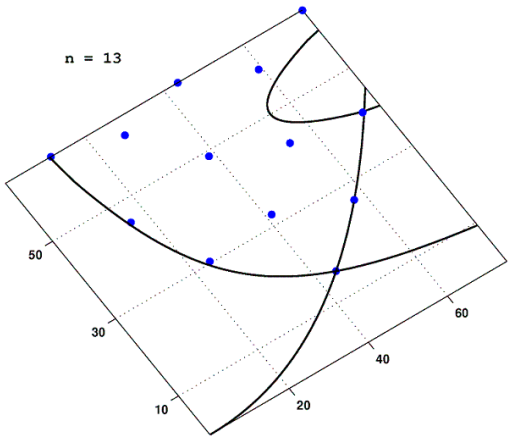


Figure 9: Example of a constrained problem where $n = 13$ sample points are distributed in the feasible area using a maximin distance objective.

In Figure 9 above the minimum distance between the $n = 13$ sample points have been maximized. This is a powerful approach since it is able to handle any kind of constraints. Notice that the distance measured between two points is not affected by the infeasible regions.

Deterministic Global Solver

Deterministic global optimization algorithms are designed to find the global optimum for a given problem. They are not suited for expensive problems, but can still be used to find good initial points.

For the Deterministic Global Solver (DGS) strategy, apply any standard global optimization solver for a limited number of iterations, just in order to get an initial set of n sample points.

The TOMLAB implementation of the DIRECT algorithm, `glcDirect`, have been used in many experiments. Because of the algorithmic structure, trisecting rectangles, the result will always be collapsing.

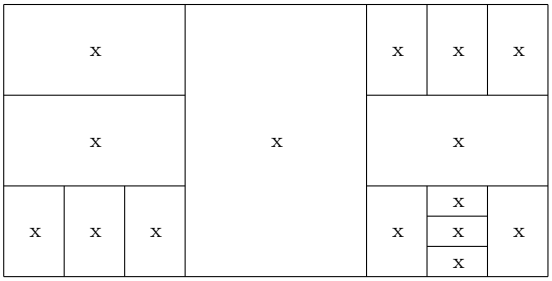


Figure 10: Using the deterministic global solver DIRECT to generate an initial design. The bounding box is iteratively trisected in order to explore promising areas systematically.

4 Convergence Criteria

A major problem in the area of CGO is the lack of a practical convergence criteria. Like any standard optimization problem, it is clear what is meant by a local and global optimizer, it is just not possible to verify for a given point.

A local minimizer x^* is a feasible point $x^* \in \Omega_C$ such that:

$$f(x^*) \leq f(x) \quad \forall\, x \in \Omega_C \quad \text{and} \quad \|x^* - x\| \leq \delta \;:\; \quad \delta \geq 0$$

restricted to a local area surrounding x^*. For a big enough value of δ, the minimizer is also global. Normally, gradient information is used to verify if a point is a local optimizer. But since no derivative information is available, what to do? When using a surrogate model approach, it is possible to approximate the real derivatives with the ones of the interpolation surface.

For some special cases, it might be possible to find a lower bound on $f(x)$, e.g. for a sum of squares or the distance between points to be minimized. If a feasible point with function value 0 is found, it must be a global optimizer.

It is also possible to utilize a measure of relative error E_r whenever a lower bound is available, stopping at some predefined tolerance:

$$E_r = \frac{f_{min} - LB}{|LB|} \ , \tag{14}$$

where f_{min} is the currently best feasible function value and LB is the best known lower bound. But in practice it is common to stop after a given budget of function evaluations or time limit.

5 Summary of Papers

This thesis is based on three papers, two which have been published in journals and one research report presented at Mälardalen University. Here follows a short summary of these papers.

Paper I and Paper III deals with the implementation and evaluation of two surrogate model based algorithms for CGO problems. In Paper II we discuss the problems involved in choosing an initial set of sample points, referred to as an Experimental Design (ExD).

Paper I

The algorithm described in Paper I utilize radial basis functions (RBF) to interpolate sampled points and contain details on implementing an enhanced version of the RBF algorithm. Introducing a range of target values for f^* we solve the merit function multiple times each iteration and cluster the resulting points.

This adaptive feature add stability to the search process, compared to the static choice of f^* defined by a cyclic scheme in the standard RBF algorithms, hence the name Adaptive RBF algorithm (ARBF). The implementation is able to handle all sorts of constraints, both linear and non-linear as well as integer restrictions on certain variables. It is available in TOMLAB and named `ARBFMIP`.

Paper II

Paper II investigate the influence of different experimental designs (ExD) on the performance of surrogate model based CGO solvers. New methods are suggested and evaluated together with standard experimental designs on a benchmark of test problems. Three CGO solvers from the TOMLAB environment are used to compare the performance of the different experimental designs.

Paper III

In Paper III we describe an extension of the EGO algorithm. The standard algorithm is a two-stage method, first estimating parameters in order to build the surrogate model, then finding a new point to sample using some merit function. The drawback with all two-stage methods is that surrogate models based on a small set of sample points might be very misleading.

It is possible though to turn EGO into a one-stage method, and in the paper we address implementation details and numerical issues, some inherited from the two-stage approach, but also some new situations. Except for an implementation by Jones, there exists no one-stage EGO algorithms to the best of our knowledge.

References

[1] H.-M. Gutmann: A radial basis function method for global optimization. *Journal of Global Optimization* **19** (3), 201–227 (2001).

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

[3] K. Holmström: An adaptive radial basis algorithm (ARBF) for expensive black-box global optimization. *Journal of Global Optimization* **41**, 447–464 (2008).

[4] 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.

[5] S. Jakobsson, M. Patriksson, J. Rudholm, and A. Wojciechowski: A method for simulation based optimization using radial basis functions. *Optimization and Engineering* (2009).

[6] D. R. Jones, C. D. Perttunen, and B. E. Stuckman: Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications* **79**, 157–181 (1993).

[7] D. R. Jones, M. Schonlau, and W. J. Welch: Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization* **13**, 455–492 (1998).

[8] D. R. Jones: DIRECT. *Encyclopedia of Optimization* (2001).

[9] D. R. Jones: A Taxonomy of Global Optimization Methods Based on Response Surfaces. *Journal of Global Optimization* **21**, 345–383 (2002).

[10] D. Lindström and K. Eriksson: A Surrogate Model based Global Optimization Method. *Proceedings 38th International Conference on Computers and Industrial Engineering* (2009).

[11] M. J. D. Powell: The theory of radial basis function approximation in 1990. In W.A. Light, editor, *Advances in Numerical Analysis, Volume 2: Wavelets, Subdivision Algorithms and Radial Basis Functions* **2**, 105–210 (1992).

[12] M. J. D. Powell: Recent Research at Cambridge on Radial Basis Functions. *New Developments in Approximation Theory*, 215–232 (2001).

[13] R. G. Regis and C. A. Shoemaker: Constrained Global Optimization of Expensive Black Box Functions Using Radial Basis Functions. *Journal of Global Optimization*, **31**, 153–171 (2005).

[14] E. Stinstra, D. den Hertog, P. Stehouwer, and A. Vestjens: Constrained Maximin Designs for Computer Experiments. *Technometrics*, **45**, 340–346 (2003).

[15] E. van Dam, B. Husslage, D. den Hertog, and H. Melissen: Maximin Latin Hypercube Designs in Two Dimensions. *CentER Discussion Paper* (2005).

Paper I

This paper has been published as:

K. HOLMSTRÖM, N-H. QUTTINEH and M. M. EDVALL, An adaptive radial basis algorithm (ARBF) for expensive black-box mixed-integer constrained global optimization, *Optimization and Engineering*, 9: 311-339, 2008.

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 $\operatorname{rank}(P) = d + 1$, 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

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

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*

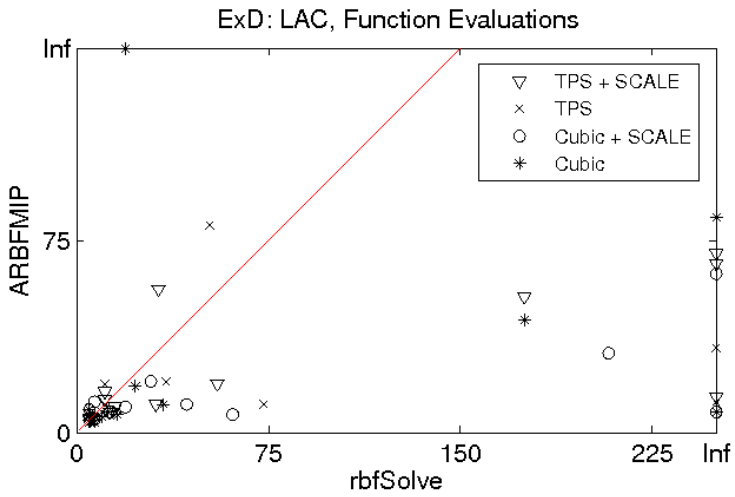


Figure 1: Number of function evaluations for ARBFMIP vs. rbfSolve for Table 13 and Table 15. Values for the choice Repl = Yes in Table 13 are used. In total there are 13 points of each kind. Four points were removed since both solvers used the maximum number of function evaluations.

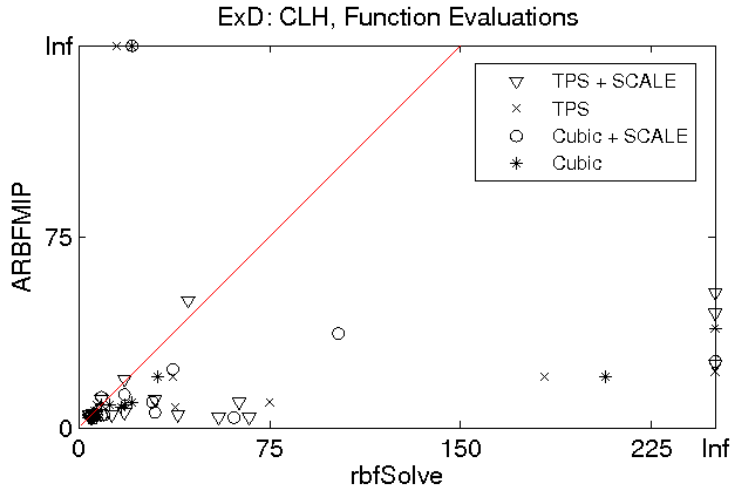


Figure 2: Number of function evaluations for ARBFMIP vs. rbfSolve for Table 13 and Table 15. Values for the choice Repl = Yes in Table 13 are used. In total there are 13 points of each kind. Three points were removed since both solvers used the maximum number of function evaluations.

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.

Paper II

This paper has been published as:

N-H. QUTTINEH and K. HOLMSTRÖM, The influence of Experimental Designs on the Performance of Surrogate Model Based Costly Global Optimization Solvers, *Studies in Informatics and Control*, 18, 2009.

The influence of Experimental Designs on the performance of surrogate model based costly global optimization solvers

Nils-Hassan Quttineh* and Kenneth Holmström*

Abstract When dealing with costly objective functions in optimization, one good alternative is to use a surrogate model approach. A common feature for all such methods is the need of an initial set of points, or "experimental design", in order to start the algorithm. Since the behavior of the algorithms often depends heavily on this set, the question is how to choose a good experimental design. We investigate this by solving a number of problems using different designs, and compare the outcome with respect to function evaluations and a root mean square error test of the true function versus the surrogate model produced. Each combination of problem and design is solved by 3 different solvers available in the TOMLAB optimization environment. Results indicate two designs as superior.

Keywords: Black-box, Surrogate model, Costly functions, Latin Hypercube Designs, Experimental Design.

Abbreviations:

CGO	Costly Global Optimization
ExD	Experimental Design
MINLP	Mixed-Integer Nonlinear Programming

Authors: N-H. Quttineh got a M.Sc. in Optimization at Linköping University 2004 and is a graduate student at Mälardalen University since 2005, with main subject *Algorithms for costly global optimization.*

K. Holmström got a Ph.D. in Numerical Analysis at Umeå University 1988 and did industrial research and development work for ABB 1990-93. Besides his academic career, he has been consultant in more than 100 industrial and scientific projects. Since 2001 he is Full Professor in Optimization at Mälardalen University with main research interests algorithm and software development for industrial and financial optimization problems, in particular costly global mixed-integer constrained nonlinear optimization. Holmström is CEO of Tomlab Optimization Inc. and creator of TOMLAB, an advanced MATLAB optimization environment distributed since 1998.

