Mathematical Programming Computation (2018) 10:597–629
https://doi.org/10.1007/s12532-018-0144-7

FULL LENGTH PAPER

RBFOpt: an open-source library for black-box optimization with costly function evaluations

Alberto Costa[1] · Giacomo Nannicini[2]

Received: 5 November 2015 / Accepted: 13 August 2018 / Published online: 27 August 2018

Abstract
We consider the problem of optimizing an unknown function given as an oracle over a mixed-integer box-constrained set. We assume that the oracle is expensive to evaluate, so that estimating partial derivatives by finite differences is impractical. In the literature, this is typically called a *black-box* optimization problem with costly evaluation. This paper describes the solution methodology implemented in the open-source library RBFOpt, available on COIN-OR. The algorithm is based on the Radial Basis Function method originally proposed by Gutmann (J Glob Optim 19:201–227, 2001. https://doi.org/10.1023/A:1011255519438), which builds and iteratively refines a surrogate model of the unknown objective function. The two main methodological contributions of this paper are an approach to exploit a noisy but less expensive oracle to accelerate convergence to the optimum of the exact oracle, and the introduction of an automatic model selection phase during the optimization process. Numerical experiments show that RBFOpt is highly competitive on a test set of continuous and mixed-integer nonlinear unconstrained problems taken from the literature: it outperforms the open-source solvers included in our comparison by a large amount, and performs slightly better than a commercial solver. Our empirical evaluation provides insight on which parameterizations of the algorithm are the most effective in practice. The software reviewed as part of this submission was given the Digital Object Identifier (DOI) https://doi.org/10.5281/zenodo.597767.

Keywords Black-box optimization · Derivative-free optimization · Global optimization · Radial basis function · Open-source software · Mixed-integer nonlinear programming

Mathematics Subject Classification 90C56 · 90C30 · 65K05 · 97N80

✉ Giacomo Nannicini
nannicini@us.ibm.com

Alberto Costa
costa@lix.polytechnique.fr

1 Future Cities Laboratory Program, ETH Zurich, Singapore, Singapore

2 IBM T.J. Watson Research Center, Yorktown Heights, NY, USA

1 Introduction

In this paper, we address a problem cast in the following form:

$$\left.\begin{array}{l} \min\ f(x) \\ \qquad x \in [x^L, x^U] \\ \qquad x \in \mathbb{Z}^q \times \mathbb{R}^{n-q}, \end{array}\right\} \qquad (1)$$

where $f : \mathbb{Z}^q \times \mathbb{R}^{n-q} \rightarrow \mathbb{R}$, $x^L, x^U \in \mathbb{R}^n$ are vectors of (finite) lower and upper bounds on the decision variables, and $q \leq n$. We assume that the analytical expression for f is unknown and function values are only available through an oracle that is expensive to evaluate, e.g., a time-consuming computer simulation. In the literature, this is typically called a *black-box* optimization problem with costly evaluation.

This problem class finds many applications. Our work originated from a project in architectural design where we faced the following problem (see also [12,53]). During the design phase, a building can be described by a parametric model and the parameters are the decision variables, which can be continuous or discrete. Lighting and heating simulation software can be used to study energy profiles of buildings, simulating sun exposure over a prescribed period of time. This information can be used to determine a performance measure, i.e., an objective function, but the analytical expression is not available due to the complexity of the simulations. The goal is to optimize this function to find a good parameterization of the parametric model of the building. However, each run of the simulation software usually takes considerable time: up to several hours. Thus, we want to optimize the objective function within a small budget of function evaluations to keep computing times under control. Other applications of this approach can be found in engineering disciplines where the simulation relies on the solution of a system of PDEs, for example in the context of performance optimization for complex physical devices such as engines, see e.g., [4,25].

There is a very large stream of literature on black-box optimization in general, also called *derivative-free optimization* (sometimes generating confusion). Numerous methods have been proposed, and the choice of a particular method should depend on the number of function evaluations allowed, the dimension of the problem, and its structural properties. Heuristic approaches are very common thanks to their simplicity, for example scatter search, simulated annealing and evolutionary algorithms; see [19,20] for an overview. However, these methods are not specifically tailored for the setting of this paper and often require a large number of function evaluations, as has been noted by [22,47] among others. In general, methods that do not take advantage of the inherent smoothness of the objective function may take a long time to converge [9]. Unfortunately, because of the assumption of expensive function evaluations, estimating partial derivatives by finite differences may be impractical and often has prohibitive computational cost. A commonly used approach in this context is that of building an approximate model of f, also called *response surface* or *surrogate model*. Examples of this approach are the Gutmann's Radial Basis Function (RBF) method [22] (see also [46]), the stochastic RBF method [47], and the kriging-based Efficient Global Optimization method (EGO) [33]. The surrogate model constructed by these methods is a global model that uses all available information on f, as opposed to methods that only build a local model such as trust region methods [9,10]. Other methods for black-box optimization rely on direct search, i.e., they do not build a surrogate model of the objective function or of its gradient. An overview of direct search methods can be found in [34], and a comprehensive treatment is given in [10]. We refer the reader to [49] and the references therein for a recent survey on black-box methods with an extensive computational evaluation.

In this paper we focus on problems that are potentially multimodal, relatively small-dimensional, and for which only a small number of function evaluations is allowed. For this type of problems, algorithms based on a surrogate model showed good performance in practice. Empirical evidence reported in [27] suggests that Gutmann's RBF method is one of the most effective on engineering problems. Recent work showcases the effectiveness of global optimization approaches that rely on RBF surrogate models in a variety of contexts, see e.g. [29,39,53].

In this paper, we review Gutmann's RBF method and present some extensions aimed at improving its practical performance. Our contributions are a fast procedure for automatic model selection, and an approach to accelerate convergence in case we have access to an additional oracle that returns function values affected by error but is less expensive to evaluate than the exact oracle for f. Although we only test these ideas in the context of Gutmann's RBF method, these contributions can be adapted to other surrogate model methods that use RBF interpolants, and should therefore be considered of general interest. Another contribution of this paper is the associated implementation in an open-source library called RBFOpt. We show that RBFOpt is competitive with state-of-the-art commercial software, and substantially more effective than several black-box optimization methods taken from the literature.

The rest of the paper is organized as follows. In Sect. 2 we review Gutmann's RBF method. Sections 3 and 4 present our modifications of the algorithm (automatic model selection and exploitation of a noisy oracle) to improve the efficacy of the method. Section 5 describes our implementation. Section 6 reports the results of a computational evaluation on a set of test problems taken from the literature. Section 7 concludes the paper.

2 The radial basis function algorithm for black-box optimization

Before we discuss global optimization algorithms based on RBFs, we must define the surrogate model. Let $\Omega := [x^L, x^U] \subset \mathbb{R}^n$, $\Omega_I := \Omega \cap (\mathbb{Z}^q \times \mathbb{R}^{n-q})$, and we assume that the box constraints on the first q variables have integer endpoints. Given k distinct points $x_1, \dots, x_k \in \Omega$, the radial basis function interpolant s_k is defined as:

$$s_k(x) := \sum_{i=1}^{k} \lambda_i \phi(\|x - x_i\|) + p(x), \tag{2}$$

where $\phi : \mathbb{R}_+ \to \mathbb{R}$, $\lambda_1, \dots, \lambda_k \in \mathbb{R}$ and p is a polynomial of degree d. The minimum degree $d_{\min}$ to guarantee existence of the interpolant depends on the form of the

Table 1 Common RBF functions

$\phi(r)$		$d_{\min}$
r	(linear)	0
r^3	(cubic)	1
$\sqrt{r^2+\gamma^2}$	(multiquadric)	0
$r^2 \log r$	(thin plate spline)	1

functions ϕ. Typically, the polynomial is picked to be of degree exactly $d_{\min}$ in practical implementations, and we follow this custom. Table 1 gives the most commonly used radial basis functions $\phi(r)$ and the corresponding value of $d_{\min}$. The parameter $\gamma > 0$ can be used to change the shape of these functions, but it is usually set to 1.

If $\phi(r)$ is cubic or thin plate spline, $d_{\min} = 1$ and with $d = d_{\min}$ we obtain an interpolant of the form:

$$s_k(x) := \sum_{i=1}^{k} \lambda_i \phi(\|x - x_i\|) + h^T \begin{pmatrix} x \\ 1 \end{pmatrix}, \tag{3}$$

where $h \in \mathbb{R}^{n+1}$. The values of λ_i, h can be determined by solving the following linear system:

$$\begin{pmatrix} \Phi & P \\ P^T & 0_{(n+1)\times(n+1)} \end{pmatrix} \begin{pmatrix} \lambda \\ h \end{pmatrix} = \begin{pmatrix} F \\ 0_{n+1} \end{pmatrix}, \tag{4}$$

with:

$$\Phi = \left(\phi(\|x_i - x_j\|)\right)_{i,j=1,\dots,k}, \quad P = \begin{pmatrix} x_1^T & 1 \\ \vdots & \vdots \\ x_k^T & 1 \end{pmatrix}, \quad \lambda = \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_k \end{pmatrix}, \quad F = \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_k) \end{pmatrix}.$$

The algorithm presented later in this section ensures that the points $x_1, \dots, x_k$ are pairwise distinct and $\text{rank}(P) = n + 1$, guaranteeing that the system (4) is nonsingular [22]. We denote by A_k the matrix of (4) with points $x_1, \dots, x_k$.

If $\phi(r)$ is linear or multiquadric, $d_{\min} = 0$ so that for $d = d_{\min}$, P is the all-one column vector of dimension k. The dimensions of the zero matrix and vector in (4) are adjusted accordingly.

The above discussion shows that given a set of points, a RBF interpolant can be computed by solving a linear system. The introduction of RBFs as a tool for the global optimization of expensive black-box functions is due to [22], relying on the work of [44]. Since then, several global optimization methods that follow the general scheme given below have been proposed, see e.g. [38,41,47]:

- *Initial step* Choose affinely independent points $x_1, \dots, x_{n+1} \in \Omega_I$ using an initialization strategy. Set $k \leftarrow n + 1$.
- *Iteration step* Repeat the following steps.

  (i) Compute the RBF interpolant s_k to the points $x_1, \dots, x_k$, solving (4).

(ii) Choose a trade-off between *exploration* and *exploitation*.
(iii) Determine the next point x_{k+1} based on the choice at step (ii).
(iv) Evaluate f at x_{k+1}.
(v) If we exceed a prescribed number of function evaluations, stop. Otherwise, set $k \leftarrow k + 1$.

At Iteration step (ii), *exploration* implies trying to improve the surrogate model in unknown parts of the domain, whereas *exploitation* implies trying to find the best objective function value based on the current surrogate model. To turn the general scheme above into a fully specified algorithm, we must describe how to choose points in the Initial step, and an implementation of Iteration steps (ii) and (iii).

2.1 Choice of the initial sample points

One of the simplest methods to select the initial sample points is to pick the 2^n corner points of the box Ω_I, but this is reasonable only for small values of n. A commonly used strategy [22,26,27] is to choose $n + 1$ corner points of the box Ω_I and the central point of Ω_I, but this could prioritize the exploration in a part of the domain. Holmström et al. [27] chooses x^L and $x^L + (x_i^U - x_i^L)e^i$ for $i = 1, \dots, n$ as initial corner points, where e^i is the ith vector of the standard orthonormal basis. Another popular strategy is to use a Latin Hypercube experimental design, typically chosen among some randomly generated Latin Hypercube designs according to a maximum minimum distance or a minimum maximum correlation criterion. Points sampled this way may not be feasible for Ω_I because the integrality constrained variables may not be integer valued. Rounding recovers feasibility for Ω_I, but it may destroy the properties of the Latin Hypercube; even worse, some of the rounded points may coincide, thereby preventing the computation of the RBF interpolant. However, in our experience simple rounding works well in practice, and in case the points in the resulting set are not pariwise distinct, we generate a new random sample, which is computationally inexpensive. This is our default strategy in the computational experiments, and it is also the one followed by [38,41].

A few papers in the literature discuss how to choose the initial points when explicit constraints are given in the problem formulation. This is not necessary in the setting of this paper, but we refer the interested reader to [13,17,27].

2.2 Exploration versus exploitation: a measure of bumpiness

We now discuss the trade-off between exploration and exploitation from the point of view of Gutmann's RBF method [22], using a measure of "bumpiness" of the surrogate model. The implicit assumption is that the unknown function f does not oscillate too much, therefore we aim for a model that interpolates the data and minimizes the bumpiness. We now provide details on the measure of bumpiness σ used in [22]. This will give us the foundations to describe how the trade-off between exploration and exploitation is determined.

