J Glob Optim (2012) 52:45–55
DOI 10.1007/s10898-011-9678-1

Low dimensional simplex evolution: a new heuristic for global optimization

Changtong Luo · Bo Yu

Received: 11 April 2006 / Accepted: 30 January 2011 / Published online: 18 February 2011

Abstract This paper presents a new heuristic for global optimization named low dimensional simplex evolution (LDSE). It is a hybrid evolutionary algorithm. It generates new individuals following the Nelder-Mead algorithm and the individuals survive by the rule of natural selection. However, the simplices therein are real-time constructed and low dimensional. The simplex operators are applied selectively and conditionally. Every individual is updated in a framework of try-try-test. The proposed algorithm is very easy to use. Its efficiency has been studied with an extensive testbed of 50 test problems from the reference (J Glob Optim 31:635–672, 2005). Numerical results show that LDSE outperforms an improved version of differential evolution (DE) considerably with respect to the convergence speed and reliability.

Keywords Global optimization · Heuristic · Real-coded · Evolutionary algorithm · Differential evolution · Low dimensional simplex evolution

1 Introduction

Global optimization (GO) aims to find the best possible solution to an existing problem under some given feasibility constraints. It has become an important tool in many practical applications (see, e.g., [14,15,18,19]). However, GO is still a challenging research topic because practical problems might be highly nonlinear, non-convex, and/or involve many

This research has been supported by the National Natural Science Foundation of China (Grants 10632090 and 90916028).

C. Luo (✉)
Key Laboratory of High Temperature Gas Dynamics, Chinese Academy of Sciences,
100190 Beijing, China
e-mail: luo@imech.ac.cn

B. Yu
Department of Applied Mathematics, Dalian University of Technology, Dalian 116024, China

local extreme points. Even worse, the derivative of objective function might be unavailable, unreliable (numerical unstable, e.g., in the presence of noise), or hard to compute. In this paper, we consider the continuous global optimization with box-constraints of the form

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

where the feasible region $\Omega = \{\boldsymbol{x} \in \mathbb{R}^n | l_j < x_j < u_j, j = 1, 2, \ldots, n\}$. The objective function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ is of the black-box type but it is assumed to be continuous or at least piecewise continuous in general.

Existing GO algorithms can be roughly divided into two classes: deterministic algorithms [6] and stochastic heuristics [16,28]. A most widely used deterministic GO algorithm is the branch-and-bound (BB) method. It is a class of adaptive partition strategies based upon partition, sampling, and subsequent lower and upper bounding procedures [18]. Besides, there are several other deterministic GO algorithms including the dividing rectangles (DIRECT) provided by D. R. Jones, C. D. Perttunen and B. E. Stuckman [9,10], the multilevel coordinate search (MCS, based on DIRECT) provided by W. Huyer and A. Neumaier [7], cutting plane methods provided by H. Tuy [27], etc. Theoretically, all those deterministic algorithms might fail to get the global optimum because of their deterministic selection rules and non-ergodic properties. On the contrary, stochastic heuristics can get the global optimum in probability 1 due to their stochastic selection rules and ergodic properties. Dozens of stochastic GO algorithms have been provided during the past decades, one can mention adaptive simulated annealing (ASA) [8,17],covariance matrix adaptation evolution strategy (CMA-ES) [4], differential evolution (DE) [26], and particle swarm optimization (PSO) [12].

Dozens of GO software products have been developed in the last decades. They can be divided into three types. Some GO products are only a part of some general purpose mathematical software, e.g., the *genetic algorithm and direct search* toolbox in Matlab, the *NMinimize* package in Mathematica, etc. Some are professional GO solvers, e.g., BARON [24], LGO [20], LINGO, and many more. Some give a uniform interface to several GO solvers, e.g., GAMS, AMPL, etc. Usually, a professional GO solver is an integration of several algorithms with adaptive strategies and constraints-handling techniques. For more information on GO algorithms and products, consult [20,21] and the references therein.

Nevertheless, convergence speed and reliability are still the bottleneck of GO algorithms for their applications. In this paper, we try to get an efficient algorithm by hybridizing evolutionary and traditional optimization algorithms. We find that simple combinations do not work. It needs essential modifications to get an efficient hybrid evolutionary algorithm.

The rest of the paper is organized as follows. Section 2 presents four possible ways of hybridizing evolutionary and traditional algorithms. Section 3 describes the proposed hybrid algorithm, low dimensional simplex evolution (LDSE), in detail. In Sect. 4 we discuss the constraint handling in LDSE. Section 5 gives the numerical results. Finally, we draw some concluding remarks in Sect. 6.