*Department of Applied mathematics, Mälardalen University, SE-721 23 Västerås, Sweden.

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 have been developed over the years. A common feature is that, unlike local optimization methods, every computed function value is saved and utilized.

Problems that are costly to evaluate are commonly found in engineering design, as well as industrial and financial applications. A function value could be the result of a complex computer program or an advanced simulation, e.g. computational fluid dynamics (CFD). Hence consuming anything from a few minutes to many hours of CPU time.

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. The most general problem formulation is as follows:

The Mixed-Integer Costly Global Black-Box Nonconvex Problem

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

where $f(x) \in \mathbb{R}$ and $x_L,\ x,\ x_U \in \mathbb{R}^d$. Matrix $A \in \mathbb{R}^{m_1 \times d}$, $b_L,\ b_U \in \mathbb{R}^{m_1}$; defines the m_1 linear constraints and $c_L,\ c(x),\ c_U \in \mathbb{R}^{m_2}$ defines the m_2 nonlinear constraints. The variables x_I are restricted to be integers, where $\mathbb{I}$ is an index subset of $\{1,\ldots,d\}$. Let $\Omega \in \mathbb{R}^d$ be the feasible set defined only by the simple bounds, the box constraints, and $\Omega_C \in \mathbb{R}^d$ be the feasible set defined by all the constraints in (1).

Almost every Costly Global Optimization (CGO) solver utilize a surrogate model, or response surface, to approximate the true (costly) function. The RBF algorithm introduced by Powell and Gutmann [2, 9] use radial basis function interpolation to build an approximating surrogate model. The EGO algorithm by Jones et al. [6] utilizes the DACE framework. By optimizing a less costly utility function these algorithms determine a new point, where afterwards the original objective function is evaluated. This is repeated until some convergence criteria is fulfilled.

2 Experimental Designs

Common for all surrogate model CGO solvers is the need of an initial sample of points (experimental design) to be able to generate the initial surrogate model. For all these points the costly function values are calculated. The initial surrogate model is built from these sampled points and used as an approximation of the true function. A new

point to sample is then decided by some algorithmic strategy, and this continues until some convergence criteria is met.

It is not obvious how to choose this initial set of points, but there are some criteria we strive to fulfill. As the problems to solve are considered black-box, we have no idea what the function might look like. Therefore it is most important that the experimental design have some sort of space filling ability, i.e. avoid sampling only a certain part of the design space.

2.1 Deterministic Global Solver

It is of course possible to utilize any standard global optimization solver for a limited number of iterations, just in order to get an initial set of sample points for the surrogate model to get going. After all, deterministic global optimization algorithms are designed to find the global optimum as fast as possible, so why not use this fact and let the solver find good initial points.

In this paper we utilize the DIRECT algorithm (DIviding RECTangles) by Jones et al. [5], implemented in the TOMLAB Optimization Environment [4] as solver *glcDirect*[2]. This is a deterministic global optimization solver, but not itself suited for the costly problems considered. The maximal number of sample points N is possible to set when running glcDirect. But, because the algorithm generates more than one new point in each iteration, the costly function value might be computed for a few more sample points than N.

2.2 Corner Point Strategy

RBF solvers tend to sample points on the border, which seldom contribute as much information as interior points to the interpolation surface. This problem is thoroughly discussed by Holmström in [3]. To increase the chances of sampling interior points, a first idea was to sample all corner points of the box constraints Ω, and additionally the midpoint of the box.

It turns out that unless the midpoint is the point with lowest function value, the initial interpolation surface will have its minimum somewhere on the boundary, and the CGO solver sometime samples a new border point. To avoid this, we propose to additionally sample corner points of half the bounding box, centered around the original midpoint, until an interior point with lowest function value is found. The idea is demonstrated in Figure 1 on page 58.

For problems in higher dimensions d, the exponential growth in number of corner points $N = 2^d$ becomes an issue. A good alternative is then to sample only a subset of corner points. One idea is to sample only the lower left corner point of the bounding box plus all its adjacent corner points. This yields a more moderate number of initial sample points $N = d + 1$, which is also the minimum number of initial points needed for the initialization of the RBF algorithm. This is due to the fact that a minimum of $N \geq d + 1$ points are required to build an interpolation surface.

[2] http://www.tomopt.com/tomlab/

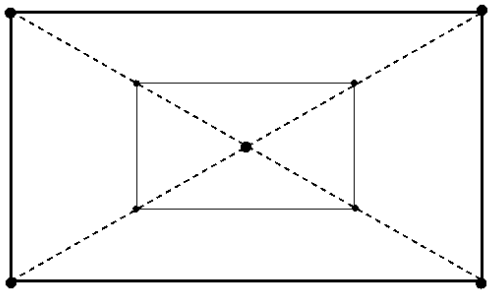


Figure 1: The Corner Point strategy in 2 dimensions.

A generalization of the previous corner idea is to choose both the lower left and the upper right corner points, plus all adjacent corner points. This gives an initial sample of size $N = 2 \cdot (d+1)$ if $d > 2$. In two and three dimensions, the strategy is equivalent to sampling all corner points.

2.3 Maximin LH Designs

Latin Hypercube Designs (LHD) is a popular choice of experimental design. The structure of LHDs ensure that the sampled points cover the sampling space in a good way. They also have a non-collapsing feature, i.e. no points ever share the same value in any dimension. Maximin LHDs give an even better design, as the points not only fulfill the structural properties of LHD designs, but also separate as much as possible in a given norm, e.g. the standard Euclidean norm. It is possible to generate Maximin LHDs for any number of points N.

A good collection of Maximin LHDs, together with many other space filling designs, can be found at `http://www.spacefillingdesigns.nl` together with state-of-the-art articles in this area.

3 Handling Constraints

When solving problems with additional constraints, besides the box constraints, it might be better to avoid sampling initial points that are not feasible since the function evaluation is extremely costly. We now describe how the proposed methods in Section 2 are adjusted to handle constraints, whenever possible. The methods presented here can not handle equality constraints at the moment, however nonlinear equality constraints are also difficult in general.

There exist other ideas on how to find a space filling initial sample, taking into account the constraints. Stinstra et al. [10] solve an optimization problem, where the objective is to maximize the minimum (euclidian) distance between N feasible points.

3.1 Constrained Deterministic Global Solver

We need to select a global deterministic solver that is able to handle constraints. The DIRECT algorithm was extended to handle nonlinear inequality constraints by Jones in [7]. In the TOMLAB implementation of the constrained DIRECT, *glcDirect*, the DIRECT algorithm is generalized to separately treat linear equality and inequality constraints, and nonlinear equality and inequality constraints. Since the algorithm always divides a rectangle in three pieces, infeasible points might still be included in the initial iterations, even if glcDirect has a feature to delete rectangles that are infeasible with respect to linear inequality constraints, and avoid computing $f(x)$ for points infeasible with respect to linear and nonlinear constraints.

3.2 Corner Point Strategy

The Corner Point Strategy is not able to handle constraints in a straightforward way. It is possible to check which generated points are feasible, but what should be done if only a few of them are feasible? One could develop strategies on how to choose additional points, but then we diverge too much from the original idea of sampling the corner points. Therefore we only consider the basic approach, i.e. not taking constraints into account.

3.3 Constrained Maximin LH Designs

We have developed a method to create an initial sample fulfilling both the LHD structure and all constraints given for the problem. The method utilizes large Maximin LHDs, where the number of points in the design is significantly larger than the desired number of initial points, and only picks out the feasible points. The method is described in pseudo-code below, see Algorithm 1.

Algorithm 1 Find N feasible Maximin LHD points

1: Initialize $M := N + \#$ constraints.
2: Apply Maximin LHD with M points to constrained problem.
3: Calculate number of feasible points M_f.
4: **if** $M_f == N$ **then**
5: STOP.
6: **else if** $M_f < N$ **then**
7: Increase M, go to 2.
8: **else**
9: Decrease M, go to 2.
10: **end if**

If the value of M_f starts to alternate between two values, one less than N and the other one greater than N, stop the algorithm and declare failure to find exactly N feasible points. The Maximin LHD with too many feasible points is used. The resulting design includes N feasible points with a Maximin LHD structure. An illustrative example is found in Figure 2 on page 60.

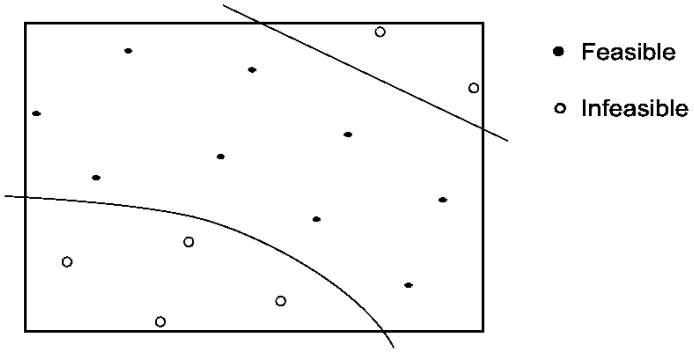


Figure 2: A 15 point Maximin LHD with $M_f = 9$ feasible points.

4 Benchmark and Tests

Our aim is to test the set of experimental designs presented in previous sections. Define the set of experimental designs as E, and pick a set of test problems P and a set of solvers S.

Every combination of problem $p \in P$ and experimental design $e \in E$ is solved with each solver $s \in S$. Below the different designs, solvers and test problems used in the benchmark is presented. The set of experimental designs E is summarized in Table 1 on page 61. Information on the test problems are found in Table 2 on page 61.

Three solvers from the TOMLAB /CGO environment are used. The `rbfSolve` and `arbfmip` solvers utilize radial basis functions, and the `EGO` solver utilizes the DACE framework. The algorithmic structures are coded in MATLAB but all heavy calculations are in TOMLAB implemented in Fortran and C code, and interfaced using so called mex file interfaces.

4.1 Set of Experimental Designs

There are two main parameters to consider: first, the number of initial sample points N, and second, for constrained problems, whether or not to take the constraints into account. The tested combinations are described and motivated below.

Size N

The Corner Point Strategy generates a fixed number of initial sample points, one for each corner point of the bounding box. The other two strategies can generate any number of initial sample points. We use $N_1 = (d+1)(d+2)/2$ and $N_2 = 10 \cdot d + 1$, where d is the dimension of the problem to be solved.

Constraints

The Maximin LHD strategy can handle constraints by applying Algorithm 1. To test whether it is more efficient to force all the initial sample points to be feasible, all problems with constraints in combination with the Maximin LHD design are solved twice. First using the standard strategy and then applying a Maximin LHD with only feasible points.

Combinations

Inspired by some preliminary results we also tried to combine the Corner Point Strategy with the other two designs. All corner points (no interior points) were added to the result of either the global optimization solver or the Maximin LHD.

Table 1: The Set of Experimental Designs (E).

Experimental Design	Size of N	Constraints
Corner Points	Fixed	No
GO Solver	N_1 and N_2	Yes
Maximin LHD	N_1 and N_2	No
Maximin LHD	N_1 and N_2	Yes
Corners + GO	N_1 and N_2	Yes
Corners + LHD	N_1 and N_2	No
Corners + LHD	N_1 and N_2	Yes

4.2 Set of Test Problems

In total, a set of 15 box-bounded unconstrained problems P_U and a set of 6 constrained problems P_C are solved. Most of them are 2-dimensional problems, except a few problems in 3 and 4 dimensions. All problems in P_C, in combination with the Maximin LHD experimental design, are solved twice (with and without taking constraints into account).

Table 2: The Set of Test Problems (P).

Problem set	P_U			P_C	
Dimension d	2	3	4	2	3
No. of problems	13	1	1	4	2

None of the test problems above have a global minimum in a corner point or midpoint, as this obviously would benefit the Corner Point Strategy.

Problems in only 2 or 3 dimensions might seem very simple, but even problems of this size are non-trivial and might be hard to solve when the problems are costly to compute. It is quite common that costly problems are of small size, with less than 10 unknowns.

5 Numerical Results

To present the benchmark results in an easy way, we utilize profiling. A performance profile [1] shows the relative performance of the solvers in S on the given set of problems P. However, performance profiles do not provide the number of function evaluations required to solve any of the problems.

Since function evaluations are expensive we are interested in the percentage of problems solved (for a given tolerance) within a given number of function evaluations. Data profiles [8] are specifically designed to handle this. These profiles are both probability density functions, but with an important difference. A data profile is independent of the set of solvers S, while the performance profiles are computed relative the other solvers in S.

