J Glob Optim (2014) 59:865–889
DOI 10.1007/s10898-013-0101-y

SO-I: a surrogate model algorithm for expensive nonlinear integer programming problems including global optimization applications

Juliane Müller · Christine A. Shoemaker · Robert Piché

Received: 4 October 2012 / Accepted: 12 August 2013 / Published online: 5 September 2013

Abstract This paper presents the surrogate model based algorithm SO-I for solving purely integer optimization problems that have computationally expensive black-box objective functions and that may have computationally expensive constraints. The algorithm was developed for solving global optimization problems, meaning that the relaxed optimization problems have many local optima. However, the method is also shown to perform well on many local optimization problems, and problems with linear objective functions. The performance of SO-I, a genetic algorithm, Nonsmooth Optimization by Mesh Adaptive Direct Search (NOMAD), SO-MI (Müller et al. in Comput Oper Res 40(5):1383–1400, 2013), variable neighborhood search, and a version of SO-I that only uses a local search has been compared on 17 test problems from the literature, and on eight realizations of two application problems. One application problem relates to hydropower generation, and the other one to throughput maximization. The numerical results show that SO-I finds good solutions most efficiently. Moreover, as opposed to SO-MI, SO-I is able to find feasible points by employing

Electronic supplementary material The online version of this article (doi:10.1007/s10898-013-0101-y) contains supplementary material, which is available to authorized users.

J. Müller (✉)
School of Civil and Environmental Engineering, Cornell University,
209 Hollister Hall, Ithaca, NY 14853-3501, USA
e-mail: juliane.mueller2901@gmail.com

C. A. Shoemaker
School of Civil and Environmental Engineering, Center of Applied Mathematics,
Cornell University, 210 Hollister Hall, Ithaca, NY 14853-3501, USA

C. A. Shoemaker
School of Operations Research and Information Engineering, Center of Applied Mathematics,
Cornell University, 210 Hollister Hall, Ithaca, NY 14853-3501, USA

R. Piché
Department of Mathematics, Tampere University of Technology,
P.O. Box 553, 33101 Tampere, Finland

a first optimization phase that aims at minimizing a constraint violation function. A feasible user-supplied point is not necessary.

Keywords Integer optimization · Derivative-free · Computationally expensive · Surrogate model · Response surface · Linear and nonlinear · Nonconvex · Radial basis functions · Multimodal · Global optimization

List of symbols

GA	Genetic algorithm
NOMAD	Nonsmooth Optimization by Mesh Adaptive Direct Search
RBF	Radial basis function
SEM	Standard error of means
SO-I	Surrogate Optimization-Integer
SO-MI	Surrogate Optimization-Mixed Integer
local-SO-I	local-Surrogate Optimization-Integer
VNS	Variable neighborhood search
$\mathbb{R}$	Real numbers
$\mathbb{Z}$	Integer numbers
$\mathbf{u}$	Discrete decision variable vector, see Eq. (1d)
$f(\cdot)$	Objective function, see Eq. (1a)
$c_j(\cdot)$	jth constraint function, $j = 1, \ldots, m$, see Eq. (1b)
m	Number of constraints
k	Problem dimension
j	Index for the constraints
i	Index for the variables
u_i^l, u_i^u	Lower and upper bounds for the ith variable, see Eq. (1c)
Ω_b	Box-constrained variable domain
Ω	Feasible variable domain
$\mathcal{S}$	Set of already evaluated points
n_0	Number of points in initial experimental design
$q(\cdot)$	Auxiliary function for minimizing constraint violation in phase 1, see Eq. (5)
$f_p(\cdot)$	Objective function value augmented with penalty term, see Eq. (6)
$f_{\max}$	Objective function value of the worst feasible point found so far, see Eq. (6)
p	Penalty factor, see Eq. (6)
$v(\cdot)$	Squared constraint violation function, see Eq. (7)
$\boldsymbol{\chi}_J$	Jth candidate point for next sample site, $J = 1, \ldots, t$
n	Number of already sampled points
$s(\cdot)$	Radial basis function interpolant
$V(\cdot)$	Weighted score, see Eq. (8)
ω_R, ω_D	Weights for response surface and distance criteria, respectively

1 Introduction

Management and engineering disciplines often require solving discrete optimization problems. Scheduling, inventory control, or structural optimization are just few examples. Also, applications arising from throughput maximization are often integer problems [50]. This

type of applications is encountered, for example, in production planning and facility layout [17,60], but also in the design of networks-on-chip routers [29]. Another purely integer optimization problem arises in hydropower energy generation where several turbines of different efficiencies are installed in a hydropower plant, and the question is how many turbines of the same kind should be in use to maximize the total generated power given a certain amount of water that can be released from the reservoir [41].

Many discrete optimization problems are NP-hard and therefore difficult to solve. Additionally, in many applications the objective and constraint functions are black-box functions, i.e. an analytical problem description is not available, and the objective and constraint function values are the output of a time consuming simulation. When optimizing such problems the number of objective and constraint function evaluations for finding accurate solutions must be as low as possible in order to keep computation times reasonable.

The most commonly used algorithms in the literature for solving discrete optimization problems comprise branch and bound methods (introduced by Land and Doig [38]), evolutionary algorithms as, for example, genetic algorithms [6,7,44], swarm algorithms such as particle swarm [33,34,39,51,59] or ant colony algorithms [13,42], tabu search [15,20,21,43], scatter search [19,22,36], and variable neighborhood search (VNS) [24,45].

VNS is a metaheuristic algorithm primarily developed for hard integer (global) optimization problems. The algorithm systematically explores the neighborhood of the current solution in order to find local optima. A perturbation phase enables the algorithm to escape from local optima and search globally for further improvements.

Evolutionary algorithms such as genetic algorithms [27] mimic the natural process of survival of the fittest. The genetic algorithm creates a set of initial candidate solutions (population) which may be randomly generated. In every generation the fitness of each individual is evaluated, and its fitness value is used for selecting the individuals for building the next generation. The chosen individuals are modified by mutation and crossover operations. Evolutionary algorithms are able to escape from local optima. In the literature genetic algorithms are often tailored to the specific problem under consideration. Coit and Smith [10], for example, use a genetic algorithm for solving optimal reliability design problems, and Haupt [25] solves a mixed-integer antenna design problem with a genetic algorithm. Attempts to adjust the crossover and mutation probabilities to improve the performance of the genetic algorithm have been examined by Hesser and Männer [26], Srinivas and Patnaik [61], Wu et al. [64], and Zhang et al. [66]. However, genetic algorithms may not be suitable for optimization problems with computationally expensive objective and constraint functions because a large number of function evaluations is in general needed to find good approximations of the global optimum.

Nonsmooth Optimization by Mesh Adaptive Direct Search (NOMAD) [2–4] is a derivative-free algorithm. NOMAD uses a mesh adaptive direct search algorithm designed for solving constrained black-box optimization problems. The mesh adaptive direct search is an extension of generalized pattern search algorithms for which superior convergence properties can be shown [1]. NOMAD is applicable for integer optimization problems, and by using the VNS [24,45] option, it is able to escape from local minima [5]. There are no extensive numerical studies of using NOMAD for solving constrained integer optimization problems.

In this paper a surrogate model approach is introduced. Surrogate models have proved successful when solving *continuous* global black-box optimization problems (see for example [23,28,30,47,54,56] and [63]). Although some surrogate model algorithms for mixed-integer problems have been developed [12,48,53], derivative-free algorithms for solving computationally expensive (global) integer optimization problems have barely been studied in the literature.

The remainder of this paper is organized as follows. In Sect. 2 the mathematical description of discrete optimization problems is given. The general concept of using surrogate models is briefly described in Sect. 3. Section 4 introduces the surrogate model based algorithm Surrogate Optimization-Integer (SO-I) for discrete global optimization problems that may have computationally expensive constraints. SO-I uses a first optimization phase for finding feasible points, i.e. an initial user-supplied feasible point is in contrast to SO-MI [48] not necessary. SO-I has been compared to a genetic algorithm, NOMAD 3.5, SO-MI [48], local-SO-I, which is a version of SO-I that only uses a local search, and VNS on 17 test problems from the literature, and two types of application problems (throughput maximization, and hydropower generation) that are briefly described in Sect. 5. The numerical results presented in Sect. 6 show that SO-I clearly outperforms algorithms that do not use a surrogate model. Because the focus is on computationally expensive functions, the computational effort is measured in the number of objective function evaluations rather than the CPU time needed by the algorithms. When dealing with computationally expensive objective functions that may take several hours for computing one function value, the algorithms' own computation times become insignificant. Section 7 concludes this paper. The Online Supplement contains specific information about the examined test problems and additional numerical results.

2 Constrained integer optimization problems

The optimization problem type considered in this paper is in general of the following form

$$\begin{aligned} & \min f(\mathbf{u}) && \text{(1a)}\\ \text{s.t.} \quad & c_j(\mathbf{u}) \le 0, \quad \forall j = 1, \ldots, m && \text{(1b)}\\ & -\infty < u_i^l \le u_i \le u_i^u < \infty, \quad \forall i = 1, \ldots, k && \text{(1c)}\\ & u_i \in \mathbb{Z}, \quad \forall i = 1, \ldots, k && \text{(1d)} \end{aligned}$$

where $f(\mathbf{u}) \in \mathbb{R}$ denotes the costly black-box objective function, $c_j(\mathbf{u}) \in \mathbb{R}$, $j = 1, \ldots, m$, are the m costly black-box constraints, and u_i^l and u_i^u denote the lower and upper bounds on the variable u_i, respectively (box constraints). Denote the box-constrained variable domain by Ω_b, and by Ω the feasible variable domain. Throughout this paper the variables $\mathbf{u}$ are assumed to be discrete and it is assumed that each variable can take on at least two different values. The set of already sampled points is denoted by $\mathcal{S}$. Moreover, it is assumed that objective and constraint function evaluations are the output of the same simulation code, i.e. whenever the objective function value is computed, also the constraint function values are obtained (if they exist).