2 Ways of hybridization

Evolutionary algorithms have the virtue of global search, while their local search ability is generally limited. Many population based algorithms (may) converge slowly even if the best solutions are close to the global optimizer. On the contrary, traditional algorithms are usually good at local search, but easily trapped into a local minimum or stationary point. As a result,

hybridization with a traditional algorithm is a good way to speed up evolutionary algorithm. Existing ways of hybridization could be divided into three classes:

(1) (E+T)-type, in which the EA is used for the global exploration (step 1) and then the traditional algorithm is used for the local exploitation (step 2). Specifically, in step 1, the EA is used to find a most promising basin of attraction and an initial point for step 2. In step 2, the traditional algorithm is used to discover the bottom of the basin from the initial point provided by EA. This type of hybridization is very easy to use, so it has been embedded in many GO software products (e.g., the optimization toolbox in Matlab) and is widely used in practical engineering computations.
(2) (E+T+E)-type, in which traditional algorithm is embedded in EA to improve the individuals in the current population. For example, GPL [22] is a combination of real-coded GA and Powell's method.
(3) (E<T>E)-type, in which traditional optimization operator is directly used as an evolutionary operator. For example, simplex-GA [22] and simplex coding genetic algorithm [5] are hybridizations of real-coded GA and downhill simplex method [13].

The above ways of hybridization do work in some cases. However, we need more efficiency and more reliability in practical applications. In this paper, we try to hybridize traditional and evolutionary algorithms in a new way:

(4) **(<ET>)-type**, in which some selected operators from the traditional optimization algorithm are used as evolutionary operators after essential modifications, and the modified operators are applied conditionally in conjunction with some other new introduced operators. Here, by <ET> we mean evolutionary and traditional algorithms are tightly integrated.

3 Low dimensional simplex evolution

3.1 Lower dimensional simplex operators

There are two kinds of operators in evolutionary algorithms: (1) the reproduction operators (such as crossover and mutation) to decide how to generate new individuals, and (2) the selection operator(s) to decide which individuals will survive to the next generation. In this subsection, we focus our attention on the reproduction operators. For the black-box type global optimization problem (1), direct search methods are good choices because they are derivative-free and only use the information of function values. We will derive our reproduction operators from direct search methods.

Downhill simplex method (also referred to as Nelder-Mead method) is a classic direct search method. It moves, deforms, and resizes a simplex shape until its volume becomes sufficiently small. The method was introduced by Spendley, Hext and Himsworth [25]. Afterward, Nelder and Mead developed a modified version that allows the procedure to adjust its search step according to the evaluation result of the new point generated [13]. Nelder-Mead method has a good performance of local search. This makes it possible to get an efficient hybrid EA by designing evolutionary operators inspired from Nelder-Mead.

Nelder-Mead method must maintain a full dimensional simplex (n-simplex for n-dimensional problems) during the search process to ensure its convergence. Once the dimension of the new transformed simplex becomes degenerated, the algorithm will be more likely to get stuck in a non-optimizer (i.e., neither global minimizer nor local minimizer). This is not what we expect. We have tried to remedy this defect by maintaining a point-set

with more (than $n+1$) points during the search process. In our experiments, we use n-simplex operators as the reproduction operators for generating new individuals/points. The population size N is set to be much lager than $n+1$. The results show that maintaining a set with more points does help the algorithm converge to global minimizer. However, the convergence speed becomes very slow as the population size N increases, especially for high dimensional problems.

In linear algebra, subspace iteration is a successful and widely used technique, e.g., Krylov subspace method for large-scale linear systems [3] is regarded as one of the ten most important classes of numerical methods in the 20th century. Subspace iteration can both save the required memory storage and accelerate the search process. The Nelder-Mead method and the subspace iteration motivate us to design our reproduction operators with lower dimensional simplex. We use three lower dimensional operators for generating new individuals: reflection, contraction and local learning. Suppose the dimension of the simplex is m, and $m \leq n$, then the reproduction operators can be described as follows.

(1) Reflection: Reflect the worst vertex X_w across the centroid of the other m points, i.e., the reflection point $X_r = \bar{X} + \alpha \cdot (\bar{X} - X_w)$, where α is a predetermined scaling factor, $\alpha \in [0.5,\ 2]$, and the centroid $\bar{X} = \frac{1}{m} \sum_{k \neq w} X_k$.
(2) Contraction: Draw back the reflection point X_r towards the worst vertex X_w, i.e., the contraction point $X_c = \bar{X} + \beta \cdot (X_w - \bar{X})$, where β is another predetermined scaling factor, and $\beta \in [-0.5, -0.1] \cup [0.1,\ 0.5]$.
(3) Local learning: Move the current point X_i towards the best vertex X_b or backwards the worst vertex X_w, i.e., the new individual $X_l = X_i + 0.618 \cdot (X_b - X_i)$ or $X_l = X_i + 0.382 \cdot (X_i - X_w)$.