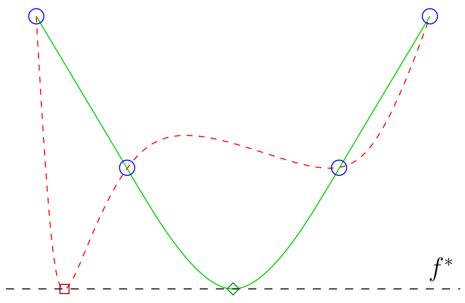


Fig. 1 An example with four interpolation points (circles) and a value of f_k^* represented by the horizontal dashed line. Gutmann's RBF method assumes that it is more likely that a point with value f_k^* is located at the diamond rather than the square, because the resulting interpolant is less bumpy

The motivation to use bumpiness comes from the theory of natural cubic spline interpolation in dimension one (RBFs can be seen as its extension to multivariate functions). It is well known that the natural cubic spline interpolant with $n = 1$ whose parameters are found by solving system (4) is the function that minimizes $\int_{\mathbb{R}} [g''(x)]^2 \, dx$ among all the functions $g : \mathbb{R} \to \mathbb{R}$ such that $\forall i \in \{1, \dots, k\}$ $g(x_i) = f(x_i)$. Hence, $\int_{\mathbb{R}} [g''(x)]^2 dx$ is a possible measure of bumpiness for functions over $\mathbb{R}$. It is shown in [22] that in the case of RBF interpolants in dimension n, such a measure of bumpiness can be written as:

$$\sigma(s_k) = (-1)^{d_{\min}+1} \sum_{i=1}^{k} \lambda_i s_k(x_i) = (-1)^{d_{\min}+1} \sum_{i=1}^{k} \sum_{j=1}^{k} \lambda_i \lambda_j \phi(\|x_i - x_j\|) =$$
$$= (-1)^{d_{\min}+1} \lambda^T \Phi \lambda.$$

Let us assume that after k function values $f(x_1), \dots, f(x_k)$ are evaluated, we want to find a point in Ω_I where it is likely that the unknown function attains a target value $f_k^* \in \mathbb{R} \cup \{-\infty\}$ (strategies to select f_k^* will be discussed in Sect. 2.3). Let s_y be the RBF interpolant subject to the conditions:

$$s_y(x_i) = f(x_i) \qquad \forall i \in \{1, \dots, k\} \tag{5}$$
$$s_y(y) = f_k^*. \tag{6}$$

The assumption of Gutmann's RBF method is that a likely location for the point y with function value f_k^* is the one that minimizes $\sigma(s_y)$. That is, we look for the interpolant that is "the least bumpy" and interpolates at (y, f_k^*) as well as at previously evaluated points. A sketch of this idea is given in Fig. 1.

Instead of computing the minimum of $\sigma(s_y)$ as defined above to find the least bumpy interpolant, we define an equivalent optimization problem that is easier to solve. Let ℓ_k be the RBF interpolant to the points $(x_i, 0)$, $\forall i \in \{1, \dots, k\}$ and $(y, 1)$. A solution to (5)–(6) can be rewritten as:

$$s_y(x) = s_k(x) + [f_k^* - s_k(y)]\ell_k(x), \; x \in \mathbb{R}^n,$$

which clearly interpolates at the desired points by definition of ℓ_k. Let $\mu_k(y)$ be the coefficient corresponding to y of ℓ_k. $\mu_k(y)$ can be computed by extending the linear system (4), which becomes (see [22]):

$$\begin{pmatrix} \Phi & u(y) & P \\ u(y)^T & \phi(0) & \pi(y)^T \\ P^T & \pi(y) & 0 \end{pmatrix} \begin{pmatrix} \alpha(y) \\ \mu_k(y) \\ b(y) \end{pmatrix} = \begin{pmatrix} 0_k \\ 1 \\ 0_{n+1} \end{pmatrix}, \tag{7}$$

where $u(y) = (\phi(\|y - x_1\|), \dots, \phi(\|y - x_k\|))^T$ and $\pi(y)$ is $\begin{pmatrix} y^T & 1 \end{pmatrix}$ when $d = 1$, and it is 1 when $d = 0$. With algebraic manipulations (see [22,46]) we can obtain from the system (7) the following expression for $\mu_k(y)$:

$$\mu_k(y) = \frac{1}{\phi(0) - \begin{pmatrix} u(y)^T & \pi(y)^T \end{pmatrix} A_k^{-1} \begin{pmatrix} u(y) \\ \pi(y) \end{pmatrix}}. \tag{8}$$

A way of storing the factorization of A_k to speed up the computation of $\mu_k(y)$ is described in [5].

It can be shown [22] that computing the minimum of $\sigma(s_y)$ over $y \in \mathbb{R}^n$ is equivalent to minimizing the utility function:

$$g_k(y) = (-1)^{d_{\min}+1} \mu_k(y)[s_k(y) - f_k^*]^2, \qquad y \in \Omega \setminus \{x_1, \dots, x_k\}.$$

Unfortunately g_k and μ_k are not defined at $x_1, \dots, x_k$, and $\lim_{x \to x_i} \mu_k(x) = \infty$, $\forall i \in \{1, \dots, k\}$. To avoid numerical troubles, [22] suggests maximizing the following function:

$$h_k(x) = \begin{cases} \frac{1}{g_k(x)} & \text{if } x \notin \{x_1, \dots, x_k\} \\ 0 & \text{otherwise,} \end{cases} \tag{9}$$

which is differentiable everywhere on Ω.

We now have all the necessary ingredients to describe Iteration steps (ii) and (iii) as in Gutmann's RBF algorithm.

- *Iteration step* (for Gutmann's RBF algorithm):
  (ii) Choose a target value $f_k^* \in \mathbb{R} \cup \{-\infty\} : f_k^* \leq \min_{x \in \Omega_I} s_k(x)$.
  (iii) Compute
  $$x_{k+1} = \arg\max_{x \in \Omega_I} h_k(x), \tag{10}$$
  where $h(x)$ is defined as in (9); depending on the choice of f_k^*, we may solve a variation of this problem instead, as discussed in Sect. 2.3.

We called this algorithm "Gutmann's RBF algorithm". We remark that $q = 0$ in the framework of [22], i.e., there are no integer variables. Extensions to $q > 0$, similar to the approach adopted in our paper, are given in [27,38,41].

A number of papers in the recent literature on surrogate model algorithms advocate random sampling to select the next evaluation point (e.g., [41,47]) as opposed to optimization of a performance criterion, which is the strategy followed by [22,33] among others. The two approaches are compared in [40], concluding that there appears to be no clear winner. Both have advantages: sampling is faster, but optimization typically allows the incorporation of convex constraints at little extra cost.

RBFOpt v1.0 only allowed a mathematical programming based optimization approach, and in this paper we limit ourselves to this case. Subsequent versions of the library introduced additional strategies to select the next evaluation point, e.g., different evaluation criteria and a trust region method for local search, but the discussion of these new strategies is beyond the scope of this paper. Some of them are described in [11,14]. We simply note that in our tests, the conclusions of the computational study presented here are also valid for subsequent versions of RBFOpt, all of which adopt the general global optimization scheme using RBFs given in Sect. 2.

We still need to specify strategies to choose the target value f_k^* at Iteration step (ii). This is the subject of the next section. Afterward, we discuss a number of modifications to the basic algorithms that have been proposed in the literature and were typically found to be beneficial in practice.

2.3 Selection of the target value f_k^*

We use the technique proposed in [27] to select the target value f_k^* at each Iteration step (ii). It works as follows. Let $y^* := \arg\min_{x \in \Omega_I} s_k(x)$, $f_{\min} := \min_{i=1,\dots,k} f(x_i)$, and $f_{\max} := \max_{i=1,\dots,k} f(x_i)$. We employ a cyclic strategy that picks target values $f_k^* \in \mathbb{R} \cup \{-\infty\}$ according to the following sequence of length $\kappa + 2$:

- Step -1 (*InfStep*): Choose $f_k^* = -\infty$. In this case the problem of finding x_{k+1} can be rewritten as:

$$x_{k+1} = \arg\max_{x \in \Omega_I} \frac{1}{(-1)^{d_{\min}+1} \mu_k(x)}.$$

  This is a pure exploration phase, yielding a point far from $x_1, \dots, x_k$.
- Step $h \in \{0, \dots, \kappa - 1\}$ (*Global search*): Choose

$$f_k^* = s_k(y^*) - (1 - h/\kappa)^2 (f_{\max} - s_k(y^*)). \tag{11}$$

  In this case, we try to strike a balance between improving model quality and finding the minimum.
- Step κ (*Local search*): Choose $f_k^* = s_k(y^*)$. Notice that in this case (9) is maximized at y^*. Hence, if $s_k(y^*) < f_{\min} - 10^{-10}|f_{\min}|$ we accept y^* as the new sample point x_{k+1} without solving (10). Otherwise we choose $f_k^* = f_{\min} - 10^{-2}|f_{\min}|$. This is an exploitation phase, trying to find the best objective function value based on current information.

The choice of the target values is important for the convergence of the method. Since all variables are bounded, the integer variables take values from a finite set. In order to show ϵ-convergence to a global optimum for any continuous function, it is necessary and sufficient [51] that for every value of the integer variables in $\mathbb{Z}^q \cap \Omega_I$, the sequence of points (x_k) generated by the algorithm is dense in the projection of

Ω_I over $\mathbb{R}^{n-q}$, i.e., the space of the continuous variables. Gutmann [22] considers the case where $q = 0$ and proves that if f^* is "small enough" throughout the optimization algorithm, the sequence of points $\{x_k\}_{k=1,\dots,\infty}$ generated by Gutmann's RBF algorithm is dense over Ω when ϕ is linear, cubic or thin plate spline. A consequence of this fact, remarked in [22], is that if *Infstep* is performed for infinitely many $k \in \mathbb{N}$ then the algorithm converges to a global optimum as $k \to \infty$.

Despite what the theoretical analysis suggests, the computational evaluations of Gutmann's RBF method in the literature typically skip *InfStep* ($f_k^* = -\infty$). This can be explained by the fact that *InfStep* completely disregards the objective function, hence it rarely helps speeding up convergence in practice.

2.4 Improvements over the basic algorithm

Several modifications of Gutmann's RBF algorithm have been proposed in the literature, with the aim of improving its practical performance. We now describe the most impactful modifications tested in the computational experiments of Sect. 6, which are active by default in our implementation.

(a) In the *Global search* step, [22,46] replace $f_{\max}$ in Eq. (11) with a dynamically chosen value $f(x_{\pi(\alpha(k))})$, defined as follows. Let k_0 be the number of initial sampling points, h the index of the current *Global search* iteration as in Sect. 2.3, π a permutation of $\{1, \dots, k\}$ such that $f(x_{\pi(1)}) \le f(x_{\pi(2)}) \le \dots \le f(x_{\pi(k)})$, and

$$\alpha(k) = \begin{cases} k & \text{if } h = 0 \\ \alpha(k-1) - \left\lfloor \frac{k-k_0}{\kappa} \right\rfloor & \text{otherwise.} \end{cases}$$

As a result, $f_{\max}$ is used to define the target value f_k^* only at the first step ($h = 0$) of each *Global search* cycle. In subsequent steps, we pick progressively lower values of $f(x_i)$, with the goal of stabilizing search by avoiding too large differences between the minimum of the RBF interpolant and the target value.

(b) If the initial sample points are chosen with a random strategy (for example, a Latin Hypercube design), whenever we detect that the algorithm is stalling, we apply a complete restart strategy [46, Sect. 5]. Restart strategies have been applied to numerous combinatorial optimization problems, such as satisfiability [21] and integer programming [1]. In the context of Gutmann's RBF algorithm, the strategy introduced in [46] consists in restarting the algorithm from scratch (including the generation of new initial sample points) whenever the best known solution does not improve by at least a specified value (0.1% by default) after a given number of optimization cycles (5 by default). In our experience restarts are more useful if the initial sample points are chosen according to a randomized strategy, because otherwise, the solution run after the restart may be similar to, or even the same as, before the restart. Hence, we only apply restarts if the initial sampling strategy is randomized.

(c) A known issue of Gutmann's RBF method, explicitly pointed out in [46, Sect. 4], is that large values of h in *Global search* do not necessarily imply that the algorithm is performing a "relatively local" search as intended. In fact, the next iterate can be arbitrarily far from the currently known best point, and this can severely hamper convergence on problems where the global minimum is in a steep valley. To alleviate this issue, [46, Sect. 4.3] proposes a "restricted global minimization of the bumpiness function". The idea is to progressively restrict the search box around the best known solution during a *Global search* cycle. In particular, instead of solving (10) over Ω_I, we intersect Ω_I with the box $[\min_{y \in \Omega_I} s_k(y) - \beta_k(x^U - x^L), \min_{y \in \Omega_I} s_k(y) + \beta_k(x^U - x^L)]$, where $\beta_k = 0.5(1 - h/\kappa)$ if $(1 - h/\kappa) \leq 0.5$, and $\beta_k = 1$ otherwise (the numerical constants indicated are the values suggested by [46]). It is easy to verify that this restricts the global search to a box centered on the global minimizer of the RBF interpolant: the box coincides with Ω_I at the beginning of every *Global search* cycle, but gets smaller as h increases. This turns out to be very beneficial on problems with steep global minima.