5.1 Metrics

The solvers are set to break after 200 function evaluations or earlier if convergence to the known global optimum is obtained. The relative error is defined as

$$E_r = \frac{f_{min} - f_{opt}}{|f_{opt}|},$$

where f_{min} is the current best function value and f_{opt} is the known global optimum. Convergence is assumed if the relative error E_r is less than 10^{-4}. In the case $f_{opt} = 0$, stop when f_{min} is less than 10^{-4}. To compare the outcome of each experimental design, a number of metrics are used:

$\mathbf{f}_{\%}$ Number of function evaluations needed to reach 1,2,3 and 4 digits of accuracy ($E_r \leq 10^{-k} \quad k = 1, 2, 3, 4$). This is the primary goal for most optimization problems.

$\mathbf{x}_{\%}$ Number of function evaluations needed to sample a point within 10% and 1% of the design space, centered around the global optimum. It is very important to sample points close to the global optimum. When this basin is found, the CGO solvers tend to converge quickly.

RMS When the algorithm stops, the final surrogate model $s(x)$ is compared to the true function $f(x)$. A grid of points is used to calculate the Root Mean Square error.

$$RMS = \frac{1}{K} \cdot \left(\sum_{k=1}^{K} (f(x_k) - s(x_k))^2 \right)^{1/2}$$

For $d = 2,\ 41^2 = 1681$ points are used. For $d = 3,\ 21^3 = 9261$ points are used. For $d = 4,\ 11^4 = 14641$ points are used. It is preferable if the final surrogate model capture the main features of the costly function, however the main goal is to find the global minimum with few function evaluations rather than having an overall good approximation of the objective function.

Smaller values are better for all metrics. To compare the experimental designs, data profiles for the costly $f_{\%}$ and $x_{\%}$ metrics are used. The RMS measure is not costly and presented using performance profiles.

5.2 Results

Since our focus of interest is to compare the performance of experimental designs, not specific solvers, accumulated results for each design are presented and discussed. Analysis for each solver has been done as well, and if any result differs significantly for a specific solver, a note is given.

We present the analysis for the set of unconstrained problems P_U, but results are valid for P_C as well if not specified otherwise. First compare the experimental designs where N, the number of initial points, was set to either $N_1 = (d+1)(d+2)/2$ or $N_2 = 10 \cdot d + 1$.

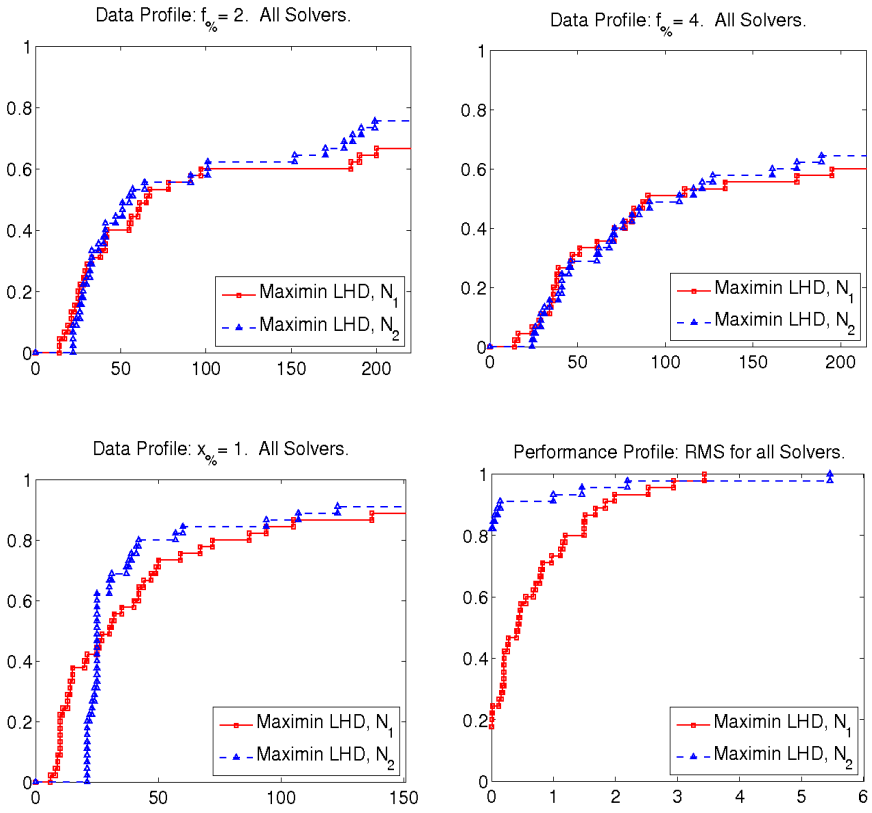


Figure 3: Comparison of setting N_1 and N_2 for Maximin LHDs. Data profiles for the metrics $f_{\%}$ and $x_{\%}$ are used, and a performance profile for RMS.

Figure 3 shows that the Maximin LHD with N_2 performs slightly better for all metrics. The results are similar for the deterministic global solver, and hence consider only the N_2 setting in forthcoming analysis.

Overall best Experimental Design

Comparing the results of the three originally proposed experimental designs, the Corner Point Strategy and the global solver approach have a very similar success rate for all metrics, as seen in Figure 4. The Maximin LHD falls behind when it comes to finding many digits of accuracy, but is superior when looking at the RMS error. But as noted, a good RMS error is not the main goal in global optimization.

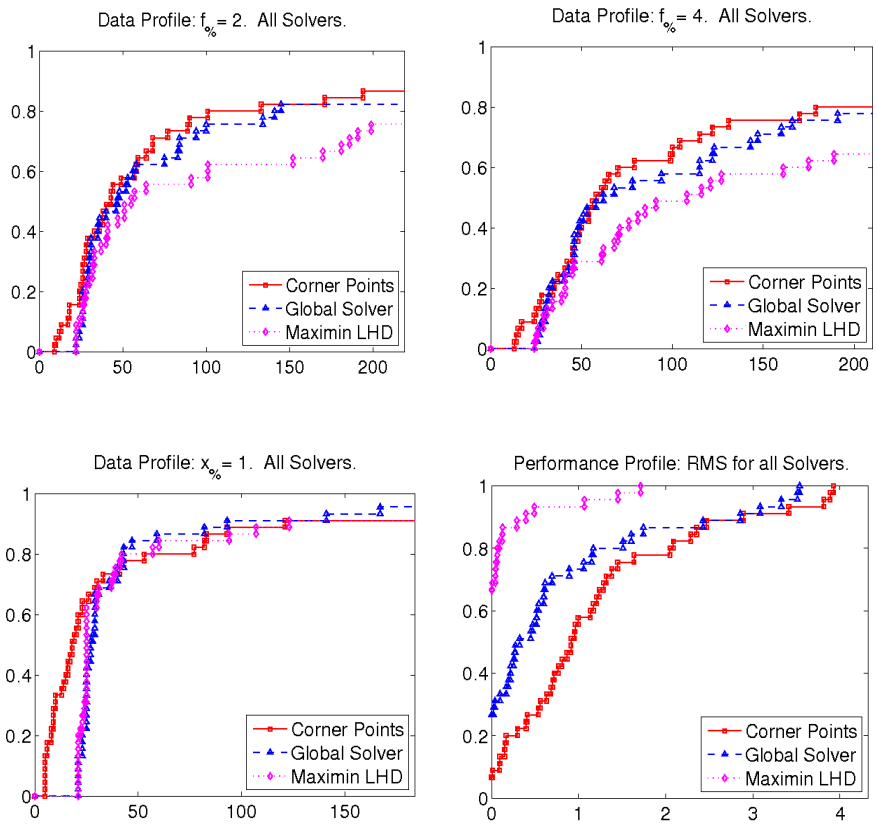


Figure 4: Comparison of the 3 proposed Experimental Designs. Data profiles for the metrics $f_{\%}$ and $x_{\%}$ are used, and a performance profile for RMS.

For the P_C problems, the Maximin LHD design performed much better and outperformed the other designs for all metrics. But since P_C contains only 6 problems this might just be a coincidence.

The high success rate of the Corner Point Strategy encouraged us to explore two combined versions. The global solver approach and the Maximin LHD is used as before, but the corner points of the bounding box are then added to the initial design.

This increases the number of initially sampled points somewhat, but should contribute to a more robust design. The outcome of this experiment is found in Figure 5.

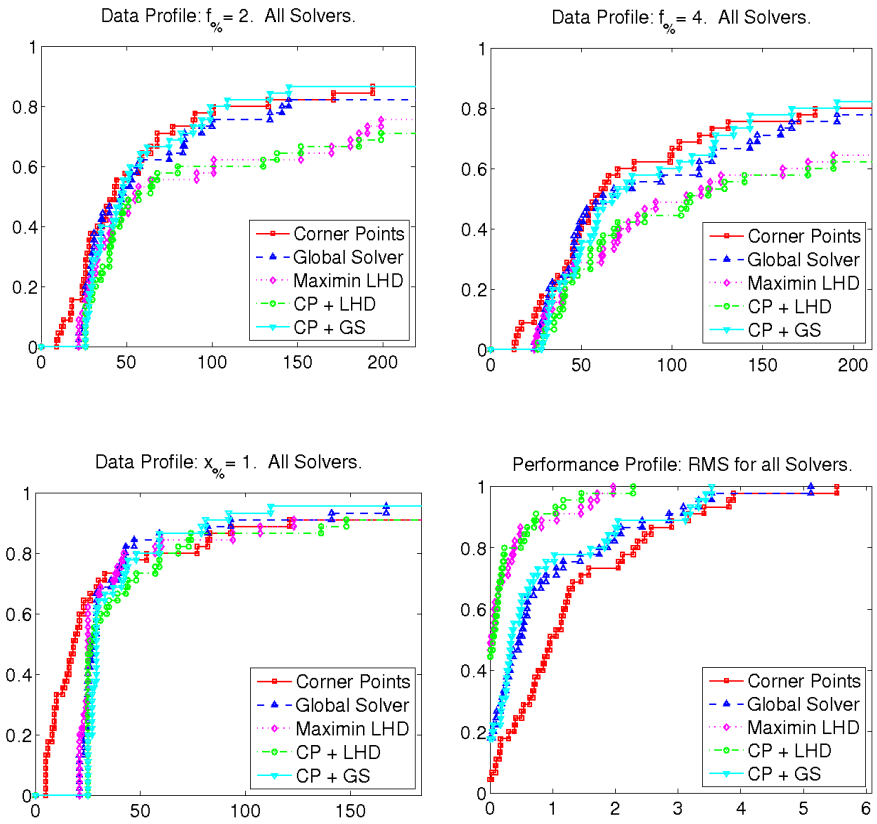


Figure 5: Comparison of Experimental Designs. Data profiles for the metrics $f_\%$ and $x_\%$ are used, and a performance profile for RMS.

A slight improvement can be seen for the Corner Points - deterministic global solver combination (CP+DGS). The second combination, Corner Points - Maximin LHD (CP+LHD), has no obvious effect on the $f_\%$ and $x_\%$ metrics.

The RMS error is improved for both combinations, and since more points are sampled initially this seems reasonable. Once again the Maximin LHD design, and the combination (CP+LHD), performed better on the P_C problems.

Constrained versions or not

The results of the ordinary Maximin LHD and the constrained version are compared on the set of constrained problems P_C. Like before, only the N_2 setting is used, since it outperforms the N_1 setting. The extra effort of finding feasible points initially seem not to pay off as one might expect. Figure 6 does not show any significant improvement for the $f_{\%}$ and $x_{\%}$ metrics.

