Journal of Computational and Applied Mathematics 236 (2011) 759–764

Contents lists available at SciVerse ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam

A metamodel-assisted evolutionary algorithm for expensive optimization[☆]

Changtong Luo [a,*], Shao-Liang Zhang [b], Chun Wang [a], Zonglin Jiang [a]

[a] *Institute of Mechanics, Chinese Academy of Sciences, Beijing 100190, China*

[b] *Department of Computational Science and Engineering, Nagoya University, Nagoya 464-8603, Japan*

ARTICLE INFO

Keywords:
Expensive optimization
Evolutionary algorithm
Low dimensional simplex evolution
Metamodel
Radial basis function

ABSTRACT

Expensive optimization aims to find the global minimum of a given function within a very limited number of function evaluations. It has drawn much attention in recent years. The present expensive optimization algorithms focus their attention on metamodeling techniques, and call existing global optimization algorithms as subroutines. So it is difficult for them to keep a good balance between model approximation and global search due to their two-part property. To overcome this difficulty, we try to embed a metamodel mechanism into an efficient evolutionary algorithm, low dimensional simplex evolution (LDSE), in this paper. The proposed algorithm is referred to as the low dimensional simplex evolution extension (LDSEE). It is inherently parallel and self-contained. This renders it very easy to use. Numerical results show that our proposed algorithm is a competitive alternative for expensive optimization problems.

1. Introduction

Global optimization (GO) aims to find a best solution to a given problem, or mathematically, to find a vector $\boldsymbol{x}^*$ within a feasible region $\Omega \subset \mathbb{R}^n$ such that $f(\boldsymbol{x}^*) \le f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \Omega$. GO is a challenging task because the gradient based algorithms such as quasi-Newton methods and nonlinear conjugate gradient methods will get stuck at a stationary point or a local minimum. During the past few decades, great efforts have been made on GO, and a number of excellent methods for global optimization have been presented including branch-and-bound methods (BB), adaptive simulated annealing (ASA) [1], the covariance matrix adaptation evolution strategy (CMA-ES) [2], differential evolution (DE) methods [3], particle swarm optimization (PSO) [4], low dimensional simplex evolution (LDSE) methods [5], etc. Despite their prominent efficiency, they still require a large number of function evaluations to escape the local minima. However, in many engineering applications, the evaluation of the objective function is computationally and/or experimentally expensive. We take optimal shape design as an example, in which for a given design vector ($\boldsymbol{x} = (x_1, \ldots, x_n)$), we need a computational fluid dynamics (CFD) simulation to obtain its corresponding performance $f(\boldsymbol{x})$. The CFD simulation may take several minutes, several hours, or even several days. The CFD simulation can be regarded as a black-box function in the optimization problem. In this case, we need to find a reasonably good vector $\boldsymbol{x}^*$ within a very limited number of CFDs. This is known as an expensive optimization problem. General GO algorithms do not work for it.

In short, expensive optimization aims to find the global minimum of a given function within a very limited number of function evaluations. It has at least three properties as follows. (1) The objective function is very costly to evaluate. (2) The

[☆] This research was supported by the National Natural Science Foundation of China (Grants 10632090 and 10621202).
[*] Corresponding author. Fax: +1 484 805 7242.
E-mail address: luo@imech.ac.cn (C. Luo).

0377-0427/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.cam.2011.05.047

landscape of the objective function might be not so complex. However, it is nonconvex and has multiple minimum points. (3) The accuracy of the solution is not very high (usually it is acceptable if just the relative error is less than 1%). However, the function evaluations are strictly limited in number (to thousands, hundreds, dozens, or even less).

In 1993, Jones et al. provided a dividing rectangles (DIRECT) method [6], in which only the function value is used (thus it is direct and derivative-free). It can be regarded as an early version of expensive optimization. But it requires many more function evaluations than expensive optimization can give. To accelerate the convergence speed, a class of approximation model based GO algorithms have also been presented. They can take advantage of the inherent smoothness of the target problems, so they will, one hopes, converge faster than direct methods. Several approximation models have been proposed. Powell used a multivariate polynomial interpolation model within a trust-region framework [7]. Jones et al. used a kriging model [8]. Ishikawa et al. used a radial basis function (RBF) model [9]. Among these models the radial basis function (RBF) model is a most promising model due to its simplicity and stability. Famous expensive optimization algorithms including CORS-RBF in [10], RBF-G in [11] and ARBF in [12] are all based on the RBF model. Expensive optimization has drawn much attention in recent years. *Evolutionary Computation in Expensive Optimization Problems* has been accepted as a special session by the 2008 IEEE World Congress on Computational Intelligence (WCCI 2008); now it has been accepted as a special session again by the incoming WCCI 2010.