3 Automatic model selection

One of the drawbacks of RBF methods is that there is no built-in mechanism to assess model quality: there are many possible surrogate models depending on the choice of the RBF function (see Table 1), and it is difficult to predict a priori which one has the best performance on a specific problem.

We propose an assessment of model quality using a cross validation scheme, in order to dynamically choose the surrogate model that appears to be the most accurate for the problem at hand. Cross validation is a commonly used model validation technique in statistics. Given a data set, cross validation consists in using part of the data set to fit a model, and testing its quality on the remaining part of the data set. The process is then iterated, rotating the parts of the data set used for model fitting and for testing.

Let s_k be the surrogate model for f based on k evaluation points $x_1, \ldots, x_k$. We assume that the points are sorted by increasing function value: $f(x_1) \leq f(x_2) \leq \cdots \leq f(x_k)$; this is without loss of generality as we can always rearrange the points. We perform cross validation as follows. For $j \in \{1, \ldots, k\}$, we can fit a surrogate model $\tilde{s}_{k,j}$ to the points $(x_i, f(x_i))$ for $i = 1, \ldots, k, i \neq j$ and evaluate the performance of $\tilde{s}_{k,j}$ at $(x_j, f(x_j))$. We use an order-based measure to evaluate performance of the surrogate model. For a given scalar y, let $\text{order}_{k,j}(y)$ be the position at which y should be inserted in the ordered list $f(x_1) \leq \cdots \leq f(x_{j-1}) \leq f(x_{j+1}) \leq \cdots \leq f(x_k)$ to keep it sorted. Since $\text{order}_{k,j}(f(x_j)) = j$, we use the value $q_{k,j} = |\text{order}_{k,j}(\tilde{s}_{k,j}(x_j)) - j|$ to assess the predictive power of the model. We then average $q_{k,j}$ with j ranging over some subset of $\{1, \ldots, k\}$ to compute a model quality score. This approach is a variation of leave-one-out cross validation in which we look at how the surrogate model ranks the left-out point compared to the other points, rather than evaluate the accuracy of the prediction in absolute terms. This is motivated by the observation that for the purpose of optimization, a surrogate model that ranks all points correctly is arguably more useful than a surrogate model that attains small absolute errors, but is not able to predict how points compare to each other. The idea of order-based performance metrics to evaluate surrogate models is explored in [3].

We perform model selection at the beginning of every cycle of the search strategy to select f_k^*, see Sect. 2.3. Our aim is to select the RBF model with the best predictive power. We choose two different models: one for local search, one for global search, corresponding to different Iteration steps of the algorithm. We do this by computing the average value $\bar{q}_{10\%}$ of $q_{k,j}$ for $j = 1, \dots, \lfloor 0.1k \rfloor$, and the average value $\bar{q}_{70\%}$ of $q_{k,j}$ for $j = 1, \dots, \lfloor 0.7k \rfloor$, for a subset of the basis functions of Table 1. The rationale of our approach is that $\bar{q}_{10\%}$ is an estimate of how good a particular surrogate model is at ranking function values for the points that have a low function value, which are arguably the most important for local search. On the other hand, for global search it seems reasonable to choose a model that has good predictive performance on a larger range of function values, hence we use $\bar{q}_{70\%}$. The points with the highest function values are the farthest from the minimum and our assumption is that they can be disregarded.

The RBF model with the lowest value of $\bar{q}_{10\%}$ is employed in the subsequent optimization cycle for the *Local search* step and the *Global search* step with $h = \kappa - 1$, while the RBF model with lowest value of $\bar{q}_{70\%}$ is employed for all the remaining steps. The RBF models that we consider in our experiments are those with linear, cubic, multiquadric (with $\gamma = 1$) or thin plate spline basis functions. It is possible to consider additional models, such as those obtained using different scaling techniques (see Sect. 5.2.1), but we did not pursue this possibility because we prefer to choose among a small number of diverse models.

A drawback of leave-one-out cross validation is that it is typically expensive to perform. We show that the values $\bar{q}_{10\%}, \bar{q}_{70\%}$ can be computed in time $O(m^3)$, where m is the number of rows of (4) (i.e., $m = k + n + 1$ for cubic and thin plate spline RBF, $m = k + 1$ for linear and multiquadric). To carry out the cross validation scheme we must compute several RBF interpolants, each one of which interpolates at the points $x_1, \dots, x_k$ except a single point x_j. Instead of repeatedly solving (4), we notice that the linear system with the interpolation conditions for all points except x_j is obtained from (4) by deleting the jth row and column. By construction, the reduced linear system is also nonsingular. Thus, we use an iterative procedure to compute the solution of each of the (at most) k linear systems with one row and one column deleted, starting from the solution of the full system (4) and the LU factorization of its matrix A_k. For convenience, denote the system (4) as $A_k x = b$, where $A_k \in \mathbb{R}^{m \times m}$ and all its first $k \leq m$ principal minors are nonzero. Our goal is to solve a sequence of systems $A^j x^j = b^j$, where A^j is obtained from A_k deleting the jth row and column, b^j is obtained from b deleting the jth row, x^j is $(m-1)$-dimensional, and $j \leq k$.

We first compute the decomposition $A_k = LU$, and using this factorization, find $\bar{x} = A_k^{-1} b$. The vector $\bar{x}$ is the base solution. For a given j, there are two cases to consider.

1. Case $\bar{x}_j = 0$. Then the vector $\tilde{x}^j = (\bar{x}_i)_{i \neq j}$, obtained by dropping the jth component of $\bar{x}$, is such that $A^j \tilde{x}^j = b^j$ by definition of $\bar{x}$, and is the desired solution.
2. Case $\bar{x}_j \neq 0$. Let e^j be the jth basis vector of dimension m, i.e., $e^j_j = 1$ and all its other components are zero. Let $\bar{y}^j$ be such that $A_k \bar{y}^j = e^j$; notice that this is the jth column of A_k^{-1}, and we can find it with one LU solve starting from the existing factorization of A_k. Since A^j is nonsingular we must have $\bar{y}^j_j \neq 0$ (otherwise, the

nonzero vector $\tilde{y}^j$ obtained by dropping the jth component of $\bar{y}^j$ would satisfy $A^j \tilde{y}^j = 0$, a contradiction). Define $\bar{w}^j = \bar{x} - \frac{\bar{x}_j}{\bar{y}^j_j}\bar{y}^j$. Then we have:

$$A_k \bar{w}^j = A_k \left(\bar{x} - \frac{\bar{x}_j}{\bar{y}^j_j}\bar{y}^j \right) = A_k \bar{x} - \frac{\bar{x}_j}{\bar{y}^j_j} A_k \bar{y}^j = b - \frac{\bar{x}_j}{\bar{y}^j_j} e^j.$$

Since $e^j_i = 0$ for $i \neq j$, $(b - \frac{\bar{x}_j}{\bar{y}^j_j} e^j)_i = b_i$ for all $i \neq j$. Furthermore, $\bar{w}^j_j = \bar{x}_j - \frac{\bar{x}_j}{\bar{y}^j_j}\bar{y}^j_j = 0$. Then we can simply drop the jth component of $\bar{w}^j$ to obtain a solution for $A^j x^j = b^j$. Notice that $\bar{w}^j$ is obtained with a single LU-solve plus m additions and m multiplications.

The two cases above show that to solve $A^j x^j = b^j$, we can reuse the LU factorization of A_k and perform a single LU solve for each j. The initial LU factorization of A_k takes time $O(m^3)$, and we perform $O(k)$ LU solves requiring time $O(m^2)$ each, yielding total execution time $O(m^3)$.

4 Taking advantage of a faster but less accurate oracle

In practical applications it is common to have a trade-off between computing time and accuracy of the black-box function f: for example, if f is evaluated through a computer simulation, the simulator can often be parameterized to achieve different levels of precision. In particular, accuracy of simulations is typically not linear in computing time, and in practice one can get relatively good estimations of the true value of $f(x)$ in a fraction of the time.

To accelerate the optimization process, we would like to exploit low-accuracy but faster simulations. We assume that, in addition to the oracle f, we have access to an oracle $\tilde{f} := (\tilde{f}^x, \tilde{f}^\ell, \tilde{f}^u)$ with the properties that $\tilde{f}^\ell(x) \leq \tilde{f}^u(x)$ and $f(x), \tilde{f}^x(x) \in [\tilde{f}^\ell(x), \tilde{f}^u(x)]$. The oracle $\tilde{f}^x(x)$ represents an approximation of the function value $f(x)$, and $\tilde{f}^\ell(x)$, $\tilde{f}^u(x)$ provide (possibly loose) lower and upper bounds on the true function value $f(x)$. We assume that evaluating $\tilde{f}$ is less expensive than evaluating f, i.e., $\tilde{f}$ can be computed significantly faster; the three values $(\tilde{f}^x(x), \tilde{f}^\ell(x), \tilde{f}^u(x))$ are returned at the same time when $\tilde{f}$ is evaluated.

Our approach consists in: a first phase (the "fast phase") where we aim to solve (1) using both f and $\tilde{f}$, and a second phase (the "accurate phase"), where (1) is reoptimized performing additional function evaluations using f. We describe each phase below.

In the first phase, oracle evaluations are performed using $\tilde{f}$ unless otherwise specified (exceptions are given below). Let N be the set of indices of the points evaluated via $\tilde{f}$, for which we have stored the triples $(\tilde{f}^x(x_i), \tilde{f}^\ell(x_i), \tilde{f}^u(x_i))$, $i \in N$; we define the vectors $f^L, f^U \in \mathbb{R}^k$ with entries $f^L_i = \tilde{f}^\ell(x_i)$, $f^U_i = \tilde{f}^u(x_i)$ for $i \in N$, and $f^L_i = f^U_i = f(x_i)$ for $i \notin N$. Because function values $\tilde{f}^x(x_i)$, $i \in N$ are not exact, it is reasonable to allow the interpolant to deviate from them while imposing the looser

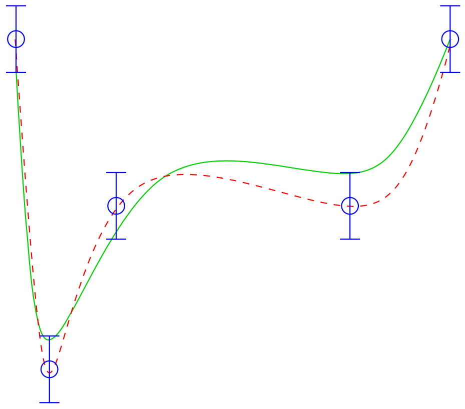


Fig. 2 Function evaluations affected by errors: the values returned by $\tilde{f}$ are the circles and the lower/upper limits of the vertical bars. The dashed line interpolates exactly at the circles, the solid line is a less bumpy model, and is still within the allowed intervals. Problem (12) would prefer the latter model

restriction that the interpolant at x_i should belong to the interval $[f_i^L, f_i^U]$. To do so, instead of solving (4) to determine the interpolant, we solve the problem:

$$\begin{aligned} \min \; & (-1)^{d_{\min}+1}\lambda^T \Phi \lambda \\ \text{s.t.:} \; & f^L \leq \Phi\lambda + Ph \leq f^U \\ & \qquad\quad P^T\lambda = 0_{n+1}. \end{aligned} \tag{12}$$

Problem (12) minimizes the bumpiness of the RBF interpolant, subject to interpolation conditions in which we only require that the value of the surrogate model at $x_1, \dots, x_k$ is within the bounds f^L, f^U. A sketch of this idea is given in Fig. 2. If we set $f_i^L = f_i^U = f(x_i)$ for all i we recover the original system (4). Note that (12) admits at least one solution if (4) admits a solution, however because Φ is not positive semidefinite (although it is positive semidefinite subject to $\Phi\lambda + Ph = F$, see [22]) the problem may not be convex. In our approach, a local optimum is sufficient. After the computation of the RBF model solving (12), the algorithm proceeds as usual to compute the new point x_{k+1}. At every iteration of the fast phase, $\tilde{f}$ is evaluated at x_{k+1}, and $k+1$ is added to the set N of indices of interpolation points affected by error. However, the exact oracle f is immediately reevaluated at x_{k+1}, and $k+1$ is removed from N, if one of the following two conditions hold:

(a) A target objective function value for the problem is known, and it is contained in the interval $[\tilde{f}^\ell(x_{k+1}), \tilde{f}^u(x_{k+1})]$; in other words, x_{k+1} could be optimal.
(b) The value $\tilde{f}(x_{k+1})$ represents a sufficient improvement over known values, and in particular it is a relative improvement (with a given tolerance, set to 10^{-3} in our tests) over the value:

$$f_B := \min \begin{cases} f(x_i) & i \in \{1, \dots, k\} \backslash N, \\ \tilde{f}^\ell(x_i) & i \in N. \end{cases}$$

Note that from a practical perspective, while we cannot expect that the optimal objective function value be known in advance, domain knowledge can often provide a target value and an optimality tolerance such that solutions within the specified tolerance are considered satisfactory, hence condition (a) can be applied. We remark that evaluating x_{k+1} with both $\tilde{f}$ and f is expensive under our assumptions, therefore we want to do it sparingly. However, our empirical evaluation found that it is beneficial to evaluate a new point with the exact oracle if it is sufficiently good: this is because interpolation points evaluated with f act as "anchors" for the surrogate model computed via (12), and we must have at least a few such anchors for the RBF model to be informative. The fast phase ends after a pre-specified number of iterations, or after a pre-specified number of restarts; the latter criterion is due to the fact that restarts are triggered if the search is not making significant progress, and if this happens repeatedly, is is likely that the noisy oracle $\tilde{f}$ is not providing enough information, and we should change our strategy.