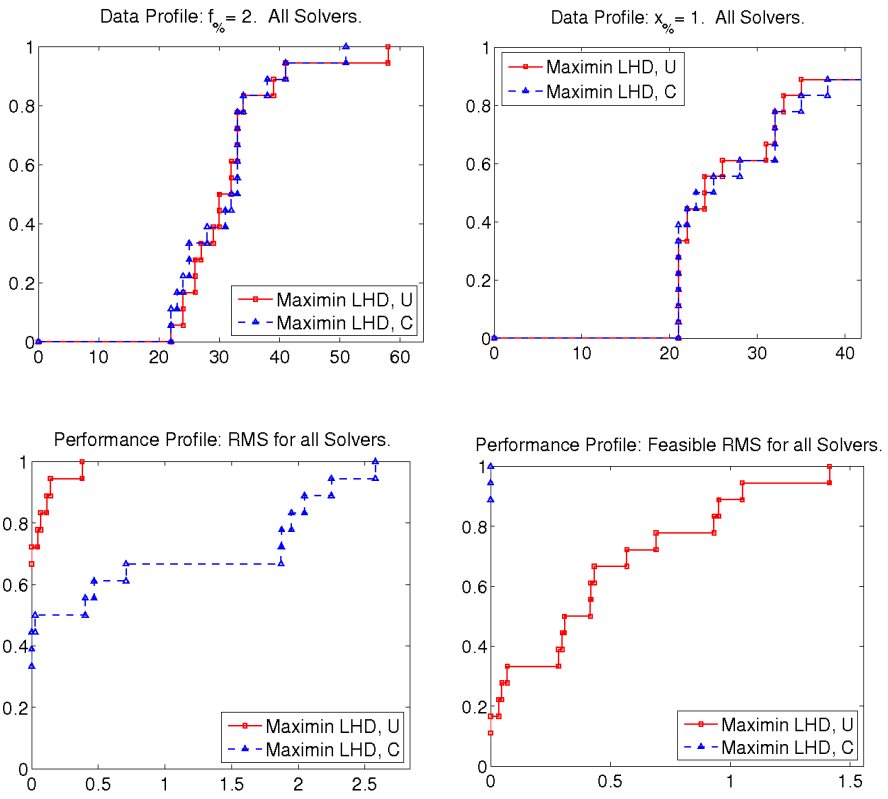


Figure 6: Constrained Maximin LHD versus standard Maximin LHD.

A possible reason for this is that although some points are infeasible, they still give information about the shape of the function. Since only feasible points are sampled by the CGO solvers, these initial infeasible points give extra stability to the surrogate model, compared to sampling only feasible points initially.

When considering the RMS metric for constrained problems, there are two ways to measure the error. One can look at the whole design space, like before, but it is also interesting to measure only the feasible space. As seen in the plots, these two results are in conflict. Using a fully feasible initial design naturally gives better RMS error when only considering the feasible design space, but not as good when measuring the whole design space.

6 Conclusions

The N_2 setting performed better for all experimental designs, so this is definitely good. One could of course try to start with even more points, but since the CGO solvers are constructed in a way where each new point is chosen carefully by utilizing information from all the sampled points, this is probably not a good idea.

Finding a feasible experimental design with space filling capacity is not easy. The algorithm proposed in this paper generates an initial design with feasible points having the structure of a Maximin LHD. To see any real effect of a fully feasible experimental design, one must probably have test problems where a large area of the design space is infeasible. Most of the problems in P_C have large feasible areas and thus the effect is not as noticeable. Also, as the number of initial points N is typically a small part of the total number of sampled points, the effect is limited.

Sampling the corner points of the bounding box add a tremendous stability to the solvers, one could think of it as pinpointing the corners of the surface and therefore getting a more stable description of the boundary. This feature is important as it tends to help the solvers sample more interior points, which often helps the convergence.

The Maximin LHD approach is superior when looking at the RMS error. Combining this with the success of the Corner Point Strategy seemed like a promising idea, but unfortunately did not improve the $f_{\%}$ and $x_{\%}$ metrics as we had hoped.

There is no obvious winner since all the experimental designs work satisfactory. But since we consider costly functions, even small differences do matter. The combination of Corner Points and global solver performs very well compared to the other experimental designs, with robust results for all metrics.

References

[1] E. D. Dolan, J. J. Moré, and T. S. Munson: Optimality Measures for Performance Profiles. *Preprint ANL/MCS-P1155-0504* (2004).

[2] H. M. Gutmann: A radial basis function method for global optimization. *Journal of Global Optimization* **19** (3), 201–227 (2001).

[3] K. Holmström: An adaptive radial basis algorithm (ARBF) for expensive black-box global optimization. *Journal of Global Optimization* **41**, 447–464 (2008).

[4] 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.

[5] D. R. Jones, C. D. Perttunen, and B. E. Stuckman: Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications* **79**, 157–181 (1993).

[6] D. R. Jones, M. Schonlau, and W. J. Welch: Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization* **13**, 455–492 (1998).

[7] D. R. Jones: DIRECT. *Encyclopedia of Optimization* (2001).

[8] J. J. Moré and S. M. Wild: Benchmarking Derivative-Free Optimization Algorithms. *Preprint ANL/MCS-P1471-1207* (2007).

[9] M. J. D. Powell: Recent Research at Cambridge on Radial Basis Functions. *New Developments in Approximation Theory*, 215–232 (2001).

[10] E. Stinstra, D. den Hertog, P. Stehouwer, and A. Vestjens: Constrained Maximin Designs for Computer Experiments. *Technometrics*, **45**, 340–346 (2003).

Paper III

This paper has been published as:

N-H. QUTTINEH and K. HOLMSTRÖM, Implementation of a One-Stage Efficient Global Optimization (EGO) Algorithm, *Research Report MDH*, 2, 2009.

Implementation of a One-Stage Efficient Global Optimization (EGO) Algorithm

Nils-Hassan Quttineh* and Kenneth Holmström*

Abstract Almost every Costly Global Optimization (CGO) solver utilizes a surrogate model, or response surface, to approximate the true (costly) function. The EGO algorithm introduced by Jones et al. utilizes the DACE framework to build an approximating surrogate model. By optimizing a less costly utility function, the algorithm determines a new point where the original objective function is evaluated. This is repeated until some convergence criteria is fulfilled. The original EGO algorithm finds the new point to sample in a two-stage process. In its first stage, the estimates of the interpolation parameters are optimized with respect to already sampled points. In the second stage, these estimated values are considered true in order to optimize the location of the new point. The use of estimate values as correct introduces a source of error. Instead, in the one-stage EGO algorithm, both the parameters and the location of a new point are optimized at the same time, removing the source of error. This new subproblem becomes more difficult, but eliminates the need of solving two subproblems. Difficulties in implementing a fast and robust One-Stage EGO algorithm in TOMLAB are discussed, especially the solution of the new subproblem.

Keywords: Black-box, Surrogate model, Costly functions, Latin Hypercube Designs, Experimental Design.

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.

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).

*Department of Applied mathematics, Mälardalen University, SE-721 23 Västerås, Sweden.

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.

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. These complicated problems are formulated as follows:

The Mixed-Integer Costly Global Black-Box Nonconvex Problem

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

where $f(x) \in \mathbb{R}$ and $x_L,\ x,\ x_U \in \mathbb{R}^d$. Matrix $A \in \mathbb{R}^{m_1 \times d}$, $b_L,\ b_U \in \mathbb{R}^{m_1}$; defines the m_1 linear constraints and $c_L,\ c(x),\ c_U \in \mathbb{R}^{m_2}$ defines the m_2 nonlinear constraints. The variables x_I are restricted to be integers, where $\mathbb{I}$ is an index subset of $\{1,\ldots,d\}$.

1.1 Surrogate Models

One approach for solving CGO problems is to utilize response surfaces. The basic idea is to start with an initial sample of points (experimental design), where the costly function values are calculated. A surrogate model (response surface) is built from the sampled points, using for example radial basis functions (RBF) or the DACE framework. Use this interpolated surface to approximate the true function, and decide a new point to sample by some algorithmic strategy. Then iterate until some convergence criteria is fulfilled.

Surrogate Model Algorithm (pseudo-code)

- ▷ Find initial set of $n \geq d+1$ points $\mathbf{x}$ using Experimental Design.
- ▷ Compute costly $f(x)$ for initial set of n points. Best point (x_{Min}, f_{Min}).
- ▷ Use the sampled points $\mathbf{x}$ to build a response surface model as an approximation of the $f(x)$ surface.
- ▷ Choose next point x_{n+1} to be added:
  - Decided by the algorithm used, like EGO, ARBF or rbfSolve.
  - Update best point (x_{Min}, f_{Min}) if $f(x_{n+1}) < f_{Min}$.
- ▷ Iterate until f_{Goal} achieved, $n > n_{Max}$ or maximal CPU time used.

2 Background to DACE and EGO

Suppose we want to predict the function value at a point $\bar{x}$ not already sampled. The DACE framework, short for "Design and Analysis of Computer Experiments" and also referred to as Kriging, is based on modeling a function as a realization of random variables, normally distributed with mean μ and variance σ^2.

The original EGO algorithm, introduced by Jones, Schonlau and Welch [7] in 1998, is a two-stage method. Such methods fit a response surface to sampled points in the first step, then utilize the surface to find new search points in the second step.

Some years later, Jones presents the idea of a one-stage approach [6]. Except for an implementation by Jones himself, the one-stage approach have not been explored.

2.1 The Correlation function

Before going into mathematical details and formulation of the algorithm, we introduce some notations and variables to be used throughout this paper. Compared with Euclidean distance, where every variable is weighted equally, the distance formula

$$D\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) \;=\; \sum_{k=1}^{d} \theta_k \cdot \left|x_k^{(i)} - x_k^{(j)}\right|^{p_k} \qquad \theta_k > 0 \;,\; p_k \in [1,2] \tag{2}$$

is designed to capture functions more precise. The parameter θ_k can be interpreted as a measure of the importance of variable x_k. Even small changes in x_k might lead to large differences in the function values at $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$.

The exponent p_k is related to the smoothness of the function in the k:th dimension. Values of p_k near 2 corresponds to smooth functions and values near 1 to less smoothness. Based on the distance formula (2), the correlation function

$$Corr\left[\epsilon(\mathbf{x}^{(i)}), \epsilon(\mathbf{x}^{(j)})\right] \;=\; e^{-D\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right)} \tag{3}$$

has all the intuitive properties one would like it to have. When the distance between $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ is small, the correlation is close to one. For large distances, the correlation approaches zero. A large value for θ will affect the distance to grow faster, which leads to a decrease in the correlation. In this way active variables are accounted for, giving a more accurate correlation function.

We denote the evaluated function values by $\mathbf{y} = \left(f(\mathbf{x}^{(1)}), \ldots, f(\mathbf{x}^{(n)})\right)'$, where n is the number of points sampled so far. Denote the matrix of correlation values by $\mathbf{R}$, where $\mathbf{R}(i,j) \;=\; Corr\left[\epsilon(\mathbf{x}^{(i)}), \epsilon(\mathbf{x}^{(j)})\right]$. Also, let $\mathbf{r}$ denote the vector of correlations between $\bar{x}$ and the sampled points, that is $r_i(\bar{x}) \;=\; Corr\left[\epsilon(\bar{x}), \epsilon(\mathbf{x}^{(i)})\right]$.

2.2 The DACE interpolation model

To build a surrogate model from the sampled points, we need to estimate parameters θ_k and p_k. This is done by choosing them to maximize the likelihood of the sampled points. A more detailed analysis is found in Section 3.

With the parameter estimates in hand, we are now able to construct the DACE model, or DACE response surface. Using the evaluated function values $\mathbf{y}$ and the matrix of correlation values $\mathbf{R}$, the DACE interpolant is defined by

$$y(\bar{x}) \;=\; \mu + \mathbf{r}'\mathbf{R}^{-1}(\mathbf{y} - \mathbf{1}\mu) \tag{4}$$

where $\mathbf{r}$ is the vector of correlations between $\bar{x}$ and the sampled points $\mathbf{x}$. The first term μ is the estimated mean, and the second term represents the adjustment to this prediction based on the correlation of sampled points $\mathbf{x}$. The derivation of this predictor can be found in [10].

The mean square error of the predictor, denoted by $s^2(\bar{\mathbf{x}})$, is given by

$$s^2(\bar{x}) \;=\; \sigma^2 \cdot \left[1 - \mathbf{r}'\mathbf{R}^{-1}\mathbf{r} + \frac{(1 - \mathbf{1}'\mathbf{R}^{-1}\mathbf{r})^2}{(\mathbf{1}'\mathbf{R}^{-1}\mathbf{1})}\right]. \tag{5}$$

As one would imagine, the mean square error for a sampled point is zero, $s^2(\mathbf{x}^{(i)}) = 0$.

We now have a formula for the expected value of a point yet to be sampled, along with an error estimation. Since this predictor is cheap to calculate, compared to the costly function, it can be used to locate a new point $\bar{x}$ with as low expected function value as possible.

3 The EGO algorithm

The Efficient Global Optimization (EGO) algorithm utilizes the DACE interpolation surface to approximate the costly function based on already sampled points. In order to use this for optimization, one must find a way to iteratively choose x^*, the next point to sample.