The present expensive optimization algorithms such as CORS-RBF, ARBF, etc., focus their attention on metamodeling and call existing GO algorithms as subroutines. Thus it is difficult for them to keep a good balance between metamodeling and global search due to their two-part property. In this work, we try to overcome this shortcoming by integrating a metamodel mechanism (including the RBF interpolation and tabu search) with an efficient global optimization, low dimensional simplex evolution (LDSE) [5]. Numerical results show that the resulting algorithm is a competitive alternative for expensive global optimization.

2. A brief review of low dimensional simplex evolution (LDSE)

Low dimensional simplex evolution (LDSE) [5] is a real-coded evolutionary algorithm (EA) for box-constrained global optimization of the form

$$\min_{\boldsymbol{x}\in\Omega} f(\boldsymbol{x}).$$

Similarly to other population-set based algorithms, LDSE maintains a population set $\vec{X}(t)$ of N individuals (points in $\mathbb{R}^n$) $X_i(t),\ i = 1, 2, \ldots, N$, during the evolutionary progress. The evolutionary progress is to drive these points to the vicinity of the global minimizer. The driving is done by replacing all *bad* points in the current population with new better points from generation to generation.

For each individual in the current population, $m + 1$ individuals are randomly selected to form an m-simplex, where $m \ll n$. Each individual $X_i(t)$ tries to improve itself in a framework of try–try–struggle, which will be described as follows. At the beginning, it has two chances. The first chance is provided by the simplex reflection. If the reflection point X_r is better than $X_i(t)$, $X_i(t)$ will be replaced by X_r. As a result, the individual $X_i(t)$ is promoted. Otherwise a second chance, the simplex contraction, will be carried out. Similarly, $X_i(t)$ will get promoted if the contraction point X_c is better. However, if the individual $X_i(t)$ has lost the previous two chances and still cannot achieve the average profit (that is, its function value is greater than or equal to the average value of the current population), it will make its last struggle. The procedure of the LDSE can be outlined as follows.

Procedure of LDSE algorithm:

Step 1. Initialize: Input population size N, initial bounds $\boldsymbol{l}$, $\boldsymbol{u}$, scaling factors α and β. Set the current generation $t := 0$; and initialize population $\vec{X}(0) = \{X_1(0), X_2(0), \ldots, X_N(0)\}$, where $X_i(0) \in \mathbb{R}^n$.

Step 2. Evaluate population: For each individual in the current population $\vec{X}(t)$, compute $f(X_i(t))$; set the current position $i := 1$.

Step 3. Update population: If the current position $i \leq N$, perform the following steps.

(3.1) Construct simplex: Randomly choose $m + 1$ mutually different individuals $X_{r_i},\ i = 1, 2, \ldots, m + 1$, from the current population, find their best X_b and the worst X_w, and calculate the centroid $\bar{X} = \frac{1}{m}\sum_{r_i\neq w} X_{r_i}$.

(3.2) Try reflection: Compute the reflection point $X_r = \bar{X} + \alpha \cdot (\bar{X} - X_w)$. If $f(X_r) < f(X_i(t))$, then $X_i(t + 1) = X_r$; set the current position $i := i + 1$, and return to step 3.

(3.3) Try contraction: Compute the contraction point $X_c = \bar{X} + \beta \cdot (X_w - \bar{X})$. If $f(X_c) < f(X_i(t))$, then $X_i(t+1) = X_c$; set the current position $i := i + 1$, and return to step 3.

(3.4) Struggle: If $f(X_i(t)) \geq \frac{1}{N}\sum_i f(X_i(t))$ then compute the struggle point

$$X_s = \begin{cases} X_i(t) + 0.618 \cdot (X_b(t) - X_i(t)), & \text{if } f(X_{r_b}(t)) < f(X_i(t)); \\ X_i(t) + 0.382 \cdot (X_i(t) - X_w(t)), & \text{else;} \end{cases}$$

let $X_i(t + 1) = X_s$, set the current position $i := i + 1$, and return to step 3.