A point $\mathbf{u}^* = (u_1^*, \ldots, u_k^*)^T \in \Omega$ is a local minimum of the function $f(\mathbf{u})$ in the domain Ω if $f(\mathbf{u}^*) \le f(\mathbf{u})$ for all $\mathbf{u} \in \Omega \cap N(\mathbf{u}^*)$, where the neighborhood $N(\mathbf{u}^*) \subset \Omega$ for integers is defined as $N(\mathbf{u}^*) = \{\tilde{\mathbf{u}} \in \Omega : \|\tilde{\mathbf{u}} - \mathbf{u}^*\|_1 \le 1\}$.

3 Surrogate models and radial basis functions

A surrogate model is an approximation of a simulation model [8]. While costly simulation models aim to represent the behavior of complex physical processes, surrogate models aim to cheaply compute approximations of the objective function values produced by the simulation models. The simulation model output $f(\mathbf{u})$ can be represented by $f(\mathbf{u}) = s(\mathbf{u}) + \epsilon(\mathbf{u})$, where $s(\mathbf{u})$ denotes the surrogate model output at point $\mathbf{u}$, and $\epsilon(\mathbf{u})$ is the difference between

the simulation and the surrogate model output. Different surrogate model types such as multivariate adaptive regression splines [16], regression polynomials [49], kriging [11,31], radial basis functions [14,23] as well as surrogate model ensembles [47] have emerged in the literature (see [30,58] for a review) and have been applied for solving continuous global optimization problems [18,32,37,40,54–56,63].

Denote in the following $\mathbf{u}_1, \ldots, \mathbf{u}_n$ the sample points where the objective function has been evaluated ($\mathcal{S} = \{\mathbf{u}_1, \ldots, \mathbf{u}_n\}$). In this paper radial basis function (RBF) models are used. An RBF interpolant can be represented as

$$s(\mathbf{u}) = \sum_{\iota=1}^{n} \lambda_\iota \phi(\|\mathbf{u} - \mathbf{u}_\iota\|_2) + \rho(\mathbf{u}) \tag{2}$$

where s denotes the response surface, ϕ denotes the radial basis function, $\rho(\mathbf{u})$ is a polynomial tail that depends on the choice of the radial basis function type, and n denotes the number of points where the costly objective and constraint functions have already been evaluated. Here, a cubic radial basis function ($\phi(r) = r^3$) with linear polynomial tail $\rho(\mathbf{u}) = \mathbf{b}^T\mathbf{u} + a$ has been used, where $\mathbf{b} = (b_1, \ldots, b_k)^T \in \mathbb{R}^k$, and $a \in \mathbb{R}$. The parameters λ_ι, a, and $b_1, \ldots, b_k$ are determined by solving a linear system of Eqs. [23]:

$$\begin{bmatrix} \boldsymbol{\Phi} & \mathbf{P} \\ \mathbf{P}^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda} \\ \mathbf{c} \end{bmatrix} = \begin{bmatrix} \mathbf{F} \\ \mathbf{0} \end{bmatrix}, \tag{3}$$

where $\Phi_{\iota_1\iota_2} = \phi(\|\mathbf{u}_{\iota_1} - \mathbf{u}_{\iota_2}\|_2)$, $\iota_1, \iota_2 = 1, \ldots, n$, $\|\cdot\|_2$ denotes the Euclidean norm,

$$\mathbf{P} = \begin{bmatrix} \mathbf{u}_1^T & 1 \\ \mathbf{u}_2^T & 1 \\ \vdots & \vdots \\ \mathbf{u}_n^T & 1 \end{bmatrix}, \quad \boldsymbol{\lambda} = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \\ a \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} f(\mathbf{u}_1) \\ f(\mathbf{u}_2) \\ \vdots \\ f(\mathbf{u}_n) \end{bmatrix}, \tag{4}$$

and where $\mathbf{0}$ is a matrix with all entries 0 of appropriate dimension. The linear system (3) has a unique solution if and only if $\text{rank}(\mathbf{P}) = k + 1$ [52].

4 SO-I: surrogate model algorithm for discrete global optimization problems

The algorithm for solving discrete optimization problems with costly objective and constraint functions starts by building an initial experimental design. A symmetric Latin hypercube design [65] with $n_0 = 2(k+1)$ points, where k denotes the problem dimension, is generated, and the values are rounded to the closest integers. The algorithm uses an RBF interpolant as defined in Eq. (2), and it must be ensured that the rank of matrix $\mathbf{P}$ in Eq. (4) is $k + 1$. Thus, the initial experimental design is regenerated until $\text{rank}(\mathbf{P}) = k + 1$.

The optimization process following the initial experimental design works iteratively and consists of two phases. The purpose of phase 1 is to find a first feasible point if there is none in the initial experimental design. The auxiliary function

$$q(\mathbf{u}) = \sum_{j=1}^{m} \max\{0, c_j(\mathbf{u})\}, \tag{5}$$

is iteratively minimized with respect to the box and integrality constraints defined in Eqs. (1c) and (1d) until a first feasible point has been found ($q(\mathbf{u}) = 0$ for a feasible point). If the

optimization problem has only box constraints or the initial experimental design contains at least one feasible point, optimization phase 1 is skipped.

In optimization phase 2, the penalized objective function

$$f_p(\mathbf{u}) = \begin{cases} f_{\max} + pv(\mathbf{u}) & \text{if } \mathbf{u} \text{ is infeasible} \\ f(\mathbf{u}) & \text{if } \mathbf{u} \text{ is feasible} \end{cases}, \tag{6}$$

is iteratively minimized subject to the box and integrality constraints in Eqs. (1c) and (1d). Here, p denotes the penalty factor, and $f_{\max}$ is the feasible point with the largest function value found so far (the worst feasible function value), and

$$v(\mathbf{u}) = \sum_{j=1}^{m} \max\left\{0, c_j(\mathbf{u})\right\}^2, \tag{7}$$

where it is assumed that $\mathbf{u} \in \Omega_b$, i.e. only points within the box-constrained domain are considered. Thus, if $v(\mathbf{u}) = 0$, the point $\mathbf{u}$ satisfies every constraint $c_j(\mathbf{u})$, $j = 1, \ldots, m$, in Eq. (1b). Note that the function values of infeasible points may change between iterations because $f_{\max}$ is being updated throughout the algorithm.

The penalty must be adjusted such that the function values at the infeasible points are higher than the feasible objective function values. Since the objective function is only defined at integer points (and sample points have therefore a distance of at least one unit to each other), there is no risk of creating jump discontinuities at the boundary of the feasible region by adding a penalty. The fact that the objective function is only evaluated at integer points does not influence the smoothness of the response surface. Adding a penalty for infeasible points works similarly to a barrier method and the response surface is likely to predict larger function values for points close to infeasible points. If no penalty was added, the surrogate model could predict better objective function values in the infeasible region, the search might be drawn into that region, and many expensive function evaluations might be done without improving the solution. To overcome numerical difficulties and to prevent the response surface from oscillating wildly, which may be caused by adding the penalty, function values that are larger than the median of all values of $f_p(\mathbf{u})$ are replaced by the median value (see also the Online Supplement for further explanation of the influence of the penalty factor).

Both optimization phases work iteratively and in each iteration one new point for doing the next expensive function evaluation is added to the set of already sampled points. A candidate point approach is applied to determine the new sample site. Candidates are generated by (a) uniformly selecting integer points in Ω_b and (b) perturbing the best point found so far (denoted by $\mathbf{u}_{\min}$). In optimization phase 1 the best point found so far is the one with minimal $q(\mathbf{u})$. In optimization phase 2 the best point found so far is the *feasible* point with minimal objective function value. Every variable of $\mathbf{u}_{\min}$ is perturbed with probability 0.5. If a variable is perturbed, the value 1, 2, or 3 is at random added or subtracted, where each perturbation range has the same probability of being selected. The perturbation ranges have been chosen with the incentive of creating candidate points in the neighborhood of $\mathbf{u}_{\min}$ in order to improve the local search. If any value of $\mathbf{u}_{\min}$ after perturbation exceeds the variable's upper or lower bound, the value is set to the value of the variable bound that is exceeded, i.e. $\boldsymbol{\chi}_J \in \Omega_b, \forall J = 1, \ldots, t$, where $\boldsymbol{\chi}_J$ denotes the Jth candidate point.