In the original EGO algorithm, described in Section 3.1, the Maximum Likelihood Estimation (MLE) of the parameters θ and p is based on the set of sampled points $\mathbf{x}$. These estimates are used to decide upon new sample points, found by optimizing some utility function, which hopefully converges towards the global optimum. This can be seen as a two-stage process.

A drawback with this approach is that the utility function depends on the estimated parameters, which might lead to inaccurate decisions. To overcome this flaw, the one-stage process described in Section 3.2 incorporates x^* into the MLE step. For a given target value f^*, the MLE tries to fit the surface to already sampled points, conditional upon the assumption that the surface goes through the point (x^*, f^*).

3.1 Two-stage process, Standard EGO

We need to estimate the parameters θ_k and p_k in order to construct an interpolation surface of our costly function. The estimates are found by choosing them to maximize the likelihood of the already sampled points $\mathbf{x}$.

The likelihood function

$$L(\theta, p) \;=\; \frac{1}{(2\pi)^{n/2}(\sigma^2)^{n/2}|\mathbf{R}|^{\frac{1}{2}}} \cdot e^{-\frac{(\mathbf{y}-\mathbf{1}\mu)'\mathbf{R}^{-1}(\mathbf{y}-\mathbf{1}\mu)}{2\sigma^2}} \tag{6}$$

Note that the dependence on parameters θ and p is via the correlation matrix $\mathbf{R}$. Assuming we know the values of θ and p, we find the values of μ and σ^2 that maximizes the log-likelihood function:

$$LL(\theta, p) \;=\; -\frac{n}{2}\log(\sigma^2) - \frac{1}{2}\log(|\mathbf{R}|) - \frac{(\mathbf{y}-\mathbf{1}\mu)'\mathbf{R}^{-1}(\mathbf{y}-\mathbf{1}\mu)}{2\sigma^2} \tag{7}$$

By taking the derivatives of (7) with respect to μ and σ^2 respectively, solving them equal to zero, the solution is:

$$\hat{\mu} \;=\; \frac{(\mathbf{1}'\mathbf{R}^{-1}\mathbf{y})}{(\mathbf{1}'\mathbf{R}^{-1}\mathbf{1})} \tag{8}$$

and

$$\hat{\sigma}^2 \;=\; \frac{(\mathbf{y}-\mathbf{1}\hat{\mu})'\mathbf{R}^{-1}(\mathbf{y}-\mathbf{1}\hat{\mu})}{n} \tag{9}$$

Substituting equations (8) and (9) back into (7), one finds the concentrated log-likelihood function only depending on parameters θ and p via $\mathbf{R}$:

$$ConLL(\theta, p) \;=\; -\frac{n}{2}\log(\hat{\sigma}^2) - \frac{1}{2}\log(|\mathbf{R}|) \tag{10}$$

Maximizing (10) yields the estimates needed. Then use equations (8) and (9) to calculate the estimates of μ and σ^2.

Once the estimates are known, a utility function is optimized in order to find x^*, the next point to sample. There are many different utility functions that could be used, but the Expected Improvement (ExpI) is most commonly used.

The Expected Improvement relies on the predicted values of μ and σ^2 to find the location x^* where the probability of improving the objective value is maximized. For details on different utility functions and the Expected Improvement, see [11].

3.2 One-stage EGO, new approach

In his paper [6], Jones introduces the idea of a one-stage EGO algorithm, incorporating the new point x^* into the estimation process of parameters θ and p. The estimates are, like before, found by choosing them to maximize the likelihood of the sampled points $\mathbf{x}$. But this time we compute the likelihood of the observed data conditional upon the assumption that the surface goes through the point (x^*, f^*).

The conditional likelihood function

$$CL(\theta, p, x^*) \;=\; \frac{1}{(2\pi)^{n/2}(\sigma^2)^{n/2}|\mathbf{C}|^{\frac{1}{2}}} \cdot e^{-\frac{(\bar{\mathbf{y}} - \bar{\mathbf{r}}\mu)'\mathbf{C}^{-1}(\bar{\mathbf{y}} - \bar{\mathbf{r}}\mu)}{2\sigma^2}} \tag{11}$$

where

$$\mathbf{C} = \mathbf{R} - \mathbf{r}\mathbf{r}' \;, \quad \bar{\mathbf{y}} = \mathbf{y} - \mathbf{r} \cdot f^* \;, \quad \bar{\mathbf{r}} = \mathbf{1} - \mathbf{r}.$$

The dependence on the parameters θ, p and x^* is via the conditional correlation matrix $\mathbf{C}$, as vector $\mathbf{r}$ describes correlation between x^* and the n sampled points. Both $\mathbf{R}$ and $\mathbf{r}$ are affected by θ and p.

When using the conditional log-likelihood to evaluate the hypothesis that the surface passes through (x^*, f^*) we also estimate the values of θ and p, and like before find the values of μ and σ^2 that maximizes the conditional log-likelihood function:

$$CLL(\theta, p, x^*) \;=\; -\frac{n}{2}\log(\hat{\sigma}^2) - \frac{1}{2}\log(|\mathbf{C}|) - \frac{(\bar{\mathbf{y}} - \bar{\mathbf{r}}\mu)'\mathbf{C}^{-1}(\bar{\mathbf{y}} - \bar{\mathbf{r}}\mu)}{2\sigma^2} \tag{12}$$

with optimal values for μ and σ:

$$\hat{\mu} \;=\; \frac{(\bar{\mathbf{r}}'\mathbf{C}^{-1}\bar{\mathbf{y}})}{(\bar{\mathbf{r}}'\mathbf{C}^{-1}\bar{\mathbf{r}})} \tag{13}$$

and

$$\hat{\sigma}^2 \;=\; \frac{(\bar{\mathbf{y}} - \bar{\mathbf{r}}\hat{\mu})'\mathbf{C}^{-1}(\bar{\mathbf{y}} - \bar{\mathbf{r}}\hat{\mu})}{n} \tag{14}$$

Substituting equations (13) and (14) back into (12), we find the concentrated form of the conditional log-likelihood function, depending on parameters θ, p and x^* via matrix $\mathbf{C}$ and vectors $\mathbf{r}$ and $\bar{\mathbf{y}}$:

$$ConCLL(\theta, p, x^*) \;=\; -\frac{n}{2}\log(\hat{\sigma}^2) - \frac{1}{2}\log(|\mathbf{C}|) \tag{15}$$

Maximizing (15) yields the estimates needed. Then use equations (13) and (14) to calculate the estimates of μ and σ^2.

To illustrate the nasty nature of (15), consider the equivalent formulation of the full Conditional Maximum Likelihood (CML) problem. Variable S and matrix $\mathbf{C}$ are both dependent on all other variables in a complicated manner. Matrix $\mathbf{R}$ and vector $\mathbf{r}$ depends on variables x^*, θ and p, which also affects vectors $\bar{\mathbf{r}}$ and $\bar{\mathbf{y}}$.

$$\min_{\theta,p,x^*} \quad n \cdot \log(S) + \log(|\mathbf{C}|) \qquad \textbf{[CML]}$$

$$\begin{aligned}
s.t. \qquad S &= \tfrac{1}{n} \cdot \left[(\bar{\mathbf{y}} - \bar{\mathbf{r}} \cdot \mu)' \cdot \mathbf{C}^{-1} \cdot (\bar{\mathbf{y}} - \bar{\mathbf{r}} \cdot \mu)\right] \\
\mu &= \frac{(\bar{\mathbf{r}}' \cdot \mathbf{C}^{-1} \cdot \bar{\mathbf{y}})}{(\bar{\mathbf{r}}' \cdot \mathbf{C}^{-1} \cdot \bar{\mathbf{r}})} \\
\mathbf{C} &= \mathbf{R} - \mathbf{r} \cdot \mathbf{r}' \\
\bar{\mathbf{y}} &= \mathbf{y} - \mathbf{r} \cdot f^* \\
\bar{\mathbf{r}} &= 1 - \mathbf{r} \\
\mathbf{R}(i,j) &= \exp\left(-\sum_{k=1}^{d} \theta_k \cdot \left|x_k^{(i)} - x_k^{(j)}\right|^{p_k}\right) \qquad i,j = 1,\ldots,n \\
\mathbf{r}(i) &= \exp\left(-\sum_{k=1}^{d} \theta_k \cdot \left|x_k^{(i)} - x_k^{*}\right|^{p_k}\right) \qquad i = 1,\ldots,n \\
0 &< \theta_k \qquad\qquad\qquad\qquad k = 1,\ldots,d \\
1 \leq p_k &< 2 \qquad\qquad\qquad\qquad k = 1,\ldots,d \\
x_L \leq x^* &\leq x_U \\
b_L \leq Ax^* &\leq b_U \\
c_L \leq c(x^*) &\leq c_U
\end{aligned}$$

Parameters :

f^*	given target value
d	dimension of problem
n	number of sampled points
$\mathbf{x}$	sampled points
$\mathbf{y}$	evaluated function values

3.3 Overview

Based on the DACE framework, we have presented two different approaches on how to perform iterations in order to find new sample points x^*, hopefully converging towards the global optimum. Each method is connected to good properties as well as some troublesome issues.

Two-Stage EGO

Stage 1. Find θ and p by MLE, interpolate surface.

Stage 2. Optimize some utility function to find x^*, a new point to sample.

\+ Two separate subproblems, easier to solve.

\- Relies on estimated parameters.

One-Stage EGO

Stage 1. Given a value for f^*, find θ, p and x^* by MLE, conditionally that the surface goes through (x^*, f^*).

\+ Only one subproblem to solve. The computation of x* is not dependent on previous biased estimates, more accurate computation.

\- We don't know the value of f^*. The new CML problem is more difficult.

In the upcoming Sections 4 and 5, we look deeper into the problems connected to a one-stage process and present methods to handle these issues.

4 Difficulties and Algorithm description

Although the one-stage process is theoretically more appealing, there are some practical issues that needs to be adressed. To start with, the optimal value f^* is obviously not known in advance and must be chosen in some way. This is the least of our troubles though, and methods to deal with this is presented in Section 4.1.

Section 5 is devoted to ideas on how to solve the challenging CML problem in a robust and efficient manner. This is essential since the subproblem needs to be solved multiple times each iteration.

There is also some numerical issues connected to the subproblem. When optimizing the concentrated conditional log-likelihood function (15) one need to evaluate points close to already sampled points $\mathbf{x}$, and this causes the function to collapse. Details and remedies are presented in Section 4.2.

Finally, a pseudo-code for the one-stage EGO algorithm is presented in Section 4.3.

4.1 Finding f^* values

The one-stage approach finds a new point x^* by computing the likelihood of the observed data conditional upon the assumption that the surface goes through the point (x^*, f^*). The value of f^* is not known in advance and must be chosen somehow.

Fortunately, the use of a target value is not unique for the one-stage EGO algorithm. When Gutmann introduced the radial basis function (RBF) method [3], he proposed a cycle of 5 target values lower than the minimum of the interpolated surface, ranging from far below (global search) to close below (local search). Set the value of f^* to

$$f_k^* \;=\; s_{min} - w_k \cdot \Big(\max_i f(x_i) - s_{min}\Big)$$

where s_{min} is the minimum of the interpolated surface. The weight factor w_k is defined using a cyclic scheme over k like

$$w_k \;=\; (1 - k/N^2), \qquad\qquad k = 0, 1, \ldots, N-1.$$

This is successfully done in TOMLABs CGO-solver `rbfSolve` [1].

A more powerful approach, but also more computer intensive, is to solve the subproblem for a wide range of f^* values every iteration. Each solution gives an x^* candidate, so how to proceed? It turns out that these solutions tend to cluster, and by applying a clustering process one could proceed with 1-3 new x^* values each iteration. This is successfully done in TOMLABs CGO-solver `ARBFMIP`, and details on the target values and the clustering process are found in [4] and [5].

Both methods overcome the problem of not knowing the optimal target value f^* in advance, but experience from solving a large number of test problems with `rbfSolve` and `ARBFMIP` suggests that using a range of target values adds robustness to the optimization process.

Another advantage of getting a cluster of new points is the possibility to benefit from parallel calculations, becoming more and more standard for computers today. Hence our implementation of the one-stage EGO algorithm will adopt the clustering methods of `ARBFMIP`.

4.2 Numerical Issues

The EGO algorithm is known to suffer from numerical problems due to the correlation matrix becoming more and more ill-conditioned as sampled points start to cluster in promising areas of the sample space. Equation (10) used for the MLE of parameters θ and p includes the logarithm of the determinant of the correlation matrix $\mathbf{R}$, which approaches zero as points are sampled close together.