Note that the local learning could be regarded as a 1-dimensional simplex operator.

3.2 m-Simplex evolution

In this section, we will first introduce a general form of simplex evolution so called m-simplex evolution (m-SE), where $m = 1, 2, \ldots, n$. Similar to other population-set-based algorithms, m-SE maintains a population set $\mathbf{X}(t)$ of N individuals/points $X_i(t)$, $i = 1, 2, \ldots, N$, during the evolutionary progress. The evolutionary progress is to guide the individuals in the current population in stepping towards (hopefully) better positions. The individuals will hopefully get improved from generation to generation, and converge to the global minimizer in the long run.

In m-SE, the so-called better position is determined by the current individual $X_i(t)$ and the other $m+1$ randomly selected individuals from the current population. The $m+1$ individuals make up of an m-simplex. In this sense, m-SE can be regarded as a kind of multi-parent genetic algorithm (with $m+2$ parents in total). We construct a new simplex for each position i in the current population. In other words, the new m-simplex is real-time constructed and it has almost nothing to do with the old ones. This is quite different from the original Nelder-Mead method, in which the new simplex is always a transformation of the old one. As a result, m-SE has no reduction operator.

The new individual (the entity of a hopefully better position) will replace the current individual $X_i(t)$ if only it is better (except for the test step). Here the *better* means it is better than the current individual, not necessarily/sufficiently better than the worst vertex X_w. Therefore, whether a simplex operator (reflection or contraction) is successful depends on the value of

$X_i(t)$, not X_w. This is another difference from the original Nelder-Mead method. As a result, m-SE has no expansion operator.

In general, the current individual $X_i(t)$ tries to improve himself in a framework of try-try-test. In the beginning it has two chances. The first chance is provided by the reflection operator. 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 $X_i(t)$ get improved. Otherwise a second chance, the contraction, will be carried out. Similarly, $X_i(t)$ will get improved if the contraction point X_c is better. However, if the i-th individual $X_i(t)$ has lost the previous two chances and cannot make the average profit (that is, its function value is greater than or equal to the average value of the current population), it will take a chance on local learning. This step is motivated, in some sense, by the behavior of a human being. Obviously, the i-th individual tries the local learning to improve himself at the risk of being degenerated. Although local learning might degenerates the i-th individual, it could increase the diversity of the population.

From the above discussion, we can see that (1) m-SE employs the simplex operators of Nelder-Mead method selectively (e.g., the expansion and reduction operators are discarded) after dimension reduction; (2) m-SE employs a new simplex operator — the local learning; and (3) the three evolutionary operators are applied conditionally, e.g., the contraction will not be applied if the reflection is successful, and the local learning will not be applied if the reflection or contraction is successful. The procedure of the m-SE can be outlined as follows.

Procedure *m*-simplex evolution algorithm:

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

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

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

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

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) Test local learning: If $f(X_i(t)) \geq \frac{1}{N} \sum_{k=1}^{N} f(X_k(t))$ then compute the new point

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

Let $X_i(t + 1) = X_l$, 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.

The stopping criteria will be described in Sect. 5. Note that the population size N is usually taken much larger than the dimension of the problem n to ensure the global convergence. Another noteworthy fact is that the m-SE should be non-generational (that is, the offspring individuals could be selected as parent individuals as soon as they are generated) to decrease the required memory (storage) space for programming.

3.3 Full and low dimensional simplex evolution

The m-simplex evolution described in the above subsection is called full dimensional simplex evolution (FDSE) if the dimension of the simplex therein is equal to that of the problem (that is, $m = n$). Accordingly, it is called lower dimensional simplex evolution if $m < n$. Particularly, the m-simplex evolution is called low dimensional simplex evolution (LDSE) if the dimension of the simplex is much less than that of the problem (that is, $m \ll n$). The LDSE is the one we highly recommend. Usually we take $m \leq 5$ for the problems which dimension $n \leq 50$. Numerical study indicates that LDSE is much superior to FDSE for the problems with a higher dimension such as $n \geq 10$ (see subsection 5.1).

In summary, LDSE is a hybrid evolutionary algorithm of (<ET>)-type (see Sect. 2 and subsection 3.2). It generates new trial points following the Nelder-Mead algorithm, and the individuals survive by the rule of natural selection. However, the simplices therein are real-time constructed and low dimensional. The simplex operators are applied selectively and conditionally. Each individual is updated in a framework of try-try-test.

If we take $m = 2$ and fix the scaling factors $\alpha = 1$, $\beta = \frac{1}{3}$, LDSE will be fully simplified, and the resulting algorithm will require only one (explicit) control parameter, the population size N. This renders it very easy to use. The simplest is hopefully the best if the simplification does not affect too much of its efficiency. The simplified LDSE is called triangle evolution (TE) because in this case the simplex is indeed a triangle. TE is the simplest form of LDSE. In order to ensure its efficiency, TE requires that the three randomly selected parents are mutually different.