Given the sample data obtained so far, the parameters of the RBF model are computed by solving the system (3). In optimization phase 1 the values of $q(\mathbf{u})$ in Eq. (5) are used, and in optimization phase 2 the values of $f_p(\mathbf{u})$ in Eq. (6) are used in the right hand side $\mathbf{F}$ of Eq. (3) to compute the model parameters, where $\mathbf{u} \in \mathcal{S}$. The response surface is then used to predict the values of $q(\boldsymbol{\chi}_J), \forall J = 1, \ldots, t$, in optimization phase 1, and $f_p(\boldsymbol{\chi}_J), \forall J = 1, \ldots, t$, in

optimization phase 2 at every candidate point $\boldsymbol{\chi}_j$. Note that the response surface is cheap to evaluate, and this step is therefore computationally inexpensive. The candidate points' predicted function values are scaled to the interval [0, 1] so that low predicted values obtain a lower score, and high predicted values obtain a higher score (see [55, page 501] for details; response surface criterion, V_R). The distance of all candidate points to the set of already sampled points $\mathcal{S}$ is calculated. These distances are scaled to the interval [0, 1] so that high distances obtain a lower score, and low distances obtain a higher score (see [55, page 501] for details; distance criterion, V_D). Based on these two criteria the weighted score

$$V(\boldsymbol{\chi}_j) = \omega_R V_R(\boldsymbol{\chi}_j) + \omega_D V_D(\boldsymbol{\chi}_j), \quad j = 1, \ldots, t, \tag{8}$$

where $\omega_R + \omega_D = 1$, and $\omega_R \geq 0$ is the weight for the response surface criterion, and $\omega_D \geq 0$ is the weight for the distance criterion, is computed and the candidate point with the lowest score is chosen as the next sample site. Note that candidates that coincide with already sampled points obtain a bad score ($V = \infty$) and will therefore never be chosen as new sample sites. In case all candidates have the score ∞, new candidates are generated until at least one unsampled point has been found.

In optimization phase 1, the weights in Eq. (8) are fixed to $\omega_R = 0.9$ and $\omega_D = 0.1$. This emphasizes the local search in an attempt to find a feasible point quickly by giving an advantage to candidate points with low predicted values of $q(\mathbf{u})$.

In optimization phase 2, four different approaches have been examined for adjusting the weights in Eq. (8):

- *Version 1:* $\omega_R = 0.9$, $\omega_D = 0.1$ constant
- *Version 2:* At the nth function evaluation, $\omega_D = 1 - \log(n)/\log(N_{\max})$, $\omega_R = 1 - \omega_D$, where $N_{\max}$ denotes the maximum number of allowed function evaluations
- *Version 3:* The algorithm repeatedly cycles through a fixed pattern $\langle 0, 0.1, 0.2, \ldots, 0.9, 1.0\rangle$, i.e. the weights change in every iteration (e.g. in iteration 1 $\omega_D = 1$, $\omega_R = 0$; in iteration 2 $\omega_D = 0.9$, $\omega_R = 0.1$, etc.)
- *Version 4:* Adjust the weights as in Version 3 and use the point that minimizes the weighted score in Eq. (8) as new sample point.

The adjustment in Version 1 may be regarded as a more local search since preference is always given to candidate points that have a low predicted objective function value, and the distance criterion has only a low influence. Version 2 may be interpreted as a transition from a global search (initially candidate points that are far away from already sampled points are preferred) to a local search. The variable domain is explored during the first iterations in order to improve the global fit of the response surface, and to find promising regions where the global optimum might be. As the sampling continues, the search becomes more local and candidate points with low predicted objective function values obtain better scores, and therefore the promising regions of the variable domain are examined more thoroughly. Version 3 repeatedly transitions from a global to a local search in an attempt to escape from local optima. Version 4 attempts to find the actual optimum of the scoring function rather than choosing the best candidate among a limited number of points.

Numerical experiments showed that Versions 1 and 2 deliver in general better results than Versions 3 and 4, and that there are no major differences between the performance of the algorithms when using Version 1 or 2. For reasons of space considerations the results of these numerical experiments are not presented here. In the following, Version 1 will be used in the comparison with the other algorithms because it does not require the knowledge of the allowed number of function evaluations.

After the best candidate point has been chosen, the costly objective and constraint functions are evaluated and the response surface parameters are updated with the new data. This process (candidate point generation, score calculation, response surface updating) iterates. Note that for building the response surface the variables are assumed to be continuous to obtain a smooth surface, but the generated candidate and sample points only have discrete variable values. The specific steps of SO-I with Version 1 as sampling strategy are summarized in Algorithm 1.

Algorithm 1 ***SO-I: surrogate optimization-integer***

1. *Build an initial experimental design.*
2. *Compute the costly objective and constraint function values.*
3. *Optimization phase 1: minimize* $q(\mathbf{u})$ *in Eq.* (5)*. Iterate until a feasible point has been found or the stopping criterion has been met:*
   (a) *Find the point with minimal value* $q(\mathbf{u})$, $\mathbf{u} \in \mathcal{S}$, *and denote this point by* $(\mathbf{u}^q_{\min}, q_{\min})$.
   (b) *Compute the response surface parameters based on the points in* $\mathcal{S}$ *and their corresponding values* $q(\mathbf{u})$*, taking median rule into account.*
   (c) *Generate candidate points.*
   (d) *Compute the candidate point scores.*
   (e) *Discard candidates that coincide with already sampled points and regenerate new candidates until at least one unsampled point has been found.*
   (f) *Do the expensive simulation at the best candidate point.*
4. *Optimization phase 2: minimize* $f_p(\mathbf{u})$ *in Eq.* (6)*. Iterate until the stopping criterion has been met:*
   (a) *Find the feasible point with lowest objective function value and denote the pair by* $(\mathbf{u}^f_{\min}, f_{\min})$*. Find the feasible point with highest objective function value and denote the pair by* $(\mathbf{u}^f_{\max}, f_{\max})$.
   (b) *Adjust function values according to Eq.* (6) *with* $p = 100$*, taking median rule into account.*
   (c) *Compute the response surface parameters using the points* $\mathbf{u} \in \mathcal{S}$ *and their corresponding function values* $f_p(\mathbf{u})$.
   (d) *Generate candidate points.*
   (e) *Compute the candidate point scores.*
   (f) *Discard candidates that coincide with already sampled points and regenerate new candidates until at least one unsampled point has been found.*
   (g) *Do the expensive simulation at the best candidate point.*
5. *Return the best point found.*

Theorem 1 *Algorithm 1 converges to the global minimum.*

Proof The convergence of the algorithm follows from a simple counting argument. Let f be the real-valued function defined on $\Omega_b \subset \mathbb{Z}^k$, and denote the cardinality $|\Omega_b| = \kappa < \infty$. Let $f(\mathbf{u}^*) = \min_{\mathbf{u}\in\Omega} f(\mathbf{u}) > -\infty$ be the feasible global minimum of f, i.e. $f(\mathbf{u}) \geq f(\mathbf{u}^*)\ \forall \mathbf{u} \neq \mathbf{u}^*, \mathbf{u} \in \Omega$. Assume the candidate points for the next sample site are generated as described in Algorithm 1 (no point will be sampled more than once and every point in Ω_b may become a candidate point because some of the candidates are generated by uniformly selecting integer points from Ω_b). Denote $\mathbf{u}_1$ the first feasible point found. Define the sequence

of feasible random vectors $\{\mathbf{u}_n^*\}_{n\geq 1}$ as $\mathbf{u}_1^* = \mathbf{u}_1$, $\mathbf{u}_{n+1}^* = \mathbf{u}_{n+1}$ if $f(\mathbf{u}_{n+1}) < f(\mathbf{u}_n^*)$ and $c_j(\mathbf{u}_{n+1}) \leq 0\,\forall j$, and $\mathbf{u}_{n+1}^* = \mathbf{u}_n^*$ otherwise. Then, $\mathbf{u}_n^* \to \mathbf{u}^*$ as $n \to \kappa - 1$. Thus, as the algorithm proceeds, the probability that the next chosen point will be the global optimum goes in the limit to 1. □

The proposed algorithm SO-I differs from the algorithm SO-MI [48] with respect to the following aspects:

- For constrained problems SO-MI requires the user to supply a feasible solution. SO-I, on the other hand, contains a routine (Optimization Phase 1) to search for a feasible solution.
- The penalty for constraint violations is treated in SO-MI differently in order to avoid jump discontinuities when reaching the boundary of the feasible domain. This step is not necessary in SO-I due to the discreteness of the variable domain.
- Candidate points are generated in SO-MI with a different perturbation probability and with different perturbation ranges [48, Section 4.3].
- The adjustment of the weights for the scoring criteria in SO-I is such that the response surface criterion has the weight $\omega_R = 0.9$ in every iteration, whereas in SO-MI the weights cycle through a predefined pattern.
- In SO-MI, four points are chosen for doing expensive evaluations in every iteration according to the four candidate point creation schemes (one point from each group is selected). In SO-I only the one best candidate point is selected for doing the expensive function evaluation.

5 Test setup and test problems

5.1 Test setup

Algorithms for solving discrete global black-box optimization problems with computationally expensive objective functions have not widely been studied in the literature. Therefore, SO-I is in the following compared to a genetic algorithm, NOMAD, SO-MI, local-SO-I (a version of SO-I that uses candidate points that are only generated by perturbing the best point found so far), and a VNS algorithm. Note, however, that SO-MI does not have a routine for finding feasible points of constrained problems. Therefore, SO-I was used until the first feasible point has been encountered, and thereafter SO-MI has been applied. Thus, the used SO-MI implementation is not exactly the original implementation proposed in [48]. Local-SO-I has been included in the experiments in order to show the significance of candidate points that are uniformly selected from the variable domain for the solution quality. All algorithms have been tested with Matlab 2011b.

For every algorithm, 30 trials have been made for the same test problem, i.e. 30 different initial symmetric Latin hypercube designs were used for SO-I, SO-MI, and local-SO-I, 30 different initial starting points were used for NOMAD and VNS, and 30 different initial populations were used with GA. The algorithms were stopped after 400 function evaluations because for expensive black-box problems this is already a rather large number. A time limit has not been defined for the algorithms because the performance of the algorithms is measured with respect to the solution quality after an equal number of function evaluations. Assuming that one objective function evaluation requires a time consuming simulation, the algorithms' own computation times become insignificant.

The initial experimental design of SO-I, SO-MI, and local-SO-I is generated by a symmetric Latin hypercube design where each trial of a specific problem is initialized with a different

random number seed. Note that for the same trial of a given problem all three algorithms use the same initial design in order to obtain a conclusive comparison.