For the one-stage process, the CML function (15) is optimized, and the numerical issues have not disappeared. The situation is even worse due to the intricate relations between sampled points and parameters θ and p, now also affected by parameters x^* and f^*. The problem with a rank deficient $\mathbf{C}$ matrix is that when evaluating (15) its inverse is needed (although not calculated implicitly, it is still problematic). It also contains the term $\log(|\mathbf{C}|)$, which is undefined as the determinant becomes 0. We now present three situations where numerical issues arise in the one-stage approach.

Solving the MLE

Numerical issues arise when maximizing (10), the standard MLE, to find parameters θ and p. The upper bound for parameter p is theoretically 2, but Sasena reports in [11] that a value strict less than two is more numerical stable. Here is an example clearly supporting the claim.

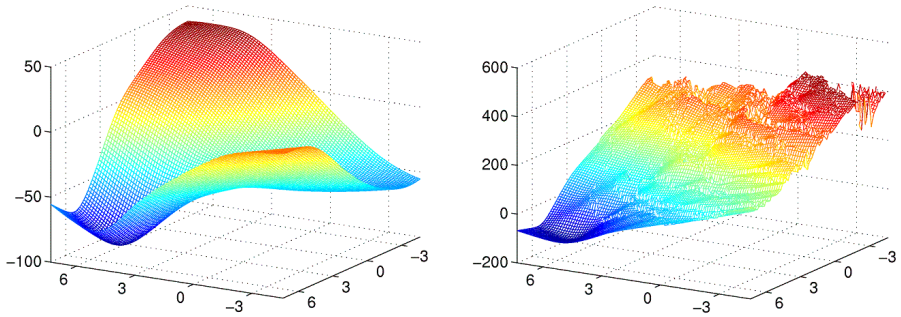


Figure 1: MLE of θ and p. To the left, $p = 1.99$ and to the right $p = 2$.

Figure 1 shows the same ML function to be minimized, but with the parameter p fixed to 1.99 and 2 respectively. Clearly the function where $p = 1.99$ is preferable, motivating the upper bound of p to be adjusted to 1.99.

Evaluating CML close to sampled points

Ill-conditioning of the correlation matrix is inherited from the two-stage approach, but also enhanced due to the definition of correlation matrix $\mathbf{C} = \mathbf{R} - \mathbf{r}\mathbf{r}'$. The correlation vector $\mathbf{r}(i)$ approaches 1 for $x^{(i)}$s close to x^*, hence creating a close-to-zero row and column in $\mathbf{C}$ whenever evaluating points close to $\mathbf{x}$, the set of so far sampled points.

In evaluating a point really close to $\mathbf{x}$, the logical thing would be for the CML function (15) to approach minus infinity. It is certainly unlikelihood for an already sampled point, with a function value not equal to f^*, to suddenly match f^*. But when zooming in on the CML function in a very small interval around a sampled point, we get something else. Figure 2 illustrates the phenomenon, showing pictures of (15) close to a sampled point. The CML functions are displayed as minimization problems.

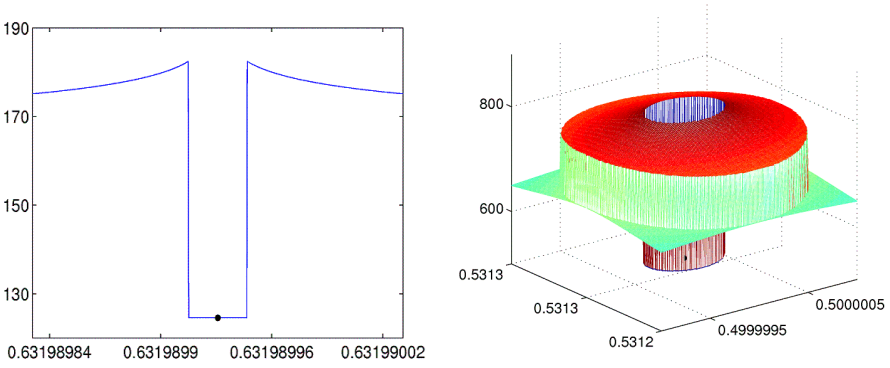


Figure 2: Evaluating CML close to an already sampled point. The function should increase continuously, but suddenly drop to a constant value.

So what happens when matrix $\mathbf{C}$ becomes very close to singular, due to reasons explained before, and we try to evaluate (15). In our implementation, first the QR-factorization of matrix $\mathbf{C}$ is found, then its determinant is calculated as the product of the diagonal elements of R which are greater than a specified lower bound ϵ, i.e.

$$\det \mathbf{C} = \prod_{i \in I} R(i,i) \quad \text{where} \quad I = \{i : |R(i,i)/R(1,1)| > \epsilon\}.$$

This pseudo-rank determinant causes the "valley" surrounding the sampled point, clearly seen in the one-dimensional example. The width of the valley depends on the value of ϵ, becoming more narrow as ϵ goes to zero. So by using a smaller value of ϵ than normally necessary, we at least decrease the critical region. To compensate for the pseudo-rank, we subtract a big number to the objective for all points inside the valley.

It is important to clarify the need of evaluating points close to already sampled points. If a function value $\mathbf{y}(i)$ corresponding to a sampled point $\mathbf{x}^{(i)}$ is much larger than f_{min}, the best found function value so far, it is not desirable to sample points close to $\mathbf{x}^{(i)}$. On the other hand, in search for a new point x^* that improves the objective value, it is natural to evaluate points close to sampled points with low function values. In order to find solutions with many digits of accuracy, it is even necessary.

Bad combinations

After some iterations, when having sampled some points, another numerical issue arises when maximizing both (10) and (15). In contrast to the previous problem, this is due to a very natural cause. Independent of the optimization method used to solve the MLE, many combinations of parameters θ and p are tested, and some of them are simply not feasible for the given set of sampled points $\mathbf{x}$.

Although the reasons are completely different, the result is the same as before. When matrix $\mathbf{C}$ becomes singular and its determinant is zero, the CML function returns minus infinity when taking the logarithm, causing the optimization process to halt. A numerical example is presented to the left in Figure 3.

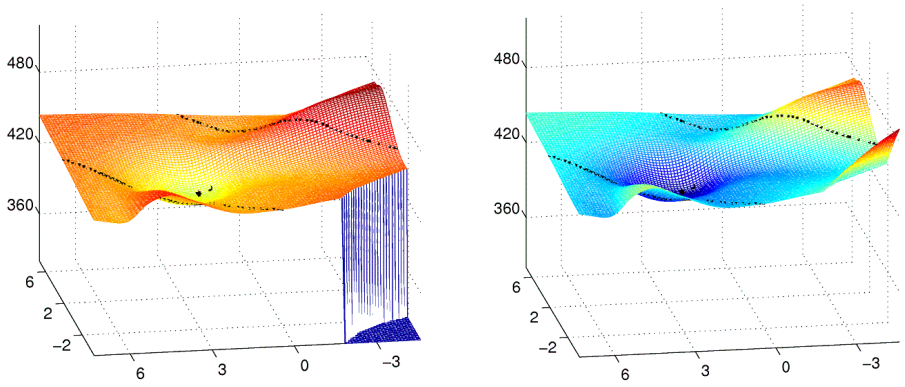


Figure 3: MLE of θ for a fixed value of p. To the left, the ML function without safeguard. To the right, the same ML function with safeguarded logarithm calculations.

To handle this situation, we safeguard the calculations of the logarithm. Whenever the determinant is zero, return a large negative value instead of calculating the logarithm. By investigation, MATLAB evaluates $\log(10^{-323})$ as -743.7469 but a smaller value like $\log(10^{-324}) = -\infty$, so whenever $|\mathbf{C}| < 10^{-323}$ we replace the term $\log(|\mathbf{C}|)$ by -743.75.

As seen in Figure 3, this removes the discontinuity caused by the determinant calculations breaking down. The CML function will act linear in the infeasible area. This might not be entirely satisfactory, but it prevents the optimization process from breaking down which is the important thing.

4.3 Pseudo Code

When implementing the one-stage EGO, the algorithmic structures are coded in MATLAB but all heavy calculations are implemented in Fortran and C code, and interfaced using so called mex file interfaces. Here follows a description of the one-stage EGO algorithm together with a pseudo-code presented in Algorithm 1.

After finding the initial set of n sample points, we estimate parameters θ and p by optimizing (10) and build an interpolation surface. We do this in order to find the surface minimum s_{min}, used to define the range of target values F. Then begins the process of solving the Conditional Maximum Likelihood (CML) for each target value $f_k^* \in F$, cluster the result and pick new points to sample. One could optionally add s_{min} as well, since the idea of the interpolation surface is to approximate the true costly function. This is repeated until the global optimum is found, or the maximum number of function evaluations is reached.

Our MATLAB implementation is named `osEGO`.

Algorithm 1 The one-stage EGO algorithm

```
Find n ≥ d + 1 initial sample points using some Experimental Design.
while n < MAXFUNC do
    Estimate parameters θ and p by optimizing the ML-function (10).
    Use θ and p to build the interpolation surface s(x).
    Find the minimum of the surface, denote it s_min.
    Define F, a range of f*-values.
    for all f*_k ∈ F do
        Maximize the Conditional ML (15) defined by f*_k.
    end for
    Apply a clustering process on the results, add new points and update.
    Optionally add s_min and update.
end while
```

5 The CML problem

Instead of solving two consecutive subproblems, the one-stage EGO combines all work into a single subproblem. This eliminates the use of estimated values for parameters θ and p, the main drawback of the original two-stage EGO algorithm. But incorporating x^* into the MLE complicate things. Not only is the dimension of the subproblem increased by $d+1$, we need to guess the value of f^*.

In order to solve this new, more complicated, conditional MLE problem efficiently for a whole range of f^* values, we need to find ways of generating good initial values for our parameters x^*, θ and p, and then solve the full problem. We do this by solving a series of restricted subproblems.

One might argue that in this way we still solve more than one subproblem, hence not gaining anything compared to the two-stage EGO algorithm. But the point is that while standard EGO first approximates the interpolation surface to already sampled points and then search for x^* in a separate problem, we do not separate x^* from the MLE of θ and p although solving the CML problem in several steps.

Subproblem 1

To get started, we solve a restricted version of CML where p_k is fixed to 1.99 for all k, and both x^* and θ are considered univariate, i.e. $\theta = \theta_1 = \theta_2 = \ldots = \theta_d$ and $x^* = x_1^* = x_2^* = \ldots = x_d^*$. The most difficult variables are x^*, so by scanning the space using a parametrization we find a suitable initial value of x^*. Notice that Subproblem 1 will always be a two-dimensional problem independent of the original problem. The peaks seen in Figure 4 on page 84 correspond to $\mathbf{x}$ lying close to the diagonal. To solve the subproblem, use starting points in between the peaks.

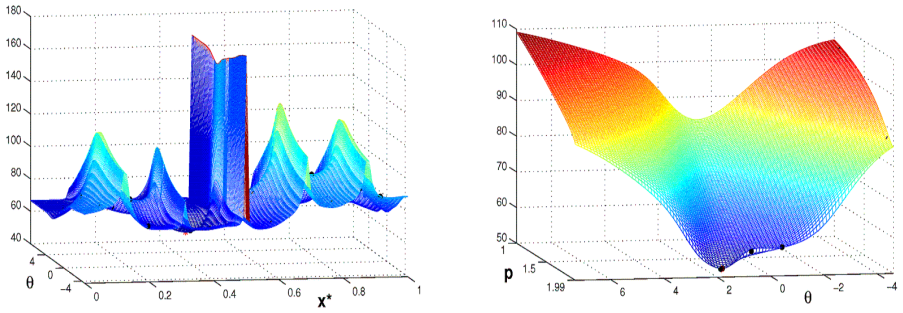


Figure 4: To the left, subproblem 1 with univariate x^* and θ for $p = 1.99$ fixed. To the right, subproblem 2 with univariate θ and p, using x^* found in subproblem 1.

Subproblem 2

Having found an initial guess x_0^*, we progress by keeping x^* fixed and consider univariate values for θ and p instead, solving another two-dimensional problem. From Subproblem 1 we get a good initial guess for θ, which makes Subproblem 2 relatively easy to solve. In Figure 4, to the right, a numerical example.