We now discuss the second phase. In this phase, all function evaluations employ the exact oracle f. The second phase receives as input from the first phase a set of points, some of which are evaluated via $\tilde{f}$. Recall that Gutmann's RBF algorithm, as described in Sect. 2, does not allow new function evaluations at previously evaluated points $x_1, \dots, x_k$. This is problematic, because it is possible that the optimum of the function is located at a point x_i for $i \in N$, i.e., a point that was previously evaluated with the noisy oracle $\tilde{f}$. We must therefore allow new function evaluations to take place at points $x_i, i \in N$. Our approach is the following. We compute x_{k+1} solving (10) where the target value f_k^* is chosen according to the cyclic strategy described in Sect. 2. We have $x_{k+1} \in \Omega_I \setminus \{x_1, \dots, x_k\}$ by construction. For $w \in N$, let $s_{k,w}$ be the least bumpy RBF model subject to the conditions:

$$\begin{aligned} \tilde{f}^{\ell}(x_i) \ \le s_{k,w}(x_i) \le \tilde{f}^{u}(x_i) & \qquad\qquad \forall i \in N \setminus \{w\}, \\ s_{k,w}(x_i) = f(x_i) & \qquad\qquad \forall i \in \{1, \dots, k\} \setminus N, \\ s_{k,w}(x_w) = f_k^*, & \end{aligned} \tag{13}$$

and let s_k^* be the least bumpy RBF model subject to the conditions:

$$\begin{aligned} \tilde{f}^{\ell}(x_i) \ \le s_k^*(x_i) \le \tilde{f}^{u}(x_i) & \qquad\qquad \forall i \in N, \\ s_k^*(x_i) = f(x_i) & \qquad\qquad \forall i \in \{1, \dots, k\} \setminus N, \\ s_k^*(x_{k+1}) = f_k^*, & \end{aligned} \tag{14}$$

both of which can be computed solving a straightforward modification of (12). When we are in the *Local search* phase of the target value selection strategy, we compare the bumpiness $\sigma(s_k^*)$ with the bumpiness $\sigma(s_{k,w})$ for all $w \in N$ such that $f_k^* \in [f_w^L, f_w^U]$. In other words, we compare the bumpiness of the RBF interpolant at the suggested new point x_{k+1}, with the bumpiness of the RBF model obtained if the function value at x_w were set to the target f_k^*. We do this only for points x_w such that f could take the value f_k^* at x_w, according to the bounds specified by $\tilde{f}$. This way, we can verify whether we obtain a smoother interpolant by placing the target value at an unexplored location, or at one of the previously existing points. If this is the case, i.e., $\sigma(s_{k,w}) < \sigma(s_k^*)$ for some $w \in N$, we evaluate the exact oracle $f(x_w)$ rather than $\tilde{f}(x_w)$, replace the corresponding value in F, and set $N \leftarrow N\backslash\{w\}$. Note that this step will be performed at most $|N|$ times and does not affect global convergence in the limit.

A drawback of the approach described in this section is that it requires the black box to return lower and upper bounds on the function value. A related approach was adopted by [30], whereby all function values are allowed to deviate from the given $f(x_1), \ldots, f(x_k)$, but these deviations are penalized in the objective function in a symmetric way, according to a pre-specified penalty parameter. The difference between our approach and the one of [30] is that we require to specify the range within which function values are allowed to vary, whereas [30] requires to specify the value of the penalty parameter in the objective function and computes the error terms accordingly. We believe that estimating a penalty parameter may prove harder in practice than providing an error range, hence our approach may be more natural for practitioners. Clearly, if the ranges given by $\tilde{f}^\ell, \tilde{f}^u$ are too large the resulting interpolant can be overly smooth and would not provide much information to the optimization procedure, therefore we expect our approach to fail if the error estimates are large compared to the amplitude of the function values.

We remark that problem (12) depends on the bounds $\tilde{f}^\ell(x_i)$ and $\tilde{f}^u(x_i)$ only, rather than the approximate function values $\tilde{f}^x(x_i)$. However, the values $\tilde{f}^x(x_i), i \in N$ are used directly in two steps of the optimization algorithm: (a) we determine a starting solution for (12) by solving (4) with the right-hand side vector set to $\tilde{f}^x(x_i)$ for $i \in N$ and $f(x_i)$ for $i \notin N$; (b) when applying the automatic model selection procedure described in Sect. 3, we again set the right-hand side vector to $\tilde{f}^x(x_i)$ for $i \in N$ and $f(x_i)$ for $i \notin N$.

5 Implementation

We implemented Gutmann's RBF method in Python, using Pyomo [23,24] to model the auxiliary optimization problems.

Our implementation is included in the open-source library called RBFOpt, available in the COIN-OR repository. RBFOpt is released under the Revised BSD license; it can be installed automatically using the Python Package Index (PyPI), and the source code can be downloaded at https://github.com/coin-or/rbfopt. RBFOpt's GitHub page contains links to the documentation and usage examples.

Besides our own version of Gutmann's RBF method, the commercial TOMLAB toolkit for MATLAB contains one, called rbfSolve. A Python toolbox for optimization using surrogate models called pySOT [16] was developed concurrently with RBFOpt. pySOT is based on the general optimization scheme discussed in Sect. 2, but does not implement Gutmann's method.

5.1 Solvers for auxiliary problems

To solve the nonlinear (mixed-integer in the presence of integer variables) optimization problems generated during the various steps of the algorithm, appropriate solvers are necessary. RBFOpt can employ any solver that is compatible with the

AMPL Solver Library. In our tests, we use IPOPT [52] for continuous problems, and BONMIN [6] with the NLP-based Branch-and-Bound algorithm for mixed-integer problems. To optimize nonconvex functions such as (9), we rely on a simple multi-start strategy for IPOPT if there are no integer variables, and on BONMIN's Branch-and-Bound algorithm in the presence of integer variables. BONMIN is parameterized with `num_resolve_at_root` 10, `num_retry_unsolved_random_point` 5, `num_resolve_at_infeasible` 5, with a time limit of 45 seconds. IPOPT is parameterized with `acceptable_tol` 10^{-3}, `honor_original_bounds` no, `max_iter` 1000, and a time limit of 20 seconds. We remark that the performance of the algorithm in practice is dependent on the quality of the solutions to the auxiliary subproblems, and for this reason, the time limit and the number multistart iterations impact the solution process. However, in our experience it is only important to find a good quality solution to these subproblems, rather than an optimal solution. The parameter values above were chosen by observing that in our tests the performance of the algorithm did not increase in a noticeable way by allowing for larger computing time or additional multistart iterations. If CPU time per iteration is an important factor, it is likely that a better trade-off between computing time and solution quality can be achieved by reducing these time limits.

5.2 Addressing numerical issues

In global optimization, numerical troubles arise very frequently in practice, therefore we take special care to ensure that our implementation is stable in a wide range of situations. We briefly describe here the steps taken to prevent numerical problems.

5.2.1 Rescaling

A key decision to be taken in the implementation is the scaling technique for domain and codomain values of the objective function. Rescaling the domain affects the left-hand side of the system (4), while rescaling the codomain affects the right-hand side. [26] suggests transforming the domain of f into the unit hypercube. This strategy is implemented in the *rbfSolve* function of the MATLAB toolkit TOMLAB. In our tests, we found this transformation to be beneficial only when the bounds of the domain are significantly skewed. When all variables are defined over an interval of approximately the same size we did not observe any benefit from this transformation, and in fact sometimes the performance deteriorated. Note that the transformation cannot be applied on integrality-constrained variables. After computational testing, our default strategy is to transform the domain into the unit hypercube on problems with no integer variables and such that the ratio of the lengths of the largest to smallest variable domain exceeds a given threshold, set to 5 by default.

To prevent harmful oscillations of the RBF interpolant due to large differences in the function values, it has been proposed to rescale the codomain of f, see e.g. [27,48]. Our numerical tests show that rescaling is necessary when the original objective function has values with large magnitude, or exhibits large oscillations. Our default approach is to apply a logarithmic scaling whenever the difference between the median and the minimum objective function value among the interpolation points exceeds a certain threshold, set to 10^6 by default, and leave the function values unscaled otherwise. When the logarithmic scaling is applied, if $f_{\min} \geq 1$ we replace each $f(x_i)$ with $\log(f(x_i))$, otherwise we replace it with $\log(f(x_i) + 1 + |f_{\min}|)$ (similar to [27]). We additionally tested affine function scaling (replacing each $f(x_i)$ with $\frac{f(x_i)-f_{\min}}{f_{\max}-f_{\min}}$), but it did not show any advantage over unscaled function values in our tests.

In addition to rescaling the function codomain if necessary, we follow the approach suggested by [22] consisting in clipping the function values $f(x_i)$ at the median (in other words, we replace values larger than the median by the median). The procedure is also adopted by [5,46]. We follow this approach with one small change: function values are clipped at the median only if the ratio of the largest to smallest absolute function value exceeds a given threshold, set to 10^3 by default.

5.2.2 Ill-conditioned linear system and restoration step

When two or more interpolation points are close to each other, the matrix A_k of (4) and (8) becomes nearly singular. This can have serious consequences: we may not be able to compute the RBF interpolant, or (8) could become numerically ill-behaved. We have several mechanisms in place to prevent this from happening. First, at every iteration, say iteration k, we only accept the new point x_{k+1} if $\min_{i=1,\dots,k} \|x_{k+1} - x_i\|$ is larger than a given threshold, set to 10^{-5} by default. Second, if the problem has integer variables and the initial sample points are generated according to a Latin hypercube design, we check the smallest singular value of the matrix Φ obtained after rounding the points. If this singular value is close to zero, we generate a different Latin hypercube. Finally, if at any iteration we cannot solve the linear system (4) or invert the matrix A_k, we perform a restoration step: the last iterate x_k is discarded and replaced with a new point. In our experiments the restoration step was never necessary thanks to the other numerical safety mechanisms, hence we do not provide details here.

5.2.3 Starting point for local solvers

The auxiliary problems solved during the course of Gutmann's RBF method are highly nonconvex, and it can be difficult for a nonlinear solver to achieve feasibility on formulations that use auxiliary variables defined through equality constraints, such as those in our implementation. On the other hand, generating an initial feasible solution exploiting the problem structure is very easy. Thus, when solving an auxiliary problem, we generate an initial feasible solution by sampling a point uniformly at random over the domain and setting the auxiliary variables accordingly to make sure all the equality constraints are satisfied. We provide this feasible solution as a starting point for the local solver. As mentioned in Sect. 4, for the solution of problem (12) we use a local solver starting from the solution of (4).

6 Computational experiments

In this section we discuss computational experiments performed with RBFOpt version 1.0. All experiments were carried out on a server equipped with four Intel Xeon E5-4620 CPUs (2.20 GHz, 8 cores, Hyper Threading and Turbo Boost disabled) and 128GB RAM (32GB for each processor), running Linux.

6.1 Test instances

We test our implementation on 20 box-constrained continuous problems (called NLP in the following) and 16 box-constrained integer or mixed-integer problems (called MINLP in the following), taken from the literature. The problems are listed in Table 2. Most of these problems are nonconvex with multiple local minima. We provide more information on the instances below. Many of these problems were originally proposed as a testbed for global optimization solvers, and are now considered fairly easy in terms of global optimization. However, they can prove very challenging for black-box solvers that do not exploit analytical information on the problems.