Step 4. Check point: If some stopping criterion is satisfied, output the best-so-far individual X^* and its function value $f(X^*)$. Otherwise, set the current generation $t := t + 1$, set the current position $i := 1$ and return to step 3.

It is clear from the above procedure that LDSE hybridizes EA and the Nelder–Mead method with essential modifications. It generates new trial points in a Nelder–Mead way, and the individuals survive by the rule of natural selection. However, the simplex therein is low dimensional and real-time constructed; and the simplex operators are employed selectively (i.e., the expansion and reduction operators are discarded) and a new simplex operator (the last struggle) is introduced. Meanwhile, each individual is updated in a framework of try–try–struggle.

LDSE has shown its prominent performance over an improved version of differential evolution (DE) [5,13]. However, it is still not fast enough for expensive optimization problems.

3. Metamodeling

The idea of metamodeling is to take the advantage of the inherent smoothness of the expensive optimization problems. To this end, it constructs a series of metamodels $m_k(\boldsymbol{x})$ $(k = p, p+1, p+2, \ldots)$ to approximate the costly target function $f(\boldsymbol{x})$, where the model function $m_k(\boldsymbol{x})$ is much cheaper to evaluate (usually it takes much less than one second). We hope to have $m_k(\boldsymbol{x}) \to f(\boldsymbol{x})$ as $k \to \infty$. Then the model functions are used repeatedly to identify the promising points for the target function. In other words, the expensive global optimization problem

$$\min_{\boldsymbol{x}\in\Omega} f(\boldsymbol{x}) \tag{1}$$

is now reduced into a series of cheap global optimization problems

$$\min_{\boldsymbol{x}\in\Omega} m_k(\boldsymbol{x}), \quad k = p, p+1, p+2, \ldots. \tag{2}$$

The metamodel (also called the approximation model, the surrogate model, or the response surface) might be constructed using the multivariate polynomial interpolation model, the kriging model, or the radial basis function (RBF) model (they have been mentioned in Section 1). Users can also choose other metamodels for approximating the costly target function. In this work, we choose the radial basis function (RBF) model because of its simplicity and stability. Suppose we have k distinct points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ with known function values $f_i = f(\boldsymbol{x}_i),\ i = 1, \ldots, k$; then the model function $m_k(\boldsymbol{x})$ has the form

$$m_k(\boldsymbol{x}) = \sum_{i=1}^{k} \omega_i \Phi(\|\boldsymbol{x} - \boldsymbol{c}_i\|).$$

We use the multiquadric type basis function $\Phi(r) = \sqrt{r^2 + \beta^2}$ in our experiments. For more information of RBF, refer to [14].

Keep in mind that the above mentioned assumption $k \to \infty$ is impractical because we want to optimize the costly target function within a very limited number of function evaluations, i.e., k is upper bounded, $k < K$. Usually the upper bound K is limited to thousands, hundreds, dozens, or even less. We need to find a reasonably good vector $\boldsymbol{x}^*$ within K costly evaluations.

In fact, we can get a reasonably good vector $\boldsymbol{x}^*$ provided that (1) the response surface $m_k(\boldsymbol{x})$ is good enough, and (2) the global optimization search (over $m_k(\boldsymbol{x})$) is reliable. Obviously, it is difficult to get a good approximation model with a very limited number of function evaluations. This makes it necessary to keep a good balance between the model approximation and the global search during the optimization process. Usually, the assumption (1) is more critical in the earlier phase of optimization and the assumption (2) becomes more important in the later phase.

4. The metamodel-assisted LDSE algorithm

Our new algorithm is referred to as the low dimensional simplex evolution extension (LDSEE, pronounced LDC). It is based on the previously described low dimensional simplex evolution (LDSE), and uses the radial basis function (RBF) response surface as its metamodel. To make a better approximation of the costly target function, the idea of tabu search [15] is also applied.