Subproblem 3

At this point, we have univariate initial values for all parameters. From solving Subproblem 1 we get values for x_0^* and θ_0, and the latter is refined in Subproblem 2 together with p_0. We are now ready to solve the full subproblem, with no parameterizations or fixations. The solution of CML, which is defined by the current target value f_k^*, is a suggestion for a new point to sample, denoted x_k^*. Figure 5 and Figure 6 show pictures of the CML problem, displayed as a maximization problem for best possible visualization.

Resolving for a new f_k^*.

Since we need to solve CML for $f_k^* \in F$, where F denotes a range of target values for the current iteration, one should exploit the possibility of an efficient reoptimization process. We notice that a relative small change in the value of f_k^* results in a very similar problem to solve. Therefore, if $f_k^* - f_{k+1}^*$ is small, the solution x_k^* serves as a good initial guess for x_{k+1}^*. In this way, by keeping track of when consecutive values of f_k^* differ a lot, it suffices to solve all subproblems only a few times. Whenever the change in f^* value is relatively small, solve the full problem directly using the previous solution as starting point. This will speed up the solution process significantly.

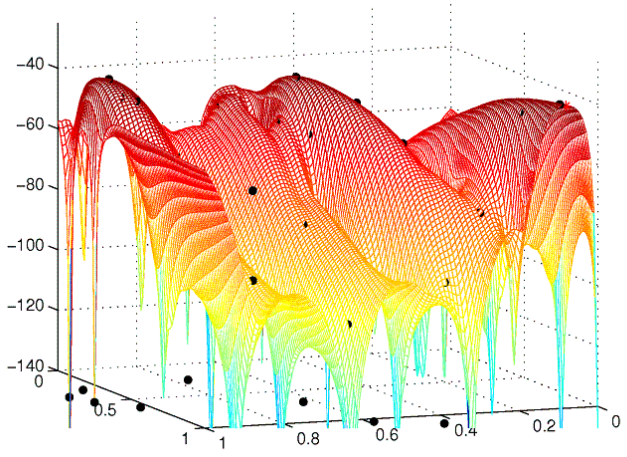


Figure 5: The full CML subproblem for fixed values of parameters θ and p. At this stage, the number of sampled points n is relatively small. Black dots indicate starting points for the solver used to maximize CML.

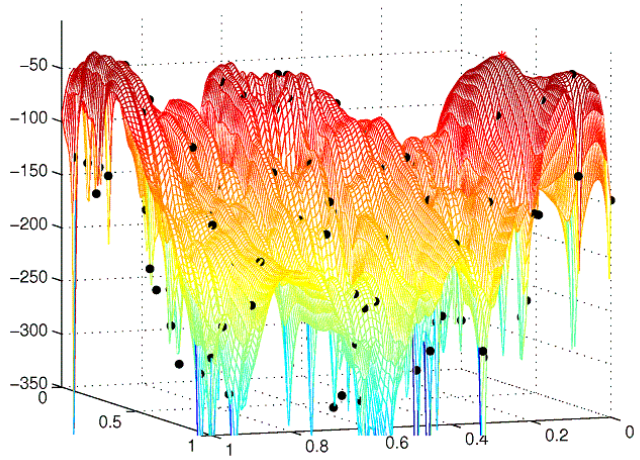


Figure 6: The full CML subproblem for fixed values of parameters θ and p. The number of sampled points n is much larger than before, and the problem has become much more difficult. Black dots indicate starting points used by the solver.

Modifications

It is possible to solve a restricted version of CML and accept the solution as x_k^* without solving the full subproblem. The most straightforward way is to consider univariate values for θ and p throughout the whole algorithm, hence decreasing the problem dimension from $3 \cdot d$ to $d + 2$. This will most likely be necessary when attempting to solve higher dimensional problems.

Experience over the years using the EGO algorithm also suggests that keeping p fixed to a single value at all times, eliminating this parameter completely from the optimization process, does not affect the results significantly. A value close to 2 is suggested, and from the earlier discussion in Section 4.2 on numerical issues, an upper bound of $p = 1.99$ is strongly recommended.

6 Benchmark and Tests

This paper aims at evaluating the one-stage EGO implementation `osEGO` by solving a set of test problems and compare the results with other algorithms. Three solvers from the TOMLAB/CGO environment are used. The `rbfSolve` and `arbfmip` solvers utilize radial basis functions, and the `EGO` solver is an implementation of the standard two-stage EGO algorithm.

All solvers run with their default parameter settings, controlling algorithmic options like variable scaling and the choice of merit function.

Following the definition of Dolan and Moré [2], a benchmark consists of a set P of benchmark problems, a set S of optimization solvers and a well defined convergence test. Since the problems are considered costly, we define a performance measure $t_{s,p} > 0$, the number of function evaluations required for problem $p \in P$ to converge using solver $s \in S$.

All solvers are set to break after 200 function evaluations or earlier if convergence to the known global optimum is obtained. The relative error is defined as

$$E_r = \frac{f_{min} - f_{opt}}{|f_{opt}|},$$

where f_{min} is the currently best feasible function value and f_{opt} is the known global optimum. Convergence is assumed if the relative error E_r is less than 10^{-4}. When $f_{opt} = 0$, stop when f_{min} is less than 10^{-4}. If convergence is not reached after 200 iterations, declare failure.

All CGO solvers need an initial set of points, or experimental design, in order to start the algorithm. Since the behavior of any such algorithm often depends heavily on this set, we solve each problem for a set of experimental designs E, summarized in Table 1 on page 87.

We consider two kinds of test problems in this benchmark, unconstrained box-bounded problems P_U and constrained problems P_C. A thorough presentation of the problems is found in Section 6.2, summarized in Table 2 on page 88.

6.1 Experimental Designs

Here follows a short description of the five different experimental designs (ExD) used in this benchmark. All but one design are defined by the number of initial points to sample, denoted by N. Some of them are able to handle constraints, while others can do this optionally.

The Corner Point Strategy (CPS) generates a fixed number of initial sample points, one for each corner point of the bounding box. This method is not able to handle constraints. The Deterministic Global Solver (DGS) approach applies the DIRECT algorithm to find N initial points, and is able to handle constraints.

The Maximin LHD strategy apply a Latin Hypercube Design of any given size N, where the points are separated subject to the maximin distance. This method could optionally handle constraints, using a method by Quttineh presented in [9]. The LHD designs used are taken from the webpage `http://www.spacefillingdesigns.nl` where a large collection of optimal maximin designs are available.

Finally, we also consider a combination of the CPS strategy with the Maximin LHD and DGS methods respectively. The $N = 2^d$ corner points of the bounding box are added to the designs generated by the Maximin LHD and DGS methods.

Table 1: The set of Experimental Designs (E). Five different ExD methods are listed, together with the available options. The Maximin LHD method can handle constraints optionally, hence $E_C = 4$ combinations for the constrained problems.

ExD	Experimental Design	Size of N	Constraints	E_U	E_C
CPS	Corner Points	Fixed	No	1	1
DGS	DG Solver	N_1 and N_2	Yes	2	2
LHD	Maximin LHD	N_1 and N_2	Yes/No	2	4
CP+DGS	Corners + DGS	N_1 and N_2	Yes	2	2
CP+LHD	Corners + LHD	N_1 and N_2	Yes/No	2	4

For all designs, except the CPS strategy, we use the settings $N_1 = (d+1)\cdot(d+2)/2$ and $N_2 = 10\cdot d + 1$, where d is the dimension of the problem to be solved. For the Maximin LHD design, it is possible to choose whether constraints should be considered or not. Therefore, each constrained problem will be solved in total 4 times using the Maximin LHD design, with and without constraints taken into account and for the options N_1 and N_2.

To summarize, we solve each box-bounded problem using nine different designs, nine being the sum of column E_U in Table 1. The constrained problems are solved using 13 different designs, the sum of column E_C. For a thorough description on the different experimental designs, see [9].

6.2 Test Problems

We define a set of 13 box-bounded unconstrained problems P_U, each solved for E_U different designs, and a set of 6 constrained problems P_C, each solved for E_C different designs. Most of them are 2-dimensional, except a few problems in 3 dimensions.

Table 2 and Table 3 give a compact description of the test problems. Column d is the number of variables, Ax the number of linear inequality constraints and $c(x)$ the number of nonlinear inequality constraints. In the *Domain* column, the lower and upper bounds for all variables are shown. The *Range* column shows the order of the objective.

None of the test problems have a global minimum in a corner point or midpoint.

Table 2: The set of box-bounded test problems P_U.

Nr.	Problem Name	d	Domain	Range
B4	Hartman 3	3	$[0,0,0] - [1,1,1]$	$4{\cdot}10^{0}$
B6	Branin RCOS	2	$[-5,0] - [10,15]$	$3{\cdot}10^{2}$
B7	Goldstein and Price	2	$[-2,-2] - [2,2]$	$1{\cdot}10^{6}$
B8	Six-Hump Camel	2	$[-3,-2] - [3,2]$	$2{\cdot}10^{2}$
B16	Shekels foxholes 2	2	$[0,0] - [10,10]$	$1{\cdot}10^{1}$
B19	Michalewiczs function 2	2	$[0,0] - [\pi,\pi]$	$2{\cdot}10^{0}$
B30	Myers smoothly fluctuating	2	$[-0.5,-0.5] - [3.5,3.5]$	$6{\cdot}10^{0}$
B32	LOG-Goldstein and Price	2	$[-2,-2] - [2,2]$	$1{\cdot}10^{1}$
B53	Dixon and Price	2	$[-10,-10] - [10,10]$	$9{\cdot}10^{5}$
L20	M20	2	$[0,0] - [5,5]$	$3{\cdot}10^{0}$
L25	M25	2	$[-2,-2] - [2,2]$	$1{\cdot}10^{0}$
L28	M28	2	$[3,3] - [9.99,9.99]$	$1{\cdot}10^{2}$
L49	M39	2	$[-500,-500] - [500,500]$	$2{\cdot}10^{3}$

Table 3: The set of constrained test problems P_C.

Nr.	Problem Name	d	Ax	$c(x)$	Domain	Range
C2	Gomez 3	2	0	1	$[-1,-1] - [1,1]$	$4{\cdot}10^{0}$
C3	Hock-Schittkowski 59	2	0	3	$[0,0] - [75,65]$	$1{\cdot}10^{2}$
C4	Hock-Schittkowski 65	3	0	1	$[-4.5,-4.5,-5] - [4.5,4.5,5]$	$2{\cdot}10^{2}$
C13	Schittkowski 343	3	0	2	$[0,0,0] - [36,5,125]$	$2{\cdot}10^{2}$
C19	Bump 2	2	1	1	$[\epsilon,\epsilon] - [10,10]$	$1{\cdot}10^{0}$
C22	HGO 468:1 + constraint	2	0	1	$[0,0] - [1,1]$	$6{\cdot}10^{0}$

6.3 Numerical Results

To present the benchmark results in a standardized manner, we utilize data profiles suggested by Moré and Wild [8], a kind of probability density function. Since function evaluations are expensive we are interested in the percentage of problems solved to a given accuracy within k function evaluations. Data profiles are designed to handle this, and are defined for a set of test problems P and a solver $s \in S$ by

$$d_s(k) = \frac{1}{|P|} \cdot \left| \left\{ p \in P : \frac{t_{p,s}}{n_p + 1} \leq k \right\} \right|$$

where n_p is the number of variables for each problem $p \in P$.

Our benchmark is defined by the set S of CGO solvers `rbfSolve`, `arbfmip`, `ego`, and `osEgo`, and the set of test problems defined by $\bar{P} = \{P \times E\}$. Since the choice of initial set of sample points have such a big impact on the performance of CGO solvers, each combination of test problem $p \in P$ and experimental design $e \in E$ is considered as a unique problem. In this way, the data profiles will show if a solver is robust or not with respect to the initial set of sample points.

Unconstrained problems

Figure 7 presents data profiles for the set of unconstrained box-bounded problems $\bar{P}_U = \{P_U \times E_U\}$. It consists of all combinations of the 13 box-bounded problems and 9 experimental designs, i.e. $|\bar{P}_U| = 117$. The tolerance of the relative error E_r is set to 1% in the left picture and to 0.01% in the right picture.