- Dixon–Szegö [15] problems: we included the most common instances in the test set. These instances are used in all computational evaluations of Gutmann's RBF method and in many other derivative-free approaches, see e.g., [22,26,46].
- GLOBALLIB problems: we selected a subset of the unconstrained instances in the library. Some problems were excluded because they were too easy or too similar to other problems in our collection.
- Schoen [50] problems: we randomly generated two problems of dimension 6 and two of dimension 10. All problems have 50 stationary point, three of which are global minima with value -1000, and the remaining ones attain a value picked uniformly at random in the interval [0, 1000]. Having steep global minima allows us to test the performance of our implementation in a situation that is considered difficult to handle for Gutmann's RBF method, see [46]. Another advantage of using problems of this class is that we can choose the dimension of the space. In particular, we test problems with 10 decision variables, which is larger than the other continuous problems in our test set. To avoid overrepresentation of a class of instances in our test set, we generate only four random problems of this class.
- Neumaier [43] problems: we included one problem of class "perm", and one of class "perm0", generated with parameters $n = 6, \beta = 60$ and $n = 8, \beta = 100$ respectively. These problems were conceived to be challenging for global optimization solvers, and are extremely difficult to solve with a black-box approach in our experience. The global minimum of these instances is originally 0, but achieving a near-optimal solution in relative or even absolute terms (e.g., less than 1% or 0.01 from the optimum) is essentially hopeless for these problems. Hence, we translated the functions up by 1000.
- MINLPLib2 [7,36] problems: we included all unconstrained instances in the library, and several problems with at most three constraints that were reformulated as box-constrained problems by penalizing the constraint violation in the objective function. The penalty parameter was chosen so that that global mini-

Table 2 Details of the instances used for the tests

Instance	Dimension (# integer)	Domain	Type	Source
branin	2 (0)	$[-5, 10] \times [0, 15]$	NLP	Dixon–Szegö [15]
camel	2 (0)	$[-3, 3] \times [-2, 2]$	NLP	Dixon–Szegö [15]
ex4_1_1	1 (0)	$[-2, 11]$	NLP	GLOBALLIB
ex4_1_2	1 (0)	$[1, 2]$	NLP	GLOBALLIB
ex8_1_1	2 (0)	$[-1, 2] \times [-1, 1]$	NLP	GLOBALLIB
ex8_1_4	2 (0)	$[-2, 4] \times [-5, 2]$	NLP	GLOBALLIB
goldsteinprice	2 (0)	$[-2, 2]^2$	NLP	Dixon–Szegö [15]
hartman3	3 (0)	$[0, 1]^3$	NLP	Dixon–Szegö [15]
hartman6	6 (0)	$[0, 1]^6$	NLP	Dixon–Szegö [15]
least	3 (0)	$[0, 600] \times [-200, 200] \times [-5, 5]$	NLP	GLOBALLIB
perm0_8	8 (0)	$[-1, 1]^8$	NLP	Neumaier [43]
perm_6	6 (0)	$[-6, 6]^6$	NLP	Neumaier [43]
rbrock	2 (0)	$[-10, 5] \times [-10, 10]$	NLP	GLOBALLIB
schoen_10_1	10 (0)	$[0, 1]^{10}$	NLP	Schoen [50]
schoen_10_2	10 (0)	$[0, 1]^{10}$	NLP	Schoen [50]
schoen_6_1	6 (0)	$[0, 1]^6$	NLP	Schoen [50]
schoen_6_2	6 (0)	$[0, 1]^6$	NLP	Schoen [50]
shekel10	4 (0)	$[0, 10]^4$	NLP	Dixon–Szegö [15]
shekel5	4 (0)	$[0, 10]^4$	NLP	Dixon–Szegö [15]
shekel7	4 (0)	$[0, 10]^4$	NLP	Dixon–Szegö [15]
gear	4 (4)	$[12, 60]^4$	MINLP	MINLPLib2 [36]
gear4	5 (4)	$[12, 60]^4 \times [0, 100]$	MINLP	MINLPLib2 [36]
nvs02	5 (5)	$[0, 200]^5$	MINLP	MINLPLib2 [36]
nvs03	2 (2)	$[0, 200]^2$	MINLP	MINLPLib2 [36]
nvs04	2 (2)	$[0, 200]^2$	MINLP	MINLPLib2 [36]
nvs06	2 (2)	$[1, 200]^2$	MINLP	MINLPLib2 [36]
nvs07	3 (3)	$[0, 200]^3$	MINLP	MINLPLib2 [36]
nvs09	10 (10)	$[3, 9]^{10}$	MINLP	MINLPLib2 [36]
nvs14	5 (5)	$[0, 200]^5$	MINLP	MINLPLib2 [36]
nvs15	3 (3)	$[0, 200]^3$	MINLP	MINLPLib2 [36]
nvs16	2 (2)	$[0, 200]^2$	MINLP	MINLPLib2 [36]
prob03	2 (2)	$[1, 5]^2$	MINLP	MINLPLib2 [36]
sporttournament06	15 (15)	$[0, 1]^{15}$	MINLP	MINLPLib2 [36]
st_miqp1	5 (5)	$[0, 1]^5$	MINLP	MINLPLib2 [36]
st_miqp3	2 (2)	$[0, 3] \times [0, 50]$	MINLP	MINLPLib2 [36]
st_test1	5 (5)	$[0, 1]^5$	MINLP	MINLPLib2 [36]

mizers of the penalized objective are exactly the global minimizers of the original constrained problem. All problems in this class, except one, have integer variables only: despite MINLPLib2 being the largest available library of MINLP problems, all mixed-integer problems contained in it are either too large or too heavily constrained for us to handle.

An extensive computational evaluation of black-box solvers is discussed in [49], which uses a much larger test set than ours. However, the setting of that paper is different because some variables are unbounded or with large bounds, the problem dimension is typically higher, and a larger number of function evaluations is allowed (up to 2500, while we limit ourselves to $30(n_p + 1)$ where n_p is the dimension of problem p). [49] does not provide computational results for any implementation of Gutmann's RBF method or its derivatives despite discussing them, and we attribute this to the fact that Gutmann's RBF method targets problem with bounded variables on which few function evaluations are allowed. Furthermore, since we test many variants of the algorithms with 20 different random seeds per instance, a larger set would have required prohibitive computation times.

6.2 Evaluation methodology

Our evaluation relies on performance profiles and data profiles, as described in [37]. We review their definition.

Define the *budget* for an algorithm as the maximum number of function evaluations allowed. In our experiments the budget is set to $30(n_p + 1)$. This is a relatively small number relative to other computational studies of derivative-free methods in the literature, but in our experience it is a reasonable number for many real-world applications, and reinforces our focus on small-dimensional problems on which we aim to achieve very fast convergence.

For a given instance and a set of algorithms $\mathcal{A}$, let f^* be the best function value discovered by any algorithm, and x_0 the first point evaluated by each algorithm, which we impose to be the same. Let τ be a tolerance; we use 10^{-3} in this paper. We also tried $\tau = 10^{-2}$, but we do not report the corresponding results unless they yield further insight. We say that an algorithm *solves* an instance if it returns a point $\bar{x}$ such that:

$$f(x_0) - f(\bar{x}) \geq (1 - \tau)(f(x_0) - f^*), \tag{15}$$

and the algorithm *fails* otherwise.

Let $\mathcal{P}$ be the set of problem instances in the test set. Let $t_{p,a}$ be the number of function evaluations required by algorithm a to solve problem p ($t_{p,a} = \infty$ if algorithm a fails on problem p according to the convergence criterion (15)), and n_p the number of variables of problem p. The *data profile* for an algorithm a is the fraction of problems that are solved within budget $\alpha(n_p + 1)$, formally defined as:

$$d_a(\alpha) := \frac{1}{|\mathcal{P}|} \text{size} \left\{ p \in \mathcal{P} : \frac{t_{p,a}}{n_p + 1} \leq \alpha \right\}.$$

The scaling factor $n_p + 1$ tries to account for the fact that problems with more decision variables are expected to be more difficult.

The performance ratio of algorithm a on problem p is defined as:

$$r_{p,a} := \frac{t_{p,a}}{\min\{t_{p,a} : a \in \mathcal{A}\}}.$$

Note that the performance ratio is 1 for the best performing algorithm on a problem instance. The performance profile of algorithm a is defined as the fraction of problems where the performance ratio is at most α, formally defined as:

$$p_a(\alpha) := \frac{1}{|\mathcal{P}|} \text{size} \left\{ p \in \mathcal{P} : r_{p,a} \le \alpha \right\}.$$

Since the algorithms tested in this paper include some elements of randomness (e.g., the initial points), every algorithm is executed 20 times on each problem instance with different random seeds. Each of the 20 runs is associated with an initial experimental design that is given to all algorithms (if the algorithm requires a single initial point, we provide the first point of the experimental design), to make the initialization phase more uniform across the algorithms and reduce variance. We use the median over 20 executions of the best objective function value known at every iteration to compute data and performance profiles. In addition to this, we built graphs using the best objective function value over 20 executions rather than the median, but we find the median to be more indicative of the overall performance, because in our experience out of 20 executions it is common to have an exceptional run that performs much better than the remaining ones.

We remark that our choice of using $\tau = 10^{-3}$ is dictated by the fact that the method we are testing is essentially a global method, therefore convergence to very high levels of precision can be too slow for efficient benchmarking.

We would like to answer the following research questions, that are investigated in Sects. 6.3 through 6.6:

1. Which algorithmic configuration is the best, and in particular, are the improvements of Sect. 2.4 beneficial in practice?
2. Is our approach to handle noisy function evaluations effective?
3. Is automatic model selection using cross validation beneficial in practice?
4. Is our implementation competitive with the state-of-the-art?

6.3 Comparison of algorithmic settings

In this section we are concerned with investigating the effect of the most important algorithmic parameters of RBFOpt. The following list summarizes the different settings that we considered, see Sect. 2.4 for details:

– "R": restart the algorithm after 6 cycles without improvement of the best solution found [item (b) in Sect. 2.4];

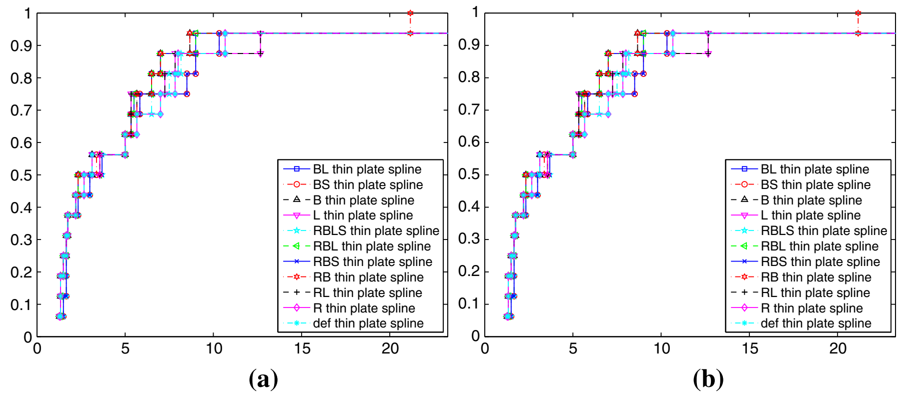


Fig. 3 Data profiles for different parametrizations of RBFOpt. **a** Median NLP, $\tau = 10^{-3}$. **b** Median MINLP, $\tau = 10^{-3}$

- "B": apply restricted global minimization of the bumpiness function [(item (c) in Sect. 2.4];
- "L": if the local search step improves the best solution, it is immediately repeated a second time;
- "S": in (11), disable the dynamic selection of $f(x_{\pi(\alpha(k))})$ to replace the statically chosen $f_{\max}$ [item (a) in Sect. 2.4].

The basic configuration, labeled "def", employs the thin plate spline basis function, a random Latin Hypercube design (generated with the maximum minimum distance criterion) for the selection of the initial sample, no InfStep, and 5 global search steps (i.e., $\kappa = 5$). This is similar to [22,46], with the only difference being the choice of the radial basis function: in our experiments, thin plate splines tend to perform better than the other types of RBFs with all tested parametrizations of the algorithm, therefore we report results with thin plate splines. This is supported by the data reported in Sect. 6.5.

We tested most of the possible combinations or "R", "B", "L" and "S" when applied on top of our default configuration. We report data profiles in Fig. 3, and performance profiles in Fig. 4. Recall that the x-axis refers to the number of function evaluations in data profiles, whereas it represents upper bound values for the performance ratio of the algorithms in performance profiles. As can be seen in Fig. 3, all configurations of the algorithm coincide on data profiles for the first few data points, but they start to diverge after $\approx 3(n_p + 1)$ iterations. This is consistently true throughout our analysis. The performance profiles are clearer and allow us to rank the algorithms. For this reason, in the rest of the paper we only report performance profiles.