LDSEE starts from an initial RBF response surface determined by a set of initial points. Suppose the number of initial points is p. Then the initial RBF model is $m_p(\boldsymbol{x})$. Suppose we have g CPUs for parallel computing; then LDSEE generates g new promising points (thus LDSEE is inherently parallel) at every iteration of the following three kinds:

(1) the global minimum of the RBF model: $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}\in\Omega} m_p(\boldsymbol{x})$;
(2) s low simplex reflection points: $X_r^l = \bar{X} + \alpha \cdot (\bar{X} - X_w),\ l = 1, 2, \ldots, s$;
(3) $(g-s-1)$ K-far random points: $\boldsymbol{x}_K^l = \arg\max_{i\in\{1,2,\ldots,K\}} \min_{j\in\{1,2,\ldots,k,k+1\}} \|\boldsymbol{x}_{r_i} - \boldsymbol{x}_j\|, l = 1, 2, \ldots, (g-s-1)$.

By "promising" we mean that it is hoped that the point can (1) improve the RBF response surface or (2) help to locate a better point for the target function. This step is referred to as global search. Then the newly obtained promising points will be added to the tabu list and used to update the RBF response surface and get a new RBF model $m_{p+g}(\boldsymbol{x})$. This step is referred to as surface reconstruction. The above steps (global search and surface reconstruction) are repeated until the limited number of costly function evaluations are used up. In general, LDSEE puts emphasis on the surface reconstruction at the beginning and the global search in the end. The procedure of the LDSEE can be outlined as follows.

Table 1
Characteristics of test functions.

func.name	abbr	dim	domain	no.min
1D	1D	1	$[-4, 4]$	4
Peaks	PK	2	$[-4, 4]^2$	2
Six-hump camel back	CB	2	$[-4, 4]^2$	6
Branin	BR	2	$[-4, 4]^2$	6
Goldstein-Price	GP	2	$[-2, 2]^2$	5
Hartman-3	H3	3	$[0, 1]^3$	5
Sheckel-10	S10	4	$[0, 10]^4$	10
Hartman6	H6	6	$[0, 1]^6$	5

Procedure of LDSEE:

Step 1. (Initialization) Input the number of initial points p, the number of CPUs g, the maximum number of expensive function evaluations K, and the number of nodes at the jth direction N_j. Select a set of initial vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p$ and calculate their target function values $f_i = f(\boldsymbol{x}_i),\ i = 1, 2, \ldots, p$. Find the best vector $\boldsymbol{x}^*$ and its target value f^*. Let the tabu list be $T = \{\boldsymbol{x}_i |\ i = 1, 2, \ldots, p\}$, set $k := p$, the initial tabu radius $\delta := (\delta_1, \ldots, \delta_n)$ for general points, where $\delta_j = \frac{1}{2(N_j+1)} \cdot 0.9,\ j = 1, 2, \ldots, n$, and the initial tabu radius for the elite point $\delta_e = 0.9$.

Step 2. (Surface construction) Construct a RBF response surface $m_k(\boldsymbol{x})$ with known data $(\boldsymbol{x}_i, f_i),\ i = 1, 2, \ldots, k$.

Step 3. (Global search) Use the LDSE algorithm (see Section 2) for solving the cheap problem

$$\min_{\boldsymbol{x}\in\Omega} m_k(\boldsymbol{x})$$

with the tabu radius δ to get the global minimum of the RBF model $\boldsymbol{x}^*$. Generate s low simplex reflection points $\boldsymbol{x}_r^l$ and $(g - s - 1)$ K-far random points $\boldsymbol{x}_K^l$ with elite tabu radius δ_e as mentioned above.

Step 4. (Target evaluation) Calculate the target function value at the g promising points obtained in step 3 and update the best vector and its target value $(\boldsymbol{x}^*, f^*)$ if $f(\boldsymbol{x}^{k+l}) < f^*,\ l = 1, 2, \ldots, g$.

Step 5. (Stop checking) If $k \geq K$, stop and output the best vector $\boldsymbol{x}^*$ and its target value f^*. Otherwise, set $k := k + g$, update the tabu list $T = \{\boldsymbol{x}_i |\ i = 1, 2, \ldots, k\}$, update the general tabu radius δ, where $\delta_j = \frac{1}{2(N_j+1)} \cdot 0.9 \cdot \left(1 - \frac{k-p}{K-p}\right), j = 1, 2, \ldots, n$, and the elite tabu radius $\delta_e = 0.9 * \cos\left(2 \cdot \frac{k-p}{K-p}\right)$, and return to step 2.