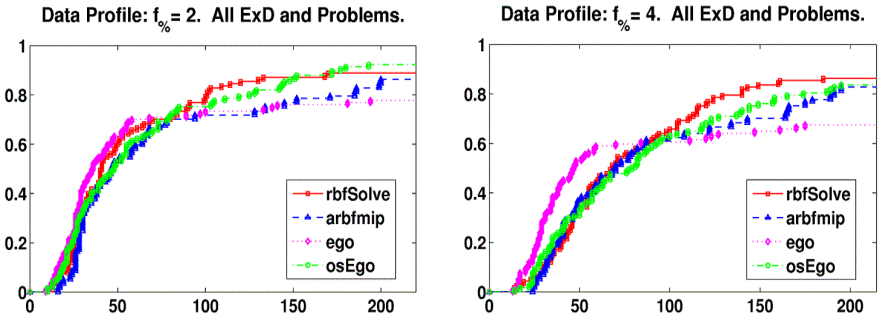


Figure 7: Data profiles for the box-bounded unconstrained problems $\bar{P}_U$ show the percentage of problems solved as a function of k, the number of function evaluations needed to converge, with tolerance 1% and 0.01%.

The `osEgo` algorithm is doing quite well, solving more problems to 2 digits of accuracy than any other solver in S. In finding 4 digits of accuracy, it is beaten by the `rbfSolve`.

Constrained problems

Figure 8 presents data profiles for $\bar{P}_C = \{P_C \times E_C\}$, the set of constrained box-bounded problems. It consists of all combinations of the 6 constrained problems and the 13 experimental designs, i.e. $|\bar{P}_C| = 78$. The tolerance is set to 1% in the left picture and to 0.01% in the right picture. All solvers perform very well on this set of constrained problems, finding the global optimum for almost all instances.

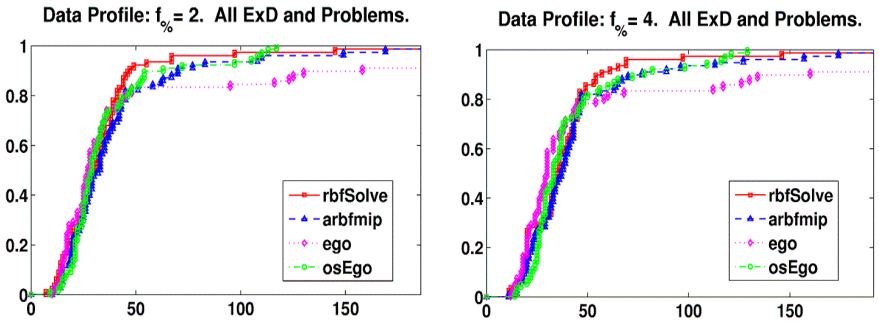


Figure 8: Data profiles for the constrained box-bounded problems $\bar{P}_C$ show the percentage of problems solved as a function of k, the number of function evaluations needed to converge, with tolerance 1% and 0.01%.

The `osEgo` algorithm is the most robust solver in S, converging to 4 digits of accuracy for all problems in less than 140 function evaluations. The `rbfSolve` and `arbfmip` algorithms are both very close to solve all problem instances as well, only the `ego` algorithm has significant failures.

Problem specific analysis

Table 4 and Table 5 present the results separately for each solver in S. The number of failures (in %) and the mean, min and max for the successful runs out of the total runs for each problem are reported. That is, the mean, min and max number of function evaluations needed to converge using the different experimental designs in E for each solver and problem.

The `osEGO` algorithm performs well, already confirmed by the data profiles. Some specific problems seem to cause a lot of trouble though. A common feature for problems B7 and B53 is the wide range of the objective. It seems like `osEGO`, just like the other solvers, is sensitive to such a large span in function values.

Another demanding problem is B16, the classical Shekel Foxholes. All solvers but `EGO` finds the global optimum for all experimental designs, in less than 200 function evaluations, which is impressive. Problem L49 is also challenging, filled with many local minima, even so both `rbfSolve` and `osEGO` are successful.

Table 4: Number of function evaluations to get within 1% of the optimal value.

Solver	%	**rbfSolve**			%	**arbfmip**			%	**ego**			%	**osEgo**		
	Fail	mean	min	max	*Fail*	mean	min	max	*Fail*	mean	min	max	*Fail*	mean	min	max
B4	*0*	43	18	58	*23*	51	38	60	*0*	41	31	53	*0*	38	17	49
B6	*0*	32	24	42	*0*	39	30	47	*0*	29	19	35	*0*	40	29	49
B7	*78*	169	168	170	*34*	166	145	200	*23*	69	26	194	*56*	162	151	172
B8	*0*	36	21	46	*0*	45	28	57	*23*	30	21	41	*0*	58	35	70
B16	*0*	99	38	130	*0*	140	37	186	*100*	-	-	-	*0*	113	55	148
B19	*0*	39	26	64	*0*	29	19	43	*12*	24	12	37	*0*	26	13	40
B30	*0*	20	10	32	*0*	26	23	30	*0*	24	13	33	*0*	22	13	30
B32	*0*	42	29	53	*0*	53	31	65	*56*	52	48	55	*0*	64	31	79
B53	*0*	79	22	133	*0*	119	68	199	*23*	40	25	56	*0*	95	32	193
L20	*0*	24	13	30	*0*	23	16	28	*0*	21	13	30	*0*	25	15	35
L25	*34*	36	24	68	*45*	32	30	38	*12*	61	9	181	*45*	18	10	26
L28	*34*	25	22	27	*0*	42	26	77	*0*	21	11	26	*0*	40	10	103
L49	*0*	86	61	103	*78*	71	68	74	*45*	112	89	141	*0*	106	21	180
C2	*0*	22	10	32	*0*	23	10	33	*0*	20	10	28	*0*	23	9	35
C3	*0*	22	12	35	*0*	25	15	36	*8*	21	11	28	*0*	24	14	31
C4	*0*	34	20	46	*0*	32	19	45	*0*	31	17	48	*0*	38	20	54
C13	*0*	28	11	45	*0*	28	11	45	*24*	31	13	45	*0*	33	15	47
C19	*8*	50	7	145	*0*	68	15	169	*24*	89	30	158	*0*	74	21	117
C22	*0*	42	32	67	*8*	59	34	149	*0*	26	17	36	*0*	29	19	53

Table 5: Number of function evaluations to get within 0.01% of the optimal value.

Solver	%	**rbfSolve**			%	**arbfmip**			%	**ego**			%	**osEgo**		
	Fail	mean	min	max	*Fail*	mean	min	max	*Fail*	mean	min	max	*Fail*	mean	min	max
B4	*0*	103	54	160	*23*	82	58	122	*0*	51	45	59	*0*	50	24	63
B6	*0*	44	30	61	0	*55*	48	62	*0*	41	29	55	*0*	62	50	79
B7	*89*	185	185	185	*56*	183	175	191	*56*	73	28	127	*89*	187	187	187
B8	*0*	53	35	71	*0*	57	38	73	*23*	34	25	48	*0*	89	76	110
B16	*0*	113	48	140	*0*	147	41	195	*100*	-	-	-	*0*	125	64	165
B19	*0*	54	40	70	*0*	35	24	54	*12*	27	15	39	*0*	33	24	47
B30	*0*	22	14	34	*0*	34	25	48	*0*	29	16	38	*0*	26	14	37
B32	*0*	64	50	81	*0*	60	39	77	*89*	52	52	52	*0*	87	38	137
B53	*12*	123	71	161	*23*	121	87	165	*67*	104	40	161	*45*	160	118	195
L20	*0*	34	23	45	*0*	36	30	42	*0*	25	16	35	*0*	39	27	50
L25	*34*	58	44	110	*45*	37	32	49	*34*	32	13	70	*45*	32	17	41
L28	*45*	64	39	91	*0*	136	79	166	*0*	26	16	35	*34*	92	25	144
L49	*0*	100	81	121	*78*	74	70	78	*45*	147	120	175	*0*	118	28	191
C2	*0*	27	18	36	*0*	26	15	37	*0*	22	12	30	*0*	27	14	39
C3	*0*	25	12	36	*0*	31	19	44	*8*	22	15	29	*0*	27	17	33
C4	*0*	44	20	61	*0*	37	23	47	*0*	37	21	60	*0*	50	29	82
C13	*0*	28	11	45	*0*	28	11	45	*24*	31	13	45	*0*	33	15	47
C19	*8*	56	11	146	*0*	72	16	174	*24*	96	33	160	*0*	78	25	129
C22	*0*	45	32	69	*8*	65	39	157	*0*	28	20	39	*0*	35	26	60

Experimental Designs and the one-stage EGO

Some interesting details were found when analyzing the performance of the `osEgo` algorithm with respect to the different experimental designs in E. For the set of unconstrained problems P_U, the CP+DGS experimental design works best with the option $N_2 = 10 \cdot d + 1$.

For the constrained problem set P_C, the Maximin LHD allowing infeasible points and using the option $N_1 = (d+1)(d+2)/2$ was the most successful design. The results also indicate, somewhat ambiguously, that one should rather use a constrained LHD when in combination with the Corner Point Strategy. It should be noted that the set P_C is quite small, hence it is difficult to draw any conclusions.

7 Conclusions

The one-stage EGO approach is promising, although it still needs to be improved. We have successfully implemented an adaptive scheme for f^*, similar to the one used in `ARBFMIP`. From the test results, though, it seems like a good idea to implement and test the cyclic choice of f^* as well.

By breaking down the full CML problem into subproblems, using univariate variables for θ, p and x^*, we are able to find good starting points and find new candidates x^* in each iteration. Using a clustering process, we find the best candidates from each cluster and proceed with multiple points each iteration.

Numerical issues are a big problem, and it might not be possible to resolve all of them. We have presented good fixes for some of the issues, allowing the `osEgo` implementation to compete with the other CGO solvers in TOMLAB, outperforming the old `ego` implementation.

The nasty subproblem CML increase with the number of sampled points n, since the size of correlation matrices $\mathbf{R}$ and $\mathbf{C}$ are $n \times n$. This causes the calculations to get heavier as the iterations go by, and more parameter combinations become infeasible, complicating the optimization of the subproblems.

At the moment, the solver used to maximize the full CML subproblem utilize a set of starting points $\mathbf{x_0}$. These points are chosen as the sampled points $\mathbf{x}$, but slightly perturbed towards the midpoint of the sample space. This is motivated by inspection of the subproblem, realizing that the optimal solution is often very close to an already sampled point.

Due to our discussion in Section 4.2, we choose to perturb the starting points, avoiding numerical issues and not starting in a deep basin surrounding sampled points. By perturbing towards the midpoint, we keep feasibility with respect to the box-bounds.

7.1 Future work

We need to speed up the subproblem solving phase. As n increases, so does the number of starting points $\mathbf{x_0}$. As points tend to pile up in promising areas, many of

the points in $\mathbf{x_0}$ are very similar (close in Euclidean meaning), a bad feature for a set of starting points. A possible remedy, partly implemented already, is to cluster the sampled points $\mathbf{x}$ and thus reduce the size of $\mathbf{x_0}$ and hence decrease the solution times.

The numerical results indicate that `osEGO` is sensitive to large spans in function values. We should consider strategies of transforming the function values, perhaps consider the log values, in order to reduce the range.

References

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

[2] E. D. Dolan, J. J. Moré, and T. S. Munson: Optimality Measures for Performance Profiles. *Preprint ANL/MCS-P1155-0504* (2004).

[3] H.-M. Gutmann: A radial basis function method for global optimization. *Journal of Global Optimization* **19** (3), 201–227 (2001).

[4] K. Holmström: An adaptive radial basis algorithm (ARBF) for expensive black-box global optimization. *Journal of Global Optimization* **41**, 447–464 (2008).

[5] K. Holmström, N.-H. Quttineh, and M. M. Edvall: An adaptive radial basis algorithm (ARBF) for expensive black-box mixed-integer constrained global optimization. *Optimization and Engineering* **41**, 447–464 (2008).

[6] D. R. Jones: A Taxonomy of Global Optimization Methods Based on Response Surfaces. *Journal of Global Optimization* **21**, 345–383 (2002).

[7] D. R. Jones, M. Schonlau, and W. J. Welch: Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization* **13**, 455–492 (1998).

[8] J. J. Moré and S. M. Wild: Benchmarking Derivative-Free Optimization Algorithms. *Preprint ANL/MCS-P1471-1207* (2007).

[9] N.-H. Quttineh and K. Holmström: The influence of Experimental Designs on the Performance of Surrogate Model Based Costly Global Optimization Solvers. *Studies in Informatics and Control* **18** (1), (2009).

[10] J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn: Design and analysis of computer experiments (with discussion). *Statistical Science*, **4**, 409–435 (1989).

[11] M. J. Sasena: Flexibility and Efficiency Enhancements for Constrained Global Design Optimization with Kriging Approximations. *Doctoral Dissertation* (2002).