A branch and bound algorithm was not applicable for many constrained test problems because the optimization in the tree nodes for finding lower bounds for the objective function rapidly increases the number of function evaluations, and hence the computational budget was in many cases used up before a first feasible point had been found. Furthermore, the application problems do not allow non-integer variable values. Thus, optimizing the relaxed subproblems (some variables are assumed to be continuous) in the tree nodes failed and a solution could not be obtained.

For the genetic algorithm, Matlab's implementation (function ga) with default parameters has been used, which is able to solve discrete problems (Matlab 2011b and newer versions). The same point that is the first point in the initial experimental design of SO-I for a respective trial is given as *partial* initial population for ga. Matlab's default function for creating the initial population is used for generating the remaining points. The default population size is 20.

NOMAD version 3.5 has been used. Note, that NOMAD is included in the *OPTI Toolbox*,[1] which is a Matlab toolbox for optimization and simplifies the use of NOMAD as compared to its C++ implementation. The constraints have been treated with the progressive barrier approach (setting PB), and the VNS has been used in order to enable the algorithm to escape from local optima (setting VNS 0.75 as suggested in the user manual). The algorithm starts for a given trial of a certain problem from the same point that is the first point in the initial experimental design of SO-I for the respective trial.

For VNS the implementation in the MEIGO software[2] has been used. The implementation is designed for box-constrained integer problems only and does not have a routine for finding feasible points. Hence, two versions of VNS have been tried. Version 1 (VNS-i) minimizes similarly to SO-I at first a constraint violation function if the given initial point is infeasible. Version 2 (VNS-ii) uses SO-I to determine the first feasible point for constrained problems which in turn is used as starting point of MEIGO's VNS implementation.

5.2 Test problems

The performance of the algorithms has been compared on 17 test problems from the literature, and eight realizations of two application problems arising in throughput maximization and hydropower generation maximization. Algorithms for purely integer black-box optimization problems have not been studied in depth in the literature, and thus finding a representative and widely used test bench of problems is not possible. Most test problems have been derived from continuous global optimization test problems by imposing integrality constraints on every variable. Although the 17 test problems have analytical descriptions (given in the Online Supplement), the problems are treated as black-boxes, i.e. the mathematical problem structure cannot be exploited by any of the algorithms. The test problems are used to examine the algorithms' ability to solve problems with different characteristics such as multimodality, linear and nonlinear constraints and objective functions, or binary variables.

Table 1 summarizes the characteristics of all test problems. Note that all variables are assumed to be discrete. The column "Notes" gives references about the problem's origin or on other characteristics that are considered "special". The last column indicates if the relaxed problem (all variables continuous) has several local, global and/or stationary points (denoted

[1] http://www.i2c2.aut.ac.nz/Wiki/OPTI/.

[2] http://www.iim.csic.es/~gingproc/meigo.html.

Table 1 Test problems (a) http://www.aridolan.com/ga/gaa/MultiVarMin.html

ID	k	Domain	Notes	Characteristics
1[c]	2	$[13, 100] \times [0, 100]$	[35]	MM, 2 NLC
2	5	$[-100, 100]^5$	(a)	MM
3[c]	5	$[0, 10]^3 \times [0, 1]^2$	[9]	lin.obj., 2 NLC, 3 LC
4[c]	5	$[78, 102] \times [33, 45] \times [27, 45]^3$	[35]	UM, 6 NLC
5[c]	7	$[-10, 10]^7$	[35]	UM, 4 NLC
6[c]	13	$[0, 1]^{10} \times [0, 100]^3$	[35]	MM, 9 LC
7	10	$[3, 9]^{10}$	[9]	UM
8[c]	16	$[0, 1]^{16}$	[9]	MM, 7 NLC
9	12	$[-1, 3]^{12}$	Rastrigin [62]	MM
10	30	$[-1, 3]^{30}$	Rastrigin [62]	MM
11[c]	25	$[0, 10]^{25}$	[35]	MM, 1 NLC, 1 LC
12	20	$[0, 1]^{20}$	Schoen [57]	MM
13	10	$[3, 99]^{10}$	[9]	MM
14[c]	13	$[0, 100]^{13}$	[35]	MM, 9 LC
15	12	$[-10, 30]^{12}$	Rastrigin [62]	MM
16	8	$[-10, 10]^8$	convex	UM
17[c]	5	$[0, 10]^3 \times [0, 1]^2$	linear	lin.obj., 3 LC
18[c]	5	$[1, 20]^5$	max. throughput	BB objective, 2 LC
19[c]	23	$[1, 20]^{23}$	max. throughput	BB objective, 2 LC
20a[c]	5	$[0, 10]^5$	max. hydropower	BB objective, 1 NLC
20b[c]	5	$[0, 10]^5$	max. hydropower	BB objective, 1 NLC
20c[c]	5	$[0, 10]^5$	max. hydropower	BB objective, 1 NLC
21a[c]	5	$[0, 10]^5$	max. hydropower	BB objective, 2 NLC
21b[c]	5	$[0, 10]^5$	max. hydropower	BB objective, 2 NLC
21c[c]	5	$[0, 10]^5$	max. hydropower	BB objective, 2 NLC

[c] denotes constrained problems; see the Online Supplement for further information

by MM), or if the relaxed problem has only one local optimum and no other stationary points (denoted by UM). Furthermore, it is indicated how many linear constraints (LC) and nonlinear constraints (NLC) each problem has, and if the objective function is linear (lin.obj.) or a black-box (BB objective).

5.2.1 Throughput maximization application

Two realizations of an application problem about throughput maximization [50] have been used to compare the algorithms. The problem of throughput maximization given restrictions on the total buffer size B and service rate R is encountered in different application areas such as production planning and facility layout [17,60]. The space in the production facility is limited and practical restrictions enforce limits on the total service rate at each station.

Throughput maximization is also an important topic in data networks in computer science where the goal is to maximize the throughput subject to given restrictions on the buffer size.

Especially for networks-on-chip router design the on-chip network should be implemented with very little area overhead in order to minimize implementation costs. The input buffers of on-chip routers take in general a significant portion of the silicon area of networks-on-chips, and thus their size should be as small as possible [29].

Here, the throughput maximization of a flow line with h stations is considered. The buffer storage in front of station 1 is infinite, whereas the buffer storages $b_2, b_3, \ldots, b_h$ in front of stations $2, 3, \ldots, h$ are finite. Every station i has a single server with service rate r_i, $i = 1, \ldots, h$. The service time required by every job is known. If the buffer at station i is full, then station $i-1$ is blocked because no job finished at station $i-1$ can be transferred to the buffer at station i. The total buffer space and the total service rate is limited, i.e.

$$\sum_{i=2}^{h} b_i \leq B \quad \text{(buffer space restriction)}, \tag{9a}$$

$$\sum_{i=1}^{h} r_i \leq R \quad \text{(service rate restriction)}, \tag{9b}$$

where B and R are known constants. The goal is to find a buffer allocation and service rates such that the throughput (average output of the flow line per unit time) is maximized.

Two problem realizations have been considered. First, a small scale problem with $h = 3$ stations has been used. For this problem $B = R = 20$. There are five variables $(b_2, b_3, r_1, r_2, r_3)$ and their lower and upper bounds have been set to 1 and 20, respectively. Secondly, a flow line with $h = 12$ stations has been considered where $R = B = 80$. For this problem the lower and upper bounds on the variable values have also been set to 1 and 20, respectively, and there are 23 variables $(b_2, \ldots, b_{12}, r_1, \ldots, r_{12})$. Altogether, 2,050 jobs have to be processed for each problem realization. To calculate the objective function value, the processing of all jobs has to be simulated. The system is initially empty and then 2,000 jobs are released. The time T for releasing the 50 last jobs is recorded, and the throughput is calculated as $50/T$.

5.2.2 Hydropower generation maximization application

Six realizations of a hydropower generation application problem have been considered. The goal is to maximize the power output over one day for hydropower plants with multiple types of generating units. Electricity produced by hydropower plants is an important source of renewable energy around the world. Large hydropower facilities, like the Three Gorges Project in China, are designed with different generator types (denoted by u_i, $i = 1, \ldots, k$) because some generators are more efficient (in terms of power generated/volume of discharged water) with high water heads and others are more efficient with lower heads. The water head is the height of the water in the reservoir relative to its height after discharge and thus determines the kinetic energy.

Hydropower planning is done for different time periods. For longer time periods (e.g. one or more weeks), stochastic analysis is more important since the weather in the future affects the inflow. For planning over a shorter period like one day, stochastic factors are less important because Q_j, the total amount of water to be released from reservoir j, over one day and the water head is known at the beginning of the day.

In this application problem a short term analysis of how to distribute Q_j units of water over one time period, e.g. a day, among all different generator units is done. This problem is inspired by current research in the Three Gorges Project-Gezhouba system on how to

allocate water over a year long period to determine what the Q_j for each day should be [41]. For computational reasons the allocation of water among different generating units cannot be considered directly in an annual analysis. The problem examined here is a subproblem of the annual hydroelectric power generation problem to determine the water allocation among generating units of different types. The current Three Gorges Project analysis solves this subproblem associated with the daily allocation of water among the generator units by exhaustive search on the integer values of u_i, which is much less efficient than the SO-I method proposed here. Because the Three Gorges Dam is such a large power source in the power grid, the focus of this study is to determine the maximum power output possible, assuming other capacities of elements of the power grid will be adjusted by long term planning after knowing the maximum power output of the Three Gorges system. Thus, there are no maximum hourly demand constraints incorporated in the optimization problem. Of course, if the problem is changed to examine optimization under additional demand constraints, it is expected that SO-I will perform similarly or even better since the constraints further restrict the parameter space.