In contrast to the original LDSE described in Section 2, LDSEE uses the RBF response surfaces to locate the minimum of the target function and requires that every new generated individual must keep a dynamic distance away from the points in the tabu list.

5. Numerical results

The proposed algorithm LDSEE is implemented in C++. As an integrated evolutionary algorithm, LDSEE is self-contained. It does not rely on other GO algorithms and is very easy to use. To test the performance of LDSEE, we choose a set of box-constrained multimodal functions (see Table 1) for performing our numerical experiments. Most of these functions (the last five) are from [16], and frequently cited for testing the performance of expensive optimization algorithms [6,10,12]. The first one-dimensional function (1D) is defined as $f(x) = -(((3 * x - 1) * \sin x - 2) * \cos x - 1)$. The second function (Peaks) is the logo function of Matlab (a popular numerical computing programming language, developed by MathWorks). The third function (2D Six-hump camel back) is a multimodal function frequently used to test GO algorithms [3]. In Table 1, the test function name, its abbreviation, dimension, domain, and the number of local minima are denoted as func.name, abbr, dim, domain, no.min respectively. As described in [10], these functions are not really costly to evaluate, but their multimodal property is similar to those of the real world costly functions. Therefore, the performance on these test functions is expected to mimic the performance on the real world costly functions.

In our numerical experiments, uniform inner grid points are used as initial points. For example, if we consider a function within $[0, 1] \times [-1, 0]$ and we want to seed two sample points in each direction, there will be four initial points $(0.333, -0.333)$, $(0.333, -0.666)$, $(0.666, -0.333)$ and $(0.666, -0.666)$. Note that the number of inner grid points will be quite large for high dimensional problems. To avoid unnecessary sampling, an experimental design method such as Latin Hypercube [17] or Uniform Design [18] might be applied. However, in our experience, too few initial points might result in unreliable results.

The control parameters of LDSEE and the performances are listed in Table 2, where the function name abbreviation, the number of initial points, the number of search points, the actual global minimum, the minimum obtained, and the relative residue are denoted as func, no.ini, no.sch, act.min, obt.min and rel.res respectively. The control parameters of the global search part are set as follows. The scaling factors are $\alpha = 1.0$, $\beta = 0.333$. The number of CPUs is $g = 1$; thus $s = 0$. The population size is $N = 50$ and the maximum number of generations is $M = 200$. N and M are set large enough to ensure that the global search is reliable. It can be seen from Table 2 that the results are very encouraging.

Table 2
Settings and performances of the LDSEE algorithm.

func	no.ini	no.sch	act.min	obt.min	rel.res
1D	4	10	−6.80484	−6.80484	0.00002
PK	4	10	−0.6551	−0.6542	0.14
CB	4	10	−1.031	−1.027	0.39
BR	4	10	0.398	0.399	0.16
GP	9	20	3.0	3.0	0.0001
H3	8	20	−3.862782	−3.82629	0.94
S10	16	40	−10.5364	−10.4514	0.81
H6	64	30	−3.322368	−3.32143	0.028

Table 3
Number of function evaluations needed to achieve a function value with relative error less than 1% for different expensive optimization algorithms.

func	DIRECT	RBF-G	CORS-RBF(SP1)	CORS-RBF(SP2)	LDSEE
BR	63	44	34	40	**14**
GP	101	63	49	64	**29**
H3	83	43	**25**	61	28
S10	97	**51**	**51**	64	56
H6	213	112	108	104	**94**

Table 4
Number of function evaluations needed to achieve a function value with relative error less than 0.1% for PSO, DE, LDSE and LDSEE on the Six-hump camel back (CB) function.

Algorithm	PSO	DE	LDSE	LDSEE
no.eval	639	210	187	14

To compare with other existing expensive optimization algorithms, we cite the results from [10]. The comparison results are listed in Table 3, where DIRECT is presented in [6], and RBF-G is presented in [11]. The best result for each test function is marked in bold. It is shown that LDSEE performs the best on three out of five functions. For the other two functions (H3 and S10), the performance of LDSEE is very close to the best. Note that we use a preset number of search points in LDSEE; thus the relative error obtained might be much less than 1% (e.g., for GP and H6). If we stop at 1%, the required number of function evaluations might be smaller. On the basis of these results, we conclude that LDSEE is a competitive alternative for expensive optimization problems.