From the figures, we see that the settings "L" (repeat the local search phase if successful) and "S" (use a static value of $f_{\max}$ instead of dynamic selection) are generally detrimental for both NLP and MINLP problems, and should therefore be disabled. The detrimental effect of "S" is obvious on MINLP problems, where there are often very large differences between the function values at the initial sample points and at the minimum. Interestingly, the simple idea of repeating a local search whenever successful has a negative impact in the long run. We attribute this to two facts: first, the

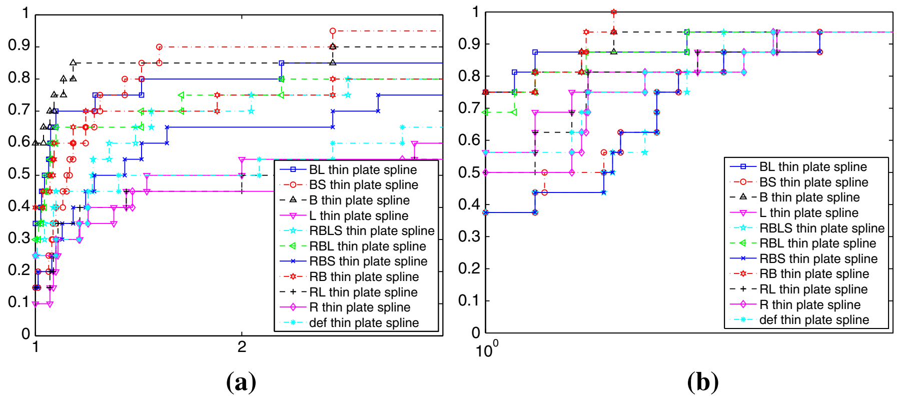


Fig. 4 Performance profiles for different parametrizations of RBFOpt. **a** Median NLP, $\tau = 10^{-3}$. **b** Median MINLP, $\tau = 10^{-3}$

accuracy of the surrogate model typically improves only slightly after a local search, therefore the second local search in a row uses essentially the same information as the previous one, which hinders its chances of success; second, the strength of Gutmann's RBF algorithm is the effectiveness of the global search, and delaying the global search by introducing additional local search steps is not a good idea. However, we remark that in our experiments the "L" setting performs very well if we consider the best run out of the 20 trials per instance, instead of the median: this is because the "L" setting aggressively tries to improve the objective function value, and this typically allows one of the 20 trials to converge very quickly.

The best performing parametrization on NLP problems is clearly "B" and its variants ("BL", "BS"), followed by "RB" and its variants. These two groups obtain very similar performance for $\tau = 10^{-3}$. On MINLP problems, "RB" is arguably the best performing parametrization, followed closely by "B". On both NLP and MINLP problems, the improvement with respect to "def" is very significant. The ranking of the various parametrizations in terms of performance does not change too much between NLP and MINLP (notable exceptions are the "S" variants), suggesting that some parametrizations may be better altogether, regardless of the class of problems.

A closer analysis of the results suggested that the excellent performance of "B" with $\tau = 10^{-3}$ on NLP problems is due to occasional restarts of "RB" when relatively close to the optimum because the relative improvement criterion is not satisfied, i.e., only small improvements are recorded and the algorithm detects that stalling has occurred, forcing a restart. The "B" parametrization does not allow restarts, and repeated local searches in the region suspected to contain the optimum eventually pay off. However, this comes at a rather large computational cost: "B" is, on average, $\approx$ 4.3 times slower than "RB", because the time per iteration decreases significantly after restarts thanks to a smaller number of interpolation nodes and easier auxiliary problems. When a smaller precision is required for convergence, i.e., $\tau = 10^{-2}$ (not reported in the paper), "RB" already performs as well as "B" on NLP problems. On MINLP problems, our experience is that restarts can be very important: occasionally, the local search phase of the algorithm becomes ineffective because all the integer points around the suspected optimum of the RBF model have already been evaluated, and no further improvement is achieved. In these cases, a restart is often beneficial, hence "RB" tends to perform better than "B" on MINLP problems. Hereafter, we use "RB" as our default configuration.

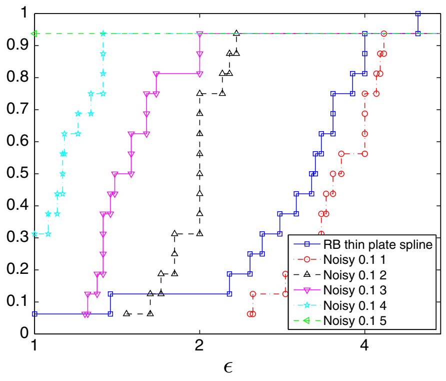
6.4 Experiments with a noisy oracle

We want to assess the effectiveness of the method proposed in this paper to exploit access to a noisy oracle $\tilde{f}$ that is faster than the exact oracle f. For ease of benchmarking, we need a way to simulate $\tilde{f}$. Our approach is to simulate the noisy oracle by applying to f a relative noise generated uniformly at random between $\pm 10\%$ or $\pm 20\%$, as well as an absolute noise generated uniformly at random between ± 0.01 (to avoid exact oracle evaluations around zero). Here we report results with a relative error of $\pm 10\%$. We performed experiments with a relative error of $\pm 20\%$, and the conclusions are similar. The lower and upper bounding function values $\tilde{f}^\ell$, $\tilde{f}^u$ returned by the noisy oracle are computed as $\tilde{f}^\ell(x) = \tilde{f}^x(x) - \epsilon_r|\tilde{f}^x(x)| - \epsilon_a$, $\tilde{f}^u(x) = \tilde{f}^x(x) + \epsilon_r|\tilde{f}^x(x)| + \epsilon_a$, where ϵ_r, ϵ_a are parameters of the numerical experiments that determine how accurately the bounding functions estimate the true amount of noise applied to f.

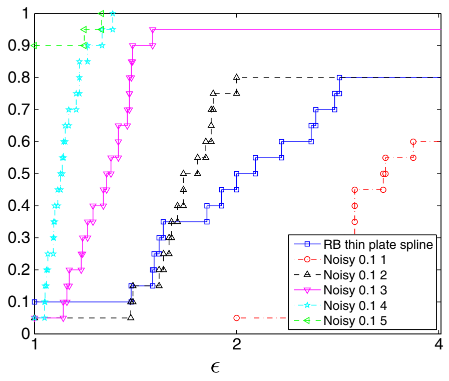
Remember that the approach described in Sect. 4 uses both the exact and the noisy oracle, in a "fast" phase and an "accurate" phase. To build data and performance profiles, at every point x_i evaluated with the noisy oracle $\tilde{f}$ we attribute the worst-case objective function value $\tilde{f}^u(x_i)$. We assume that each noisy oracle evaluation has a computational cost equal to $1/c$ of an exact oracle evaluations, where $c \geq 1$ is a parameter and we test $c = 1, \dots, 5$. In other words, the total number of function evaluations at any given time is computed as (# exact evaluations) + (# noisy evaluations)/c. Even though the maximum relative error applied to f is fixed at $\pm 10\%$, we set ϵ_r to three different values in our experiments: 10%, 30%, 50% (we tested additional values but we do not report them for space reasons). This corresponds to overestimating the true error by up to a factor 5. The absolute variation ϵ_a is always set to 0.01. By varying c and ϵ_r, we can determine for which speed/accuracy trade-offs it is advantageous to exploit the noisy oracle rather than rely on the exact oracle only. The limits for the algorithm are set to $30(n_p + 1)$ evaluations for f, $60(n_p + 1)$ evaluations for $\tilde{f}$, and at most $20(n_p + 1)$ iterations in the "fast" phase. The limit on the number of evaluations of $\tilde{f}$ was never reached. In the graphs we only report the equivalent to the first $30(n_p + 1)$ exact evaluations, to conform to the rest of the paper.

Figure 5 contains performance profiles to compare the "RB" configuration of RBFOpt with and without exploitation of the noisy oracle. For simplicity, in the rest of this section we call the two version of the algorithm "exact" and "noisy" respectively. only report $\tau = 10^{-3}$,

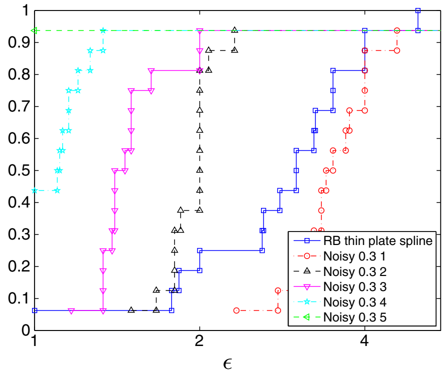
The graphs show that "noisy" performs worse than "exact" for $c = 1$, which is expected, but already for $c = 2$ "noisy" is superior as long as the error estimate is accurate, i.e., $\epsilon_r = 10\%$. Even if the error estimate is very loose, i.e., $\epsilon_r = 50\%$, "noisy" typically converges faster than "exact" for $c \geq 3$, as can be seen from the



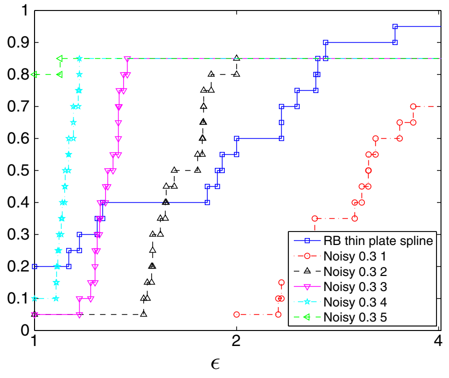
(a)



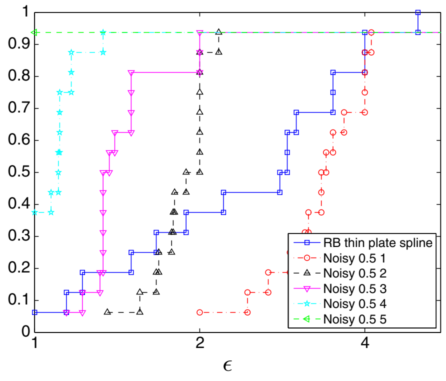
(b)



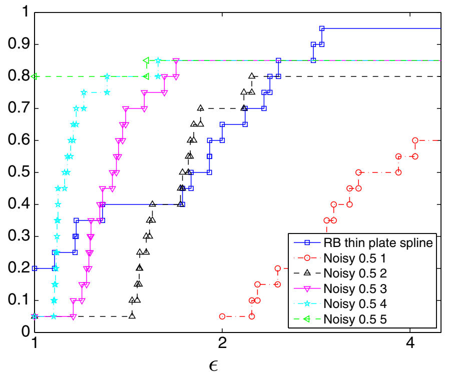
(c)



(d)



(e)



(f)

Fig. 5 Performance profiles for "RB" with different surrogate models and $\tau = 10^{-3}$. The labels for the "noisy" algorithm indicate the value of ϵ_r, which determines the looseness of the lower and upper bounding functions $\tilde{f}^\ell$, $\tilde{f}^u$, and c. **a** Median MINLP, $\epsilon_r = 10\%$, $c = 1, \dots, 5$. **b** Median NLP, $\epsilon_r = 10\%$, $c = 1, \dots, 5$. **c** Median MINLP, $\epsilon_r = 30\%$, $c = 1, \dots, 5$. **d** Median NLP, $\epsilon_r = 30\%$, $c = 1, \dots, 5$. **e** Median MINLP, $\epsilon_r = 50\%$, $c = 1, \dots, 5$. **f** Median NLP, $\epsilon_r = 50\%$, $c = 1, \dots, 5$

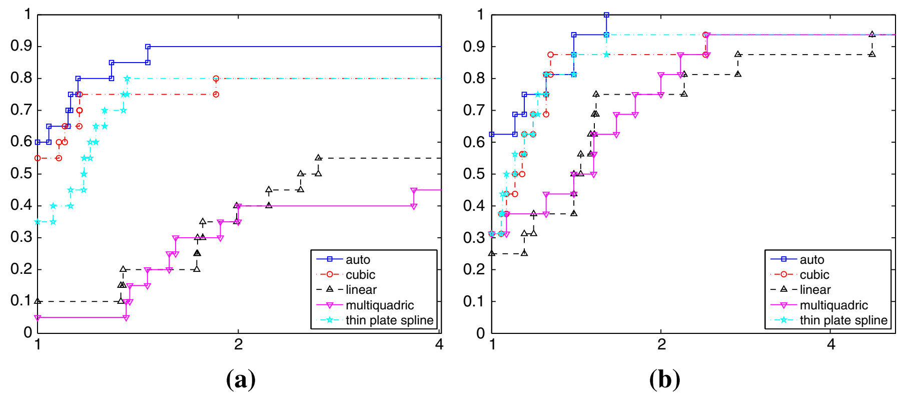


Fig. 6 Performance profiles for "RB" with different surrogate models. **a** "RB", median NLP, $\tau = 10^{-3}$. **b** "RB", median MINLP, $\tau = 10^{-3}$

performance profiles. When $c = 4$ or 5, corresponding to a situation in which the noisy oracle is computationally inexpensive, our approach to exploit the noisy evaluations is largely superior to the traditional "exact" algorithm, converging significantly faster on the vast majority of the MINLP and NLP instances. We remark that there is a small fraction of the instances (MINLP and NLP) on which "exact" converges but "noisy" does not within the equivalent of $30(n_p + 1)$ "exact" iterations, but overall, our results show that the "noisy" approach described in Sect. 4 can be very beneficial in practice and find good solutions much faster than the traditional Gutmann's RBF algorithm. To summarize, in applications where it is possible to obtain a fast but noisy oracle for the unknown objective function, and a reasonable error estimate for such noisy oracle is available, the methodology that we propose can yield a much faster convergence.