The general problem description follows. Given are five types of hydropower generating units $u_1, \ldots, u_5$ that have minimum and maximum capacities of the water $q_i, i = 1, \ldots, 5$, that runs through the respective generator type. All identical generator units of type i will receive the same amount of water q_i. The goal is to determine how many identical units of each generator type should be used in order to maximize the total power generated while not exceeding the maximum amount of water Q_j that may be released from reservoir j. For each variable set $u_1, \ldots, u_5$ the amount of water q_i, $i = 1, \ldots, 5$, that goes through the generating unit of type i must be adjusted such that the generated power is maximized while not exceeding Q_j.

Considered are two groups of hydropower test problems related to water distribution among generator types. The first problem group has only one reservoir and one hydropower plant with five different generator types. The second group has two reservoirs and two hydropower plants, one upstream with two different generator types that are more efficient with higher water heads, and one further downstream with three different generator types that are more efficient with lower water heads. For both groups three different limits on the maximal available water Q_j were considered, representing the different water heads at different times of the year.

6 Numerical results

In the following the abbreviation GA stands for genetic algorithm, VNS-i and VNS-ii denotes the VNS algorithms, SO-I is the new surrogate model based algorithm for integer problems, SO-MI is the (slightly altered) mixed-integer algorithm introduced in [48], and local-SO-I denotes the integer surrogate model algorithm that generates candidate points by only perturbing the best point found so far.

Tables 2, 3 and 4 show for all problems the average objective function value after 400 function evaluations together with the standard errors of the means (column "mean (*SEM*)"). These numbers are computed based on the successful trials (feasible points have been found within 400 evaluations). Thus, if, for example, an algorithm failed to find a feasible solution for two out of 30 trials, the values are computed based on the remaining 28 successful trials. The number of unsuccessful trials of each algorithm is reported in the column "#NF". Some algorithms had difficulties finding feasible solutions for constrained test problems. GA failed to find feasible solutions for the fewest trials, followed by SO-I (and SO-MI and

Table 2 Constrained problems

ID	Algorithm	#NF	Mean *(SEM)*
1^c	GA	2	**−3,971.00** (*0.00*)
	SO-I	0	**−3,971.00** (*0.00*)
	local-SO-I	19	**−3,971.00** (*0.00*)
	SO-MI	0	**−3,971.00** (*0.00*)
	NOMAD	30	–
	VNS-i	14	**−3,971.00** (*0.00*)
	VNS-ii	0	**−3,971.00** (*0.00*)
3^c	GA	0	0.72 (*0.14*)
	SO-I	0	**0.00** (*0.00*)
	local-SO-I	0	**0.00** (*0.00*)
	SO-MI	0	**0.00** (*0.00*)
	NOMAD	0	**0.00** (*0.00*)
	VNS-i	10	**0.00** (*0.00*)
	VNS-ii	0	0.03 (*0.03*)
4^c	GA	0	−30,073.77 (*43.30*)
	SO-I	0	**−30,303.66** (*31.17*)
	local-SO-I	0	−29,069.70 (*106.61*)
	SO-MI	0	−30,075.73 (*53.15*)
	NOMAD	0	−30,192.67 (*35.29*)
	VNS-i	0	−29,574.12 (*92.80*)
	VNS-ii	0	−29,486.62 (*68.83*)
5^c	GA	0	896.53 (*29.21*)
	SO-I	0	**771.40** (*14.97*)
	local-SO-I	0	997.10 (*246.89*)
	SO-MI	0	812.17 (*12.46*)
	NOMAD	0	1,770.50 (*462.58*)
	VNS-i	4	8,906.35 (*6,161.61*)
	VNS-ii	0	2,097.17 (*1,367.02*)
6^c	GA	0	−6.07 (*0.59*)
	SO-I	0	**−14.83** (*0.10*)
	local-SO-I	0	−12.00 (*0.00*)
	SO-MI	0	−12.00 (*0.32*)
	NOMAD	0	−5.97 (*0.03*)
	VNS-i	30	–
	VNS-ii	0	−14.37 (*0.24*)
8^c	GA	8	17.73 (*0.86*)
	SO-I	14	14.00 (*0.68*)
	local-SO-I	8	20.96 (*3.11*)
	SO-MI	14	13.50 (*0.50*)
	NOMAD	22	**13.00** (*0.00*)
	VNS-i	4	13.73 (*0.44*)
	VNS-ii	14	**13.00** (*0.00*)

Table 2 continued

ID	Algorithm	#NF	Mean (*SEM*)
11[c]	GA	0	−0.22 (*0.01*)
	SO-I	0	−0.21 (*0.01*)
	local-SO-I	0	**−0.24** (*0.01*)
	SO-MI	0	−0.17 (*0.00*)
	NOMAD	0	−0.17 (*0.01*)
	VNS-i	15	−0.19 (*0.02*)
	VNS-ii	0	**−0.24** (*0.01*)
14[c]	GA	1	−40,105.07 (*3,175.53*)
	SO-I	0	−40,687.10 (*3,145.90*)
	local-SO-I	0	−42,185.67 (*1,966.68*)
	SO-MI	0	**−50,024.17** (*36.40*)
	NOMAD	0	−48,363.03 (*1,197.02*)
	VNS-i	30	–
	VNS-ii	0	−35,687.03 (*3,252.89*)
17[c]	GA	0	−7.10 (*0.16*)
	SO-I	0	**−8.00** (*0.00*)
	local-SO-I	0	−6.88 (*0.21*)
	SO-MI	0	**−8.00** (*0.00*)
	NOMAD	8	**−8.00** (*0.00*)
	VNS-i	16	−7.75 (*0.11*)
	VNS-ii	0	−7.93 (*0.04*)

Mean objective function values and standard errors of means over 30 trials after 400 function evaluations. Best result of all algorithms is marked in bold font. All problems are minimization problems.
[c] denotes constrained problems

Table 3 Unconstrained problems

ID	Algorithm	#NF	Mean (*SEM*)
2	GA	0	−427.91 (*7.38*)
	SO-I	0	−437.16 (*12.09*)
	local-SO-I	0	−348.42 (*20.54*)
	SO-MI	0	−459.56 (*8.52*)
	NOMAD	0	**−485.73** (*9.18*)
	VNS-i/VNS-ii	0	−308.57 (*3.86*)
7	GA	0	**−43.13** (*0.00*)
	SO-I	0	**−43.13** (*0.00*)
	local-SO-I	0	**−43.13** (*0.00*)
	SO-MI	0	**−43.13** (*0.00*)
	NOMAD	0	**−43.13** (*0.00*)
	VNS-i/VNS-ii	0	**−43.13** (*0.00*)
9	GA	0	−10.87 (*0.19*)
	SO-I	0	**−12.00** (*0.00*)
	local-SO-I	0	−10.03 (*0.77*)
	SO-MI	0	**−12.00** (*0.00*)
	NOMAD	0	**−12.00** (*0.00*)
	VNS-i/VNS-ii	0	**−12.00** (*0.00*)
10	GA	0	−17.17 (*0.66*)
	SO-I	0	−27.23 (*0.16*)

Table 3 continued

ID	Algorithm	#NF	Mean (*SEM*)
	local-SO-I	0	−27.43 (*0.16*)
	SO-MI	0	**−30.00** (*0.00*)
	NOMAD	0	−29.17 (*0.41*)
	VNS-i/VNS-ii	0	−26.63 (*0.86*)
12	GA	0	−12.23 (*0.19*)
	SO-I	0	**−13.36** (*0.00*)
	local-SO-I	0	**−13.36** (*0.00*)
	SO-MI	0	**−13.36** (*0.00*)
	NOMAD	0	−13.31 (*0.02*)
	VNS-i/VNS-ii	0	**−13.36** (*0.00*)
13	GA	0	−9,289.87 (*81.24*)
	SO-I	0	**−9,591.72** (*0.00*)
	local-SO-I	0	**−9,591.72** (*0.00*)
	SO-MI	0	**−9,591.72** (*0.00*)
	NOMAD	0	**−9,591.72** (*0.00*)
	VNS-i/VNS-ii	0	−5,448.97 (*358.19*)
15	GA	0	33.83 (*4.52*)
	SO-I	0	**−12.00** (*0.00*)
	local-SO-I	0	**−12.00** (*0.00*)
	SO-MI	0	**−12.00** (*0.00*)
	NOMAD	0	−10.67 (*1.33*)
	VNS-i/VNS-ii	0	16.47 (*22.67*)
16	GA	0	2.72 (*0.95*)
	SO-I	0	**0.00** (*0.00*)
	local-SO-I	0	**0.00** (*0.00*)
	SO-MI	0	**0.00** (*0.00*)
	NOMAD	0	**0.00** (*0.00*)
	VNS-i/VNS-ii	0	**0.00** (*0.00*)

Mean objective function values and standard errors of means over 30 trials after 400 function evaluations. Best result of all algorithms is marked in bold font. All problems are minimization problems. VNS-i and VNS-ii are the same for unconstrained problems

VNS-ii),[3] local-SO-I, NOMAD, and VNS-i. Noticeable about the results is that GA failed to find feasible solutions for a lower total number of trials, but for a larger number of different problems than SO-I. The tables also show that VNS-i failed to find feasible solutions for many more trials than VNS-ii.

The problems have been divided into constrained (Table 2), unconstrained (Table 3), and application (Table 4) problems. The column "ID" contains the problem identification number as defined in Table 1 and the superscript "c" at that number indicates that the problem has additional constraints besides the variables' upper and lower bounds. The best solution for every problem is marked in bold font, and the SEM value is given in italic font. Table entries marked by a dash denote that the respective algorithm was not able to find a feasible solution within 400 function evaluations in any trial.