To help give readers a better understanding of the difference between the expensive optimization algorithms and the general (for cheap functions) GO algorithms, we give the number of function evaluations needed to achieve a function value with relative error less than 0.1% for different algorithms in Table 4. We can see that LDSEE needs far fewer function evaluations than LDSE, DE and PSO. This means that LDSEE works much better if the objective function is costly to evaluate. However this does NOT mean that LDSEE is better than LDSE, DE or PSO in general cases. In fact, these algorithms are designed for different kinds of problems. LDSEE is designed for expensive optimization problems while the others are for general GO. If the objective is cheap to evaluate, LDSEE might consume much more CPU time and memory.

6. Conclusion

We have presented a new algorithm named the low dimensional simplex evolution extension (LDSEE) for expensive optimization problems. LDSEE integrates the radial basis function (RBF) interpolation and tabu search with an efficient evolutionary algorithm, low dimensional simplex evolution (LDSE). LDSEE is inherently parallel and very easy to use. As an integrated algorithm, LDSEE can keep a good balance between the model approximation and the global search. As a self-contained algorithm, it does not rely on other GO algorithms. Numerical results indicate that it is a very promising algorithm.

In practical applications of LDSEE, parallelization implementation is very important for both the global search process and the costly function evaluations. Meanwhile, implicit constraint handling is another problem to be concerned with. These topics are left for our future research.

References

[1] L. Ingber, Simulated annealing: practice versus theory, J. Math. Comput. Model. 18 (1993) 29–57.
[2] N. Hansen, A. Ostermeier, Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation in: Proc. IEEE Conf. Evol Comput, 1996, pp. 312–317.
[3] R. Storn, K. Price, DE—a simple and efficient heuristic for global optimization over continuous space, J. Global Optim. 11 (1997) 341–359.
[4] J. Kennedy, R. Eberhart, Particle swarm optimization, in: IEEE Int. Conf. Neural Networks Conf. Proc., vol. 4, 1995, pp. 1942–1948.

[5] C.T. Luo, B. Yu, Low dimensional simplex evolution—a hybrid heuristic for global optimization, in: Proc. Eighth ACIS Int. Conf. Softw. Eng. Artif. Intell. Netw. Parallel Distrib. Comput., vol. 2, 2007, pp. 470–474.
[6] D.R. Jones, C.D. Perttunen, B.E. Stuckman, Lipschitzian optimization without the Lipschitz constant, J. Optim. Theory Appl. 79 (1993) 157–181.
[7] M.J.D. Powell, UOBYQA: unconstrained optimization by quadratic approximation, Math. Program. 92 (2000) 555–582.
[8] D.R. Jones, M. Schonlau, W.J. Welch, Efficient global optimization of expensive black-box functions, J. Global Optim. 13 (1998) 455–492.
[9] T. Ishikawa, M. Matsunami, An optimization method based on radial basis functions, IEEE Trans. Magn. 33 (1997) 1868–1871.
[10] R.G. Regis, C.A. Shoemaker, Constrained global optimization of expensive black box functions using radial basis functions, J. Global Optim. 31 (2005) 153–171.
[11] H.-M. Gutmann, A radial basis function method for global optimization, J. Global Optim. 19 (2001) 201–227.
[12] K. Holmström, An adaptive radial basis algorithm (ARBF) for expensive black-box global optimization, J. Global Optim. 41 (2008) 447–464.
[13] M.M. Ali, C. Khompatraporn, Z.B. Zabinsky, A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems, J. Global Optim. 31 (2005) 635–672.
[14] M.D. Buhmann, Radial Basis Functions: Theory and Implementations, Cambridge University Press, Cambridge, 2003.
[15] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Boston, 1997.
[16] L.C.W. Dixon, G.P. Szegö, The global optimisation problem: an introduction, in: L. Dixon, G. Szego (Eds.), Toward Global Optimization, vol. 2, 1978, pp. 1–15.
[17] M. McKay, R. Beckman, W. Conover, A comparison of three methods for selecting values of input variables in the analysis of output from a computer code, Technometrics 21 (1979) 239–246.
[18] K.T. Fang, Y. Wang, Number-Theoretic Methods in Statistics, Chapman & Hall, London, 1994.