6.5 Automatic model selection using cross validation

We now test the automatic model selection method presented in Sect. 3. We label this configuration "auto", as opposed to the default configuration that uses a predetermined basis function. We test the "auto" configuration against four commonly used types of basis functions: linear, cubic, thin plate spline, and multiquadric. In Fig. 6 we compare the results obtained with and without automatic model selection using cross validation, using the "RB" parametrization.

The first observation we make from the graphs is that thin plate splines and cubic perform better than linear and multiquadric on our test set. The two RBFs have comparable performance, with thin plate splines emerging as the marginally better choice on MINLP problems, and cubic on NLP; however, performance profiles constructed without the "auto" configuration suggest that thin plate splines is generally the best choice of RBF, even on NLP (performance profiles depend on the set of tested algorithms, and removing one algorithm from the set can alter the resulting ranking). This is consistent with our experience on simulation-based optimization problems that are not part of this benchmark, see e.g. [54].

Table 3 Average CPU times, in seconds, (standard deviation reported in brackets) to perform leave-one-out cross validation using brute force computation or the procedure based on reusing the LU factorization. Each point is 10-dimensional and we perform full leave-one-out cross validation, i.e., for k points we solve k linear system, each excluding one point

# Points	Brute force avg. (SD) [s]	LU-based avg. (SD) [s]
20	0.0035 (0.0005)	0.0011 (0.0002)
50	0.0082 (0.0048)	0.0017 (0.0007)
100	0.0217 (0.0196)	0.0030 (0.0019)
200	0.0956 (0.1300)	0.0058 (0.0051)
500	1.5753 (3.0150)	0.0297 (0.0545)
1000	62.6155 (1.0256)	1.1477 (0.0477)

The second observation is that the automatic model selection method has better performance than any of the individual RBFs, achieving faster convergence across the board. This is noteworthy because automatic model selection not only improves performance, but eliminates the difficult choice of a parameter for the algorithm (the choice of RBF). The benefits of the automatic model selection method were confirmed also in additional numerical experiments not reported here, i.e., with different values of τ, and with other RBF-based optimization methods following the scheme discussed in Sect. 2.

In the experiments in this section we use the methodology described in Sect. 3 to perform the leave-one-out cross validation procedure. The improvement over a brute force computation is significant: we provide some numbers in Table 3 to give a sense of the gain in terms of speed. The table reports the CPU time required to perform brute force cross validation resolving all linear systems from scratch, and to perform the same procedure reusing the LU factorization. The numbers are computed over 40 trials with randomly generated interpolation points, for a 10-dimensional problem. Already for a 10-dimensional problem, the savings can be of almost two orders of magnitude: the brute force computation is only viable for small problems and with a small number of interpolation points, while the procedure that we propose employs a fraction of a second even with 500 points.

To summarize, the automatic model selection procedure shows excellent performance on our test set, improving over each of the individual RBFs. If a single RBF has to be selected, cubic and thin plate splines are to be preferred over linear and multiquadric. The computational overhead of running the automatic model selection procedure every few iterations of the optimization algorithm is very small, using the scheme proposed in this paper.

6.6 Comparison with other existing software

We now compare the performance of RBFOpt with other derivative-free optimization software. Despite the fact that there is a large body of literature in this area of optimization, the availability of free software is considerably less substantial. Rios [49] benchmarks numerous solvers, but as discussed above, many of those are not suitable for the type of problems discussed here. We remark that in order to plot performance profiles, we must be able to determine the value of every single function evaluation performed by the algorithm, and we cannot simply rely on results reported in the literature. Hence, our comparison is limited to software that we could obtain and run within a reasonable amount of time. After examining the available possibilities, we decided to include the following software in our comparison:

– NOMAD 3.7.1: NOMAD [35] is an open-source C++ implementation of MADS [2], and it is well documented and supported. NOMAD can be applied on both continuous and discrete problems. We test two parametrizations of NOMAD: one uses a pure MADS search, and the other uses a hybrid Variable Neighborhood Search (VNS) / MADS search (parameter `VNS_SEARCH 0.75`, as suggested by the documentation). VNS is supposed to help the algorithm escape local minima, increasing the global search capabilities. The other parameters are left to default values.
– SNOBFIT 2.1: SNOBFIT implements the Branch-and-Fit algorithm of [28], which subdivides the domain into hyperrectangles, and fits a quadratic model for each rectangle. The algorithm is globally convergent. We used the available MATLAB implementation. SNOBFIT handles continuous box-constrained problems, and is one of the best performing solvers in the tests of [49].
– KNITRO 9.1: KNITRO [8] is a commercial optimization software package that includes several algorithms. In particular, it has an SQP algorithm that is designed for derivative-free problems for which the number of function evaluations has to be kept small. The gradient of the objective function is estimated via forward finite-differences, and the Hessian is estimated via quasi-Newton BFGS (option `hessopt = 2`, label "H2" on the graphs), which constrains the Hessian to be positive semidefinite, or via quasi-Newton SR1 (option `hessopt = 3`, label "H3" on the graphs). KNITRO was the best performing derivative-free solver in the 2015 GECCO Black-Box Competition, see Sect. 6.7.
– BOBYQA: BOBYQA [45] is a local algorithm for derivative-free bound-constrained optimization that uses a quadratic approximation of the objective function. We use the implementation given in NLopt 2.4.2 [31].
– DIRECT: the DIRECT algorithm [32] is a well-known global algorithm for derivative-free bound-constrained optimization, based on dividing the domain in hyper-rectangles. We test the original algorithm, label "DIRECT", as well as the locally-biased modification proposed in [18], label "DIRECT-L", as implemented in NLopt.
– Nelder–Mead: the Nelder–Mead simplex algorithm [42] is one of the most enduring algorithms for derivative-free optimization, dating back from the 60s. Despite its lack of strong convergence properties, it is very popular and we test its implementation given in NLopt.

The budget of function evaluations is the same for all solvers, and set to $30(n_p + 1)$ where n_p is the number of variables of problem p. All local optimization algorithms (NOMAD, KNITRO, BOBYQA, Nelder–Mead) are embedded into a multistart algorithm that is executed until the budget is depleted. The starting point for the first multistart iteration is the same used for RBFOpt, as is required for the computation of

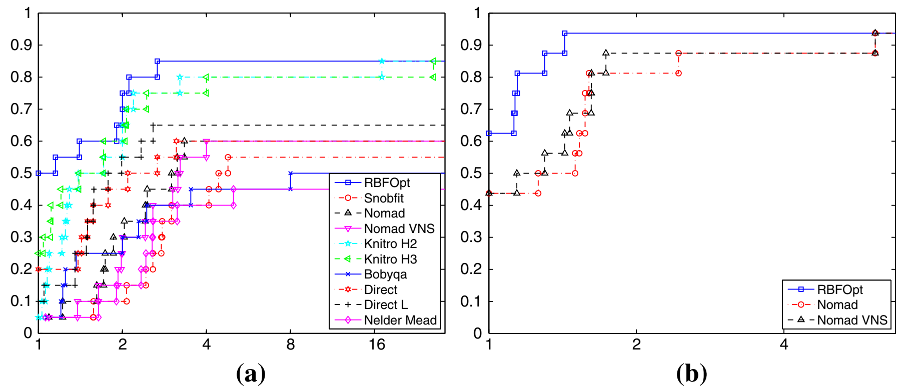


Fig. 7 Performance profiles for different parametrizations of RBFOpt and several algorithms taken from the literature. **a** Median NLP, $\tau = 10^{-3}$. **b** Median MINLP, $\tau = 10^{-3}$

performance and data profiles. We use the same framework as for RBFOpt, i.e., we perform 20 runs with different random seeds for each problem instance, and take the median value of the best known point for each run at every iteration.

Performance profiles are reported in Fig. 7. We use the "B" parametrization of RBFOpt for NLP problems, and "RB" for MINLP problems, which are the best for the respective class. We use thin plate splines as RBF, disabling the automatic model selection because none of the other solvers uses a similar model selection approach, even though some of them could in principle benefit from it (e.g., NOMAD allows usage of surrogate models, but not in its default version). Of the solvers described above, only NOMAD supports integer or mixed-integer domains. Thus, the comparison on our MINLP test set is restricted to RBFOpt and NOMAD only.

The results show that RBFOpt is highly competitive with existing software, and in fact it typically performs better on our test set, even without the automatic model selection procedure that would further improve its performance. On continuous problems, the best performing algorithm is RBFOpt: in the performance profiles, KNITRO is slightly below RBFOpt. All remaining solvers are not competitive on this test set, and are significantly outperformed for any budget of function evaluations within the $30(n_p+1)$ tested here. On mixed-integer problems, RBFOpt once again clearly outperforms NOMAD, although the difference does not seem to be as large as on continuous problems. NOMAD eventually converges on the same number of problems as RBFOpt, but the performance profiles indicate that RBFOpt is more consistent on the mixed-integer test set: RBFOpt's curve lie fully above NOMAD's.

6.7 Results of the 2015 GECCO black-box competition

RBFOpt was one of the 28 participants of the 2015 GECCO Black-Box Competition, whose results can be found at http://bbcomp.ini.rub.de/results/BBComp2015GECCO/summary.html. The problem instances for the competition are of dimension up to 64 and with a budget up to 6400, which is larger than the problems Gutmann's RBF algorithm is usually applied to. More specifically, while the budget was set to $30(n_p + 1)$

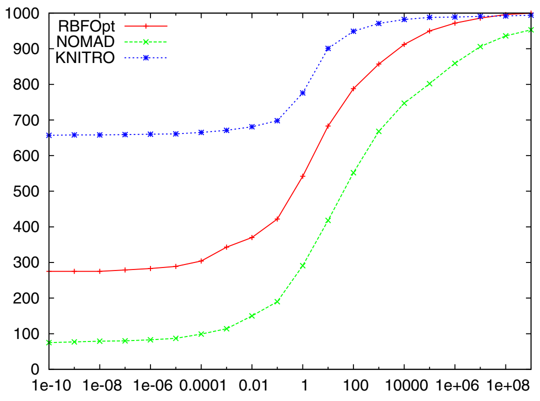


Fig. 8 Number of problems in the 2015 GECCO Black-Box Competition solved within a certain (relative and absolute) tolerance. For each value k on the x-axis, each curve indicates on how many instances out of 1000 each solver found a solution at least as good as $f^* + k \max\{|f^*|, 1\}$, where f^* is the best value returned by any algorithm

in the experiments discussed so far, the budget was as large as $100n_p$ in the competition. Our implementation performed relatively well, ranking seventh overall and first among the open-source solvers. Two of the algorithms discussed in Sect. 6.6, KNITRO and NOMAD, participated to the competition as well, ranking first and twentieth respectively.

RBFOpt was parametrized with the "RBL" configuration and automatic model selection; in hindsight, this choice may not have been ideal because our experiments in this paper suggest that "B" usually performs better on continuous problems, although it is slower in terms of time per iteration. This parametrization restarted multiple times on instances with large budget, which may have been detrimental. The code used in the competition was a version of RBFOpt prior to 1.0, and the automatic model selection procedure did not employ an order-based performance metric as described in this paper. In our tests this negatively affects performance in a significant way. We did not encounter any fatal numerical issue despite the fact that several of the black-box functions were very ill-conditioned, with amplitudes exceeding 10^{20}: this indicates that our implementation is numerically stable.

To give a sense of the performance of RBFOpt as compared to KNITRO and NOMAD, we report in Fig. 8 a graph indicating the number of problems for which each solver found a solution within a given (absolute or relative) distance from the best solution found by any of the three solvers. Clearly KNITRO is the winner here, and RBFOpt performed much better than NOMAD. We attribute the better performance of KNITRO to the larger budget of function evaluations: estimating the gradient at each point requires $(n_p + 1)$ evaluations, therefore KNITRO could perform at most 30 major iterations in the experiments of Sect. 6.6, whereas here the budget is not so tight.