[3] Note that SO-I is used in order to find a feasible solution from which both algorithms could start. Thus, the number of failed trials for SO-I, SO-MI, and VNS-ii are identical.

Table 4 Application problems

ID	Algorithm	#NF	Mean (*SEM*)
18[c]	GA	0	8.24 (*0.17*)
	SO-I	0	**9.12** (*0.12*)
	local-SO-I	0	6.57 (*0.40*)
	SO-MI	0	7.46 (*0.49*)
	NOMAD	3	7.66 (*0.26*)
	VNS-i	11	6.14 (*0.69*)
	VNS-ii	0	7.54 (*0.31*)
19[c]	GA	0	3.10 (*0.17*)
	SO-I	0	3.15 (*0.20*)
	local-SO-I	0	2.07 (*0.22*)
	SO-MI	0	**3.82** (*0.13*)
	NOMAD	0	0.89 (*0.06*)
	VNS-i	26	1.74 (*0.39*)
	VNS-ii	0	2.52 (*0.20*)
20a[c]	GA	0	735.34 (*6.75*)
	SO-I	0	**758.25** (*0.00*)
	local-SO-I	0	681.38 (*10.25*)
	SO-MI	0	754.38 (*0.88*)
	NOMAD	0	744.00 (*0.96*)
	VNS-i	0	753.83 (*1.54*)
	VNS-ii	0	755.04 (*1.12*)
20b[c]	GA	0	2,008.83 (*4.68*)
	SO-I	0	**2,020.67** (*1.33*)
	local-SO-I	0	1,835.14 (*24.94*)
	SO-MI	0	2,015.46 (*1.51*)
	NOMAD	0	2,003.83 (*5.01*)
	VNS-i	0	2,010.83 (*3.81*)
	VNS-ii	0	2,006.46 (*6.91*)
20c[c]	GA	0	4,108.84 (*4.49*)
	SO-I	0	4,114.63 (*2.50*)
	local-SO-I	0	3,890.61 (*20.74*)
	SO-MI	0	4,117.98 (*2.58*)
	NOMAD	0	**4,125.75** (*12.06*)
	VNS-i	0	4,075.42 (*10.44*)
	VNS-ii	0	4,099.17 (*6.69*)
21a[c]	GA	0	1,560.36 (*25.96*)
	SO-I	0	**1,677.17** (*2.10*)
	local-SO-I	0	1,443.12 (*46.42*)
	SO-MI	0	1,657.08 (*3.75*)
	NOMAD	0	1,626.18 (*19.64*)
	VNS-i	4	1,653.65 (*13.92*)
	VNS-ii	0	1,671.50 (*2.92*)

Table 4 continued

ID	Algorithm	#NF	Mean (*SEM*)
21b[c]	GA	0	4,016.17 (*23.14*)
	SO-I	0	**4,097.40** (*11.86*)
	local-SO-I	0	3,668.60 (*50.60*)
	SO-MI	0	4,095.50 (*6.96*)
	NOMAD	0	3,899.40 (*25.57*)
	VNS-i	2	4,000.93 (*28.66*)
	VNS-ii	0	4,070.83 (*16.29*)
21c[c]	GA	0	8,220.67 (*22.22*)
	SO-I	0	**8,299.00** (*12.84*)
	local-SO-I	0	7,550.17 (*73.20*)
	SO-MI	0	8,253.17 (*9.62*)
	NOMAD	0	8,122.33 (*32.05*)
	VNS-i	0	8,055.83 (*49.83*)
	VNS-ii	0	7,996.33 (*64.34*)

Mean objective function values and standard errors of means over 30 trials after 400 function evaluations. Best result of all algorithms is marked in bold font. All problems are maximization problems.
[c] denotes constrained problems

The numerical results show that SO-I attains the best mean objective function values for six out of nine constrained problems and for six out of eight unconstrained and application problems, respectively. Significance test for differences in means (see see Tables 1–3 in the Online Supplement) show that SO-I performs significantly better than all other algorithms especially for the application problems. For the constrained problems SO-I is outperformed only for test problems 11 and 14. For the unconstrained problems SO-I is outperformed only for problems 2 and 10. For the remaining test problems SO-I performs either significantly better than all other algorithms with a significance level 0.05, or the differences in means are not significant.

Data and performance profiles [46][4] are used in the following to illustrate the numerical results. The profiles are based on the feasible objective function value averaged over successful trials. The following summary of data and performance profiles is adapted from [46]. Denote $\mathcal{P}$ the set of problems (here the 25 test problems) and $\mathcal{A}$ the set of algorithms (GA, SO-I, local-SO-I, SO-MI, NOMAD, VNS-i, VNS-ii) in the comparison. Let $l_{\gamma,a}$, where $\gamma \in \mathcal{P}$ and $a \in \mathcal{A}$, be the used performance measure. Then the performance ratio is defined as

$$r_{\gamma,a} = \frac{l_{\gamma,a}}{\min\left\{l_{\gamma,a} : a \in \mathcal{A}\right\}}. \tag{10}$$

The performance profile of algorithm $a \in \mathcal{A}$ is then the fraction of problems where the performance ratio is at most α:

$$\rho_a(\alpha) = \frac{1}{|\mathcal{P}|}\text{size}\left\{\gamma \in \mathcal{P} : r_{\gamma,a} \leq \alpha\right\}. \tag{11}$$

Here $|\mathcal{P}|$ denotes the cardinality of $\mathcal{P}$. High values for $\rho_a(\alpha)$ are better. The performance profile reflects how well an algorithm performs relative to the other algorithms. Data profiles illustrate the percentage of problems solved for a given tolerance τ within a given number of function evaluations. If $l_{\gamma,a}$ denotes the number of function evaluations needed to satisfy a convergence test with tolerance τ, then the percentage of problems that can be solved within κ function evaluations is defined as

[4] The Matlab codes provided on http://www.mcs.anl.gov/~more/dfo/ have been used for creating the plots.

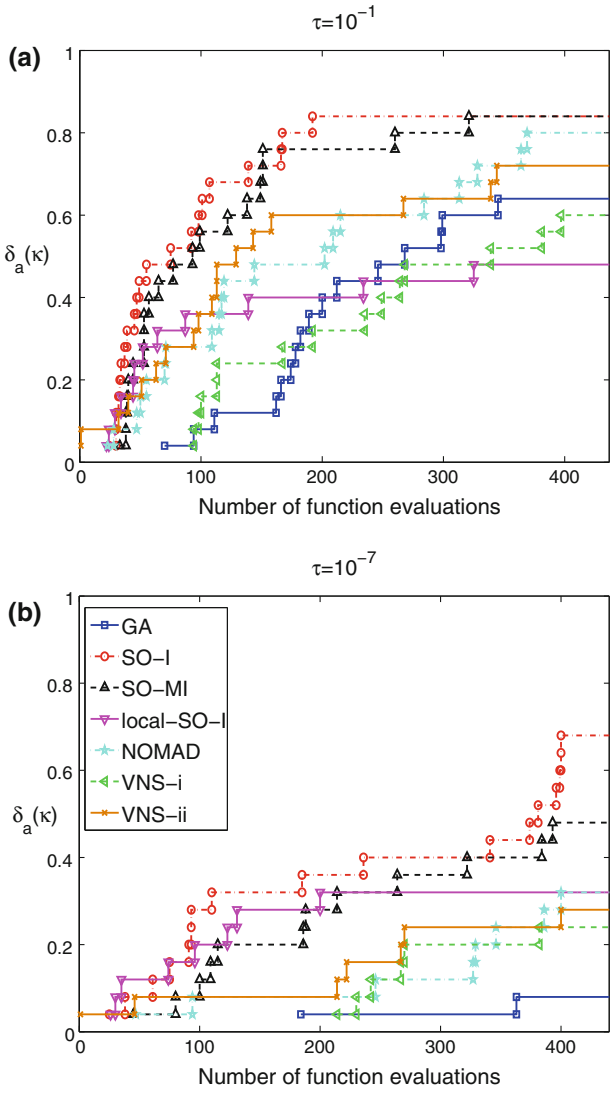

Fig. 1 Data profiles: percentage of problems solved vesus number of function evaluations for accuracy levels $\tau = 10^{-1}$ and $\tau = 10^{-7}$. Both figures share the same legend. See Eq. (12) for $\delta_a(\kappa)$

$$\delta_a(\kappa) = \frac{1}{|\mathcal{P}|}\text{size}\left\{\gamma \in \mathcal{P} : l_{\gamma,a} \leq \kappa\right\}. \tag{12}$$

Note that performance profiles compare the different algorithms whereas data profiles show the raw data.

The data profiles in Fig. 1 show the percentage of problems that could be solved by each algorithm within a given number of function evaluations for accuracy levels $\tau = 10^{-1}$ and $\tau = 10^{-7}$. Figure 1a shows, for example, that SO-I is able to solve more than 60 % of the problems within 100 evaluations with an accuracy level of $\tau = 10^{-1}$ whereas NOMAD

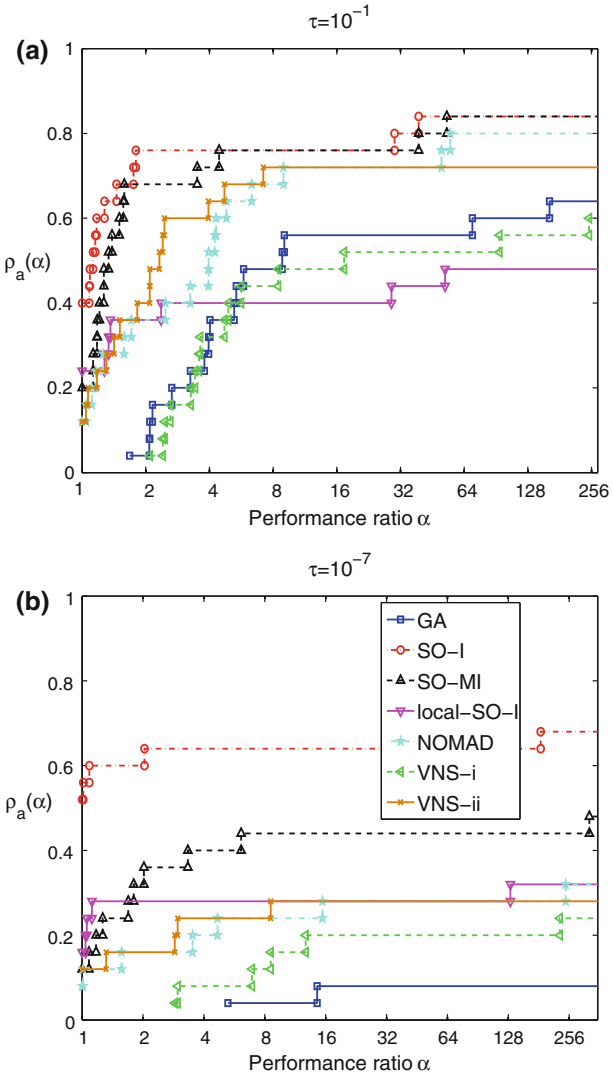

Fig. 2 Performance profiles with logarithmic scale for accuracy levels $\tau = 10^{-1}$ and $\tau = 10^{-7}$. Both figures share the same legend. See Eq. (11) for $\rho_a(\alpha)$

solves only about 28 % of the problems. Similarly, Fig. 1b shows that SO-I is able to solve about 32 % of the problems after 150 evaluations with an accuracy level of $\tau = 10^{-7}$ and NOMAD solves about 12 % of the problems.

The data profiles show that the surrogate model based algorithms, in particular SO-I and SO-MI, perform in general best. SO-I performs for the accuracy $\tau = 10^{-1}$ only slightly better than SO-MI. The performance difference increases, however, with decreasing τ (see also Fig. 2), which can be attributed to the different ways for generating candidate points and the weight adjustment for the scoring criteria (see explanation after the proof of Theorem 1). SO-I outperforms all other algorithms especially as τ decreases, indicating that SO-I is able to find accurate solutions more often than the other algorithms. SO-I is only initially (for

fewer than 100 function evaluations) outperformed by local-SO-I. The reason for this is that local-SO-I searches only close to the best point found so far for improvements, and therefore it is possible to find improvements faster than SO-I. The figures show, however, also that the data profile of local-SO-I does not improve for many consecutive function evaluations (the steps are longer than for SO-I), which may be due to the algorithm having difficulties escaping from local optima. SO-I, on the other hand, uses also candidate points that are uniformly selected from the whole variable domain and is thus able to escape from local optima. Hence, if the total number of allowed function evaluations is restricted to 100 or fewer, local-SO-I may be the better choice because in this case converging fast to a local optimum may be more important than the global exploration of the variable domain.

Using only a local search may, however, depending on the size of the variable domain, have an influence on the algorithm's ability of finding feasible solutions. Local-SO-I fails for fewer problems than SO-I for the binary problem 8, and it fails for more problems than SO-I for problem 1, which has a comparatively large variable domain. Thus, local-SO-I may not be very efficient for constrained problems with large Ω_b.

GA's performance is better when τ is larger (Fig. 1a), and its performance decreases significantly as τ decreases, which indicates that GA has difficulties finding high-accuracy solutions. VNS-i performs in general worse than VNS-ii. VNS-i not only fails more often than VNS-ii to find feasible solutions, but also when feasible points were found, the solution quality is worse. Hence, using SO-I for finding a first feasible solution and applying thereafter MEIGO's VNS (as done in VNS-ii) improves its performance for constrained problems. VNS-ii and NOMAD perform very similarly.

Figure 2 shows performance profiles with logarithmic scales for accuracy levels $\tau = 10^{-1}$ and $\tau = 10^{-7}$. The figures show that SO-I is the fastest solver for more than 50 % of the problems while all other algorithms are the fastest for less than 20 % of all problems for $\tau \leq 10^{-7}$. Figure 2a shows, for example, that for a performance ratio of $\alpha = 2$ SO-I and NOMAD have a performance difference of about 40 % which implies that for 40 % of the problems NOMAD needs twice the number of function evaluations to reach the accuracy level $\tau = 10^{-1}$. Note that NOMAD failed to find feasible solutions for more than twice the number of trials than the surrogate model based algorithms and therefore NOMAD's performance graph is not very accurate. However, NOMAD's performance is regardless of this fact worse than SO-I and SO-MI. The performance profiles also show that the VNS methods are outperformed by the surrogate model based algorithms and that VNS-ii performs better than VNS-i.

The reason for the bad performance of GA can be associated to the number of function evaluations that is generally required for finding good solutions because the number of individuals and generations have to be large. If fewer individuals are used, the diversity within the generation is reduced, whereas reducing the number of generations has a negative effect on the evolutionary character of the algorithm. This algorithm is therefore not suitable for computationally expensive black-box functions.

7 Conclusions

In this paper the algorithm SO-I for computationally expensive integer black-box global optimization problems that may have computationally expensive constraints has been introduced. For constrained problems SO-I generates a feasible point in a first optimization phase that minimizes a constraint violation function. Once a feasible point has been found, a penalty-augmented objective function is minimized in the second optimization phase. Candidates for

the next expensive function evaluation are generated by randomly perturbing the best *feasible* point found so far and by uniformly selecting points from the box-constrained variable domain. A radial basis function surrogate model is used to select the best candidate for doing the next expensive function evaluation.

The performance of SO-I was numerically compared to a genetic algorithm, NOMAD, a slightly altered version of SO-MI [48], local-SO-I, which is a version of SO-I that uses only a local search for determining sample points, and two VNS implementations. While genetic algorithms and VNS have been widely used in the literature to solve discrete optimization problems, a study that examines the performance of NOMAD and surrogate model based algorithms such as SO-I has not been conducted.

The algorithms were compared on 17 generic test problems. Since there is no "widely used" test bench for discrete global optimization algorithms, problems with different characteristics have been examined. Thus, problems where the continuous relaxation is multimodal or unimodal, problems with linear and nonlinear objective and constraint functions as well as binary problems and problems where the integer variables may take on a large range of values have been included in the comparison. Furthermore, the algorithms were applied to eight realizations of application problems arising from throughput maximization and hydropower generation maximization.

Numerical experiments showed that the surrogate model based algorithms (SO-I and SO-MI) outperformed all other algorithms. The numerical results indicate that using surrogate models is a computationally efficient and promising approach for solving integer optimization problems in general. The convergence of SO-I follows from a simple counting argument. Moreover, as opposed to SO-MI, SO-I is able to find feasible solutions itself and does not require a user-supplied feasible point. SO-I's first optimization phase may thus be linked with various other algorithms to efficiently find a first feasible point. Once such a point has been encountered, another search routine may be applied as shown in the numerical experiments by first using SO-I and subsequently using SO-MI or VNS.

It is important to remind the reader that because the focus is on computationally expensive functions, the computational effort is measured in the number of objective function evaluations. For expensive functions that need, for example, several minutes or more per function evaluation, the objective function evaluation requires at least an order of magnitude more CPU time than the remaining calculations of the optimization algorithm steps. Hence in this situation, the number of objective function evaluations is the dominant issue. However, for problems with very inexpensively computed objective function values, the SO-I algorithm would not necessarily perform as well as GA or VNS, for example, comparatively because the reduction in evaluations might be outweighed by the extra computation cost for the surrogate model construction.

As more and more engineering and management problems are based on computationally expensive black-box objective and constraint functions, it is necessary to use as few costly evaluations as possible for obtaining accurate solutions within an acceptable time frame. The SO-I algorithm shows great promise to achieve this goal, and thus substantial savings in computation times are possible. An extension of SO-I for better handling constraints will be considered in the future.

Acknowledgments The project has been supported by NSF CISE: AF 1116298 and CPA-CSA-T 0811729. The first author thanks the Finnish Academy of Science and Letters, Vilho, Yrjö and Kalle Väisälä Foundation for the financial support. The authors thank the anonymous reviewers for their helpful comments and suggestions.