7 Conclusions

In this paper we provided an overview of Gutmann's RBF method for black-box optimization, which is considered one of the best surrogate model based methods for derivative-free optimization. We proposed some modifications of the algorithm with the aim of improving practical performance. Besides the numerically stable open-source implementation, our two main contributions are an efficient methodology to perform automatic model selection using a cross-validation scheme, and an approach to exploit noisy but faster function evaluations. Computational experiments show that automatic model selection improves over each of the individual radial basis functions and eliminates the need for the user to pre-select the type of surrogate model, and exploitation of a noisy oracle yields a noticeable reduction in the number of function evaluations to achieve convergence, when reasonable error bounds are available. Our tests suggest that parametrizations of the algorithm that are effective on continuous problems are typically effective on mixed-integer problems as well. One exception is given by complete restarts: in our experiments, allowing the algorithm to restart from scratch when no progress is detected is often harmful on continuous problems, while we found it to be beneficial on mixed-integer problems. However, complete restarts of the algorithm can significantly improve the average CPU time per iteration, and therefore they are enabled in the default settings of our implementation.

The software discussed in this paper is available as part of the COIN-OR repository in a library called RBFOpt. On our test set that comprises low-dimensional problems (up to 15 variables) with a small budget of function evaluations, RBFOpt is competitive with all derivative-free solvers included in our computational study: it obtains significantly better results than other open-source solvers, and slightly better results than a commercial solver.

Acknowledgements The authors are grateful for partial support by the SUTD-MIT International Design Center under grant IDG21300102. The research of A. C. was partially conducted at the Future Resilient Systems and the Future Cities Laboratory at the Singapore-ETH Centre (SEC). The SEC was established as a collaboration between ETH Zurich and National Research Foundation (NRF) Singapore (FI 370074011, FI 370074016) under the auspices of the NRF's Campus for Research Excellence and Technological Enterprise (CREATE) program.

References

1. Achterberg, T.: SCIP: solving constraint integer programs. Math. Program. Comput. **1**(1), 1–41 (2009)
2. Audet, C., Dennis Jr., J.: Mesh adaptive direct search algorithms for constrained optimization. SIAM J. Optim. **17**(1), 188–217 (2004)
3. Audet, C., Kokkolaras, M., Le Digabel, S., Talgorn, B.: Order-based error for managing ensembles of surrogates in mesh adaptive direct search. J. Glob. Optim. **70**(3), 645–675 (2018)
4. Baudoui, V.: Optimisation robuste multiobjectifs par modèles de substitution. Ph.D. thesis, University of Toulouse Paul Sabatier (2012)
5. Björkman, M., Holmström, K.: Global optimization of costly nonconvex functions using radial basis functions. Optim. Eng. **1**(4), 373–397 (2000)
6. Bonami, P., Biegler, L., Conn, A., Cornuéjols, G., Grossmann, I., Laird, C., Lee, J., Lodi, A., Margot, F., Sawaya, N., Wächter, A.: An algorithmic framework for convex mixed integer nonlinear programs. Discrete Optim. **5**, 186–204 (2008)
7. Bussieck, M.R., Drud, A.S., Meeraus, A.: MINLPLib—a collection of test models for mixed-integer nonlinear programming. INFORMS J. Comput. **15**(1), 114–119 (2003)
8. Byrd, R.H., Nocedal, J., Waltz, R.A.: KNITRO: an integrated package for nonlinear optimization. In: Di Pillo, G., Roma, M. (eds.) Large-Scale Nonlinear Optimization, pp. 35–59. Springer, New York (2006)
9. Conn, A.R., Scheinberg, K., Toint, P.L.: Recent progress in unconstrained nonlinear optimization without derivatives. Math. Program. **79**(1–3), 397–414 (1997). https://doi.org/10.1007/BF02614326

10. Conn, A.R., Scheinberg, K., Vicente, L.N.: Introduction to Derivative-Free Optimization. MPS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, Philadelphia (2009)
11. Costa, A., Di Buccio, E., Melucci, M., Nannicini, G.: Efficient parameter estimation for information retrieval using black-box optimization. IEEE Trans. Knowl. Data Eng. **30**, 1240–1253 (2017)
12. Costa, A., Nannicini, G., Schroepfer, T., Wortmann, T.: Black-box optimization of lighting simulation in architectural design. In: Cardin, M.A., Krob, D., Chuen, L., Tan, Y., Wood, K. (eds.) Designing Smart Cities: Proceedings of the First Asia-Pacific Conference on Complex Systems Design & Management, CSD&M Asia 2014, pp. 27–39. Springer (2015)
13. D'Ambrosio, C., Nannicini, G., Sartor, G.: MILP models for the selection of a small set of well-distributed points. Oper. Res. Lett. **45**(1), 46–52 (2017)
14. Diaz, G.I., Fokour, A., Nannicini, G., Samulowitz, H.: An effective algorithm for hyperparameter optimization of neural networks. IBM J. Res. Dev. **61**(4/5), 9-1 (2017)
15. Dixon, L., Szego, G.: The global optimization problem: an introduction. In: Dixon, L., Szego, G. (eds.) Towards Global Optimization, pp. 1–15. North Holland, Amsterdam (1975)
16. Eriksson, D., Bindel, D., Shoemaker, C.: Surrogate optimization toolbox (pySOT) (2015). http://github.com/dme65/pySOT
17. Fuerle, F., Sienz, J.: Formulation of the Audze–Eglais uniform latin hypercube design of experiments for constrained design spaces. Adv. Eng. Softw. **42**(9), 680–689 (2011)
18. Gablonsky, J., Kelley, C.: A locally-biased form of the DIRECT algorithm. J. Glob. Optim. **21**(1), 27–37 (2001)
19. Gendreau, M., Potvin, J.Y. (eds.): Handbook of Metaheuristics, 2nd edn. Kluwer, Dordrecht (2010)
20. Glover, F., Kochenberger, G. (eds.): Handbook of Metaheuristics. Kluwer, Dordrecht (2003)
21. Gomes, C.P., Selman, B., Crato, N., Kautz, H.: Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. J. Autom. Reason. **24**(1–2), 67–100 (2000). https://doi.org/10.1023/A:1006314320276
22. Gutmann, H.M.: A radial basis function method for global optimization. J. Glob. Optim. **19**, 201–227 (2001). https://doi.org/10.1023/A:1011255519438
23. Hart, W.E., Laird, C., Watson, J.P., Woodruff, D.L.: Pyomo—optimization Modeling in Python. Springer Optimization and Its Applications, vol. 67. Springer, Berlin (2012)
24. Hart, W.E., Watson, J.P., Woodruff, D.L.: Pyomo: modeling and solving mathematical programs in Python. Math. Program. Comput. **3**(3), 219–260 (2011). https://doi.org/10.1007/s12532-011-0026-8
25. Hemker, T.: Derivative free surrogate optimization for mixed-integer nonlinear black-box problems in engineering. Master's thesis, Technischen Universität Darmstadt (2008)
26. Holmström, K.: An adaptive radial basis algorithm (ARBF) for expensive black-box global optimization. J. Glob. Optim. **41**(3), 447–464 (2008)
27. Holmström, K., Quttineh, N.H., Edvall, M.M.: An adaptive radial basis algorithm (ARBF) for expensive black-box mixed-integer constrained global optimization. Optim. Eng. **9**(4), 311–339 (2008)
28. Huyer, W., Neumaier, A.: SNOBFIT—stable noisy optimization by branch and fit. ACM Trans. Math. Softw. **35**(2), 1–25 (2008)
29. Ilievski, I., Akhtar, T., Feng, J., Shoemaker, C.A.: Efficient hyperparameter optimization of deep learning algorithms using deterministic RBF surrogates. In: Thirty-First AAAI Conference on Artificial Intelligence (2017)
30. Jakobsson, S., Patriksson, M., Rudholm, J., Wojciechowski, A.: A method for simulation based optimization using radial basis functions. Optim. Eng. **11**(4), 501–532 (2010)
31. Johnson, S.G.: The NLopt nonlinear-optimization package. http://ab-initio.mit.edu/nlopt
32. Jones, D., Perttunen, C., Stuckman, B.: Lipschitzian optimization without the Lipschitz constant. J. Optim. Theory Appl. **79**(1), 157–181 (1993)
33. Jones, D., Schonlau, M., Welch, W.: Efficient global optimization of expensive black-box functions. J. Glob. Optim. **13**(4), 455–492 (1998)
34. Kolda, T.G., Lewis, R.M., Torczon, V.J.: Optimization by direct search: new perspectives on some classical and modern methods. SIAM Rev. **45**(3), 385–482 (2003)
35. Le Digabel, S.: Algorithm 909: NOMAD: nonlinear optimization with the MADS algorithm. ACM Trans. Math. Softw. **37**(4), 44:1–44:15 (2011). https://doi.org/10.1145/1916461.1916468
36. MINLP Library 2. http://www.gamsworld.org/minlp/minlplib2/html/
37. Moré, J., Wild, S.M.: Benchmarking derivative-free optimization algorithms. SIAM J. Optim. **20**(1), 172–191 (2009)

38. Müller, J.: MISO: mixed-integer surrogate optimization framework. Optim. Eng. 1–27 (2015). https://doi.org/10.1007/s11081-015-9281-2
39. Müller, J., Paudel, R., Shoemaker, C.A., Woodbury, J., Wang, Y., Mahowald, N.: CH_4 parameter estimation in CLM4.5bgc using surrogate global optimization. Geosci. Model Dev. **8**(10), 3285–3310 (2015). https://doi.org/10.5194/gmd-8-3285-2015
40. Müller, J., Shoemaker, C.A.: Influence of ensemble surrogate models and sampling strategy on the solution quality of algorithms forcomputationally expensive black-box global optimization problems. J. Glob. Optim. **60**(2), 123–144 (2014). https://doi.org/10.1007/s10898-014-0184-0
41. Müller, J., Shoemaker, C.A., Piché, R.: SO-MI: a surrogate model algorithm for computationally expensive nonlinear mixed-integer black-box global optimization problems. Comput. Oper. Res. **40**(5), 1383–1400 (2013). https://doi.org/10.1016/j.cor.2012.08.022
42. Nelder, J.A., Mead, R.: A simplex method for function minimization. Comput. J. **7**(4), 308–313 (1965)
43. Neumaier, A.: Neumaier's collection of test problems for global optimization. http://www.mat.univie.ac.at/~neum/glopt/my_problems.html. Retrieved in May 2014
44. Powell, M.: Recent research at Cambridge on radial basis functions. In: Müller, M.W., Buhmann, M.D., Mache, D.H., Felten, M. (eds.) New Developments in Approximation Theory. International Series of Numerical Mathematics, vol. 132, pp. 215–232. Birkhauser Verlag, Basel (1999)
45. Powell, M.J.: The BOBYQA algorithm for bound constrained optimization without derivatives. Technical Report, Cambridge NA Report NA2009/06, University of Cambridge (2009)
46. Regis, R., Shoemaker, C.: Improved strategies for radial basis function methods for global optimization. J. Glob. Optim. **37**, 113–135 (2007). https://doi.org/10.1007/s10898-006-9040-1
47. Regis, R.G., Shoemaker, C.A.: A stochastic radial basis function method for the global optimization of expensive functions. INFORMS J. Comput. **19**(4), 497–509 (2007). https://doi.org/10.1287/ijoc.1060.0182
48. Regis, R.G., Shoemaker, C.A.: A quasi-multistart framework for global optimization of expensive functions using response surface models. J. Glob. Optim. **56**(4), 1719–1753 (2013)
49. Rios, L.M., Sahinidis, N.V.: Derivative-free optimization: a review of algorithms and comparison of software implementations. J. Glob. Optim. **56**(3), 1247–1293 (2013)
50. Schoen, F.: A wide class of test functions for global optimization. J. Glob. Optim. **3**(2), 133–137 (1993)
51. Törn, A., Žilinskas, A.: Global Optimization. Springer, Berlin (1987)
52. Wächter, A., Biegler, L.T.: On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. Math. Program. **106**(1), 25–57 (2006)
53. Wortmann, T., Costa, A., Nannicini, G., Schroepfer, T.: Advantages of surrogate models for architectural design optimization. Artif. Intell. Eng. Des. Anal. Manuf. **29**(4), 471–481 (2015)
54. Wortmann, T., Waibel, C., Nannicini, G., Evins, R., Schroepfer, T., Carmeliet, J.: Are genetic algorithms really the best choice for building energy optimization? In: Proceedings of the Symposium on Simulation for Architecture & Urban Design (SimAUD), pp. 51–58. SCS, Toronto, Canada (2017)