References

1. Abramson, M.A., Audet, C.: Convergence of mesh adaptive direct search to second-order stationary points. SIAM J. Optim. **17**, 606–619 (2006)
2. Abramson, M.A., Audet, C., Chrissis, J., Walston, J.: Mesh adaptive direct search algorithms for mixed variable optimization. Optim. Lett. **3**, 35–47 (2009)
3. Abramson, M.A., Audet, C., Couture, G., Dennis, J.E. Jr., Le Digabel, S.: The NOMAD project. Software available at http://www.gerad.ca/nomad (2011)
4. Abramson, M.A., Audet, C., Dennis Jr, J.E.: Filter pattern search algorithms for mixed variable constrained optimization problems. SIAM J. Optim. **11**, 573–594 (2004)
5. Audet, C., Béchard, V., Le Digabel, S.: Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search. J. Global Optim. **41**, 299–318 (2008)
6. Bäck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press, Oxford (1996)
7. Bäck, T., Fogel, D., Michalewicz, Z.: Handbook of Evolutionary Computation. Oxford University Press, Oxford (1997)
8. Booker, A.J., Dennis Jr, J.E., Frank, P.D., Serafini, D.B., Torczon, V., Trosset, M.W.: A rigorous framework for optimization of expensive functions by surrogates. Struct. Multidiscip. Optim. **17**, 1–13 (1999)
9. Bussieck, M.R., Drud, A.S., Meeraus, A.: MINLPLib—a collection of test models for mixed-integer nonlinear programming. INFORMS J. Comput. **15**, 114–119 (2003)
10. Coit, D.W., Smith, A.E.: Reliability optimization of series-parallel systems using a genetic algorithm. IEEE Trans. Reliab. **45**, 254–266 (1996)
11. Currin, C., Mitchell, T., Morris, M., Ylvisaker, D.: A Bayesian approach to the design and analysis of computer experiments. Technical report, Oak Ridge National Laboratory, Oak Ridge, TN (1988)
12. Davis, E., Ierapetritou, M.: Kriging based method for the solution of mixed-integer nonlinear programs containing black-box functions. J. Global Optim. **43**, 191–205 (2009)
13. Dorigo, M.: Optimization, Learning and Natural Algorithms. Ph.D. thesis, Politecnico di Milano, Italie (1992)
14. Duchon, J.: Constructive Theory of Functions of Several Variables. Springer, Berlin (1977)
15. Fiechter, C.-N.: A parallel tabu search algorithm for large traveling salesman problems. Discret. Appl. Math. **51**, 243–267 (1994)
16. Friedman, J.H.: Multivariate adaptive regression splines. Ann. Stat. **19**, 1–141 (1991)
17. Gershwin, S.B., Schor, J.E.: Efficient algorithms for buffer space allocation. Ann. Oper. Res. **93**, 117–144 (2000)
18. Glaz, B., Friedmann, P.P., Liu, L.: Surrogate based optimization of helicopter rotor blades for vibration reduction in forward flight. Struct. Multidiscip. Optim. **35**, 341–363 (2008)
19. Glover, F.W.: Heuristics for integer programming using surrogate constraints. Decis. Sci. **8**, 156–166 (1977)
20. Glover, F.W.: Tabu search—part 1. ORSA J. Comput. **1**, 190–206 (1989)
21. Glover, F.W.: Tabu search—part 2. ORSA J. Comput. **2**, 4–32 (1990)
22. Glover, F.W.: A Template for Scatter Search and Path Relinking. Springer, Heidelberg (1998)
23. Gutmann, H.-M.: A radial basis function method for global optimization. J. Global Optim. **19**, 201–227 (2001)
24. Hansen, P., Mladenović, N.: Variable neighborhood search: principles and applications. Eur. J. Oper. Res. **130**, 449–467 (2001)
25. Haupt, R.L.: Antenna design with a mixed integer genetic algorithm. IEEE Trans. Antennas Propag. **55**, 577–582 (2007)
26. Hesser, J., Männer, R.: Towards an optimal mutation probability for genetic algorithms. Lect. Notes Comput. Sci. **496**, 23–32 (1991)
27. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975)
28. Holmström, K.: An adaptive radial basis algorithm (ARBF) for expensive black-box global optimization. J. Global Optim. **41**, 447–464 (2008)
29. Hu, J., Ogras, U.Y., Marculescu, R.: System-level buffer allocation for application-specific networks-on-chip router design. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **25**, 2919–2933 (2006)
30. Jones, D.R.: A taxonomy of global optimization methods based on response surfaces. J. Global Optim. **21**, 345–383 (2001)
31. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. J. Global Optim. **13**, 455–492 (1998)
32. Jouhaud, J.-C., Sagaut, P., Montagnac, M., Laurenceau, J.: A surrogate-model based multidisciplinary shape optimization method with application to a 2D subsonic airfoil. Comput. Fluids **36**, 520–529 (2007)

33. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of IEEE International Conference on, Neural Networks, vol. 4, pp. 1942–1948 (1995)
34. Kennedy, J., Eberhart, R.: Swarm Intelligence. Morgan Kaufmann, San Francisco (2001)
35. Koziel, S., Michalewicz, Z.: Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. Evolut. Comput. **7**, 19–44 (1999)
36. Laguna, M., Martí, R.: Experimental testing of advanced scatter search designs for global optimization of multimodal functions. Technical report, University of Colorado at Boulder (2000)
37. Lam, X.B., Kim, Y.S., Hoang, A.D., Park, C.W.: Coupled aerostructural design optimization using the kriging model and integrated multiobjective optimization algorithm. J. Optim. Theory Appl. **142**, 533–556 (2009)
38. Land, A.H., Doig, A.G.: An automatic method of solving discrete programming problems. Econometrica **28**, 497–520 (1960)
39. Laskari, E.C., Parsopoulos, K.E., Vrahatis, M.N.: Particle swarm optimization for integer programming. In: Proceedings of the 2002 Congress on Evolutionary Computation vol. 2, pp. 1582–1587 (2002)
40. Li, C., Wang, F.-L., Chang, Y.-Q., Liu, Y.: A modified global optimization method based on surrogate model and its application in packing profile optimization of injection molding process. Int. J. Adv. Manuf. Technol. **48**, 505–511 (2010)
41. Li, F., Shoemaker, C.A., Wei, J., Fu, X.: Estimating maximal annual energy given heterogeneous hydropower generating units with application to the Three Gorges System. J. Water Resour. Plan. Manag. doi:10.1061/(ASCE)WR.1943-5452.0000250 (2012)
42. Liang, Y.-C.: An ant colony optimization algorithm for the redundancy allocation problem (RAP). IEEE Trans. Reliab. **53**, 417–423 (2004)
43. Malek, M., Guruswamy, M., Pandya, M., Owens, H.: Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. Ann. Oper. Res. **21**, 59–84 (1989)
44. Michalewicz, Z., Fogel, D.: How to Solve it: Modern Heuristics. Springer, New York (2004)
45. Mladenović, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**, 1097–1100 (1997)
46. Moré, J.J., Wild, S.M.: Benchmarking derivative-free optimization algorithms. SIAM J. Optim. **20**, 172–191 (2009)
47. Müller, J., Piché, R.: Mixture surrogate models based on Dempster-Shafer theory for global optimization problems. J. Global Optim. **51**, 79–104 (2011)
48. 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**, 1383–1400 (2013)
49. Myers, R.H., Montgomery, D.C.: Response Surface Methodology, Process and Product Optimization Using Designed Experiments. Wiley-Interscience Publication, New York (1995)
50. Pichitlamken, J., Nelson, B.L., Hong, L.J.: A sequential procedure for neighborhood selection-of-the-best in optimization via simulation. Eur. J. Oper. Res. **173**, 283–298 (2006)
51. Poli, R.: Analysis of the publications on the applications of particle swarm optimisation. J. Artif. Evolut. Appl. **1–10**, 2008 (2008)
52. Powell, M.J.D.: The theory of radial basis function approximation in 1990. In: Advances in Numerical Analysis, Wavelets, Subdivision Algorithms and Radial basis Functions, vol. 2, pp. 105–210. Oxford University Press, Oxford (1992)
53. Rashid, K., Ambani, S., Cetinkaya, E.: An adaptive multiquadric radial basis function method for expensive black-box mixed-integer nonlinear constrained optimization. Eng. Optim. doi:10.1080/0305215X.2012.665450 (2012)
54. Regis, R.G., Shoemaker, C.A.: Constrained global optimization of expensive black-box functions using radial basis functions. J. Global Optim. **31**, 153–171 (2005)
55. Regis, R.G., Shoemaker, C.A.: A stochastic radial basis function method for the global optimization of expensive functions. INFORMS J. Comput. **19**, 497–509 (2007)
56. Regis, R.G., Shoemaker, C.A.: Improved strategies for radial basis function methods for global optimization. J. Global Optim. **37**, 113–135 (2007)
57. Schoen, F.: A wide class of test functions for global optimization. J. Global Optim. **3**, 133–137 (1993)
58. Shan, S., Wang, G.G.: Survey of modeling and optimization strategies to solve high-dimensional design problems with computationally-expensive black-box functions. Struct. Multidiscip. Optim. **41**, 219–241 (2010)
59. Shi, Y., Eberhart, R.: A modified particle swarm optimizer. In: Proceedings of IEEE International Conference on Evolutionary Computation, pp. 69–73 (1998)
60. Spinellis, D.D., Papadopoulos, C.T.: A simulated annealing approach for buffer allocation in reliable production lines. Ann. Oper. Res. **93**, 373–384 (2000)

61. Srinivas, M., Patnaik, L.M.: Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Trans. Syst. Man Cybern. **24**, 565–667 (1994)
62. Törn, A., Zilinskas, A.: Global Optimization. In: Lecture Notes in Computer Science, 350. Springer, Berlin (1989)
63. Wild, S.M., Regis, R.G., Shoemaker, C.A.: ORBIT: optimization by radial basis function interpolation in trust-regions. SIAM J. Sci. Comput. **30**, 3197–3219 (2007)
64. Wu, Q.H., Cao, Y.J., Wen, J.Y.: Optimal reactive power dispatch using an adaptive genetic algorithm. Int. J. Electr. Power Energy Syst. **20**, 563–569 (1998)
65. Ye, K.Q., Li, W., Sudjianto, A.: Algorithmic construction of optimal symmetric Latin hypercube designs. J. Stat. Plan. Inference **90**, 145–159 (2000)
66. Zhang, J., Chung, H.S.-H., Lo, W.-L.: Clustering-based adaptive crossover and mutation probabilities for genetic algorithms. IEEE Trans. Evol. Comput. **11**, 326–335 (2007)