4 Constraint handling

Constraint handling is another challenging topic in evolutionary algorithms. Carlos A. Coello Coello provides a comprehensive list of references on constraint handling techniques used with evolutionary algorithms and keeps constantly updating [2]. We will not discuss general constraint-handling techniques in this paper. For the simple box-constraints, they are handled quite straightforward. Once the component x_j of a new trial point goes beyond its bounds (i.e., $x_j < l_j$ or $x_j > u_j$, the case might happen at the reflection and the local learning steps), it will be initialized to $x_j = l_j + r \cdot (u_j - l_j)$, where r is a random real number with uniform distribution on the interval [0, 1]. In our practical application of LDSE, nonlinear constraints are handled using a technique similar to the direct constraint handling method proposed by Price, Storn and Lampinen [23].

5 Numerical results

Performances of the proposed algorithms are tested with a set of 50 continuous global optimization problems (P) from [1]. All the problems are box-constrained. The dimensions range

Table 1 Comparison of FDSE and LDSE using 10 problems. The hyphen "-" means an algorithm has failed to find the global minimum within the given number of function evaluations E in all the 100 runs

Problems		N	FDSE			LDSE			Reduced	Increased
P	n		m	nfe	ps	m	nfe	ps	$nfe(\%)$	ps
ACK	10	30	10	7317	48	4	2859	100	60.9	52
ACK	20	30	20	10316	42	4	3516	100	65.9	58
EXP	10	20	10	887	100	4	733	100	17.4	0
EXP	20	30	20	2030	100	4	1328	100	34.6	0
GW	10	20	10	1898	87	4	1413	100	25.6	13
GW	20	30	20	3557	100	4	2215	100	37.7	0
LM2	10	150	10	19167	21	3	9703	92	49.4	71
LM2	20	400	20	–	0	2	66417	96	–	96
RG	10	20	10	4152	35	2	1316	100	68.3	65
RG	20	40	20	5433	26	2	3281	100	39.6	74
Average									44.4	42.9

from 2 to 20. They have a variety of inherent difficulties, and many of them are frequently used to test the performance of evolutionary algorithms.

We use the same stopping criteria for each algorithm and test case in our numerical experiments. The algorithm will stop if it meets one of the following conditions:

(1) the global minimum is attained in the numerical sense, i.e., $f_{GloBest}(t) - f^* < \varepsilon$, where $f_{GloBest}(t)$ is the function value of best-so-far individual, f^* is the best known function value of the GO problem, and the tolerance $\varepsilon = 10^{-6}$.
(2) the population is matured in the numerical sense, i.e., $f_{PopWorst}(t) - f_{PopBest}(t) < \delta$, $f_{PopWorst}(t)$, where $f_{PopBest}(t)$ are the function values of the worst and the best individual in current population respectively, and $\delta = 10^{-4}$.
(3) the maximum number of function evaluations E is reached, where $E = 500n^3$.

In order to reduce the effect of randomness, each test case is executed 100 times with different initial populations (by setting different random seeds). The average number of function evaluations (nfe) and the percentage of success (ps) are reported to show the performance of each algorithm. An execution is said to be successful if and only if the first stopping criterion (as above listed) is satisfied.

5.1 Comparison of LDSE and FDSE

The effect of the dimension reduction on the m-simplex evolution algorithm is demonstrated with a group of 10 dimension-adjustable problems selected from the testbed. Numerical results show that LDSE is superior to FDSE by 44.44% and 42.9 in terms of average reduced nfe and average increased ps respectively (see Table 1), where the reduced $nfe = (1 - \frac{nfe_{\text{LDSE}}}{nfe_{\text{FDSE}}}) \cdot 100\%$, and the increased $ps = ps_{\text{LDSE}} - ps_{\text{FDSE}}$.

5.2 Comparison of LDSE and DE

In this subsection, LDSE is applied to all 50 problems in the testbed to verify its performance and compare with an improved version of differential evolution (DE). We choose the DE for comparison because it has proven to be the fastest evolutionary algorithm among the 1st ICEO conference entries.Differential evolution (DE) was first proposed by R. Storn and K. Price [26]. It is also a population set based algorithm. For each individual $X_i(t)$ (a point in $\mathbb{R}^n$) in the current population, DE reproduces a new trial point by mutation and crossover. Consider the most popular strategy DE/rand/1/bin. In the mutation phase DE randomly selects three distinct points $X_{r_1}(t)$, $X_{r_2}(t)$, $X_{r_3}(t)$ from the current population set $\mathbf{X}(t)$. The mutated point $X_m(t)$ is a permutation of one point along the differential variation of the other two, that is, $X_m(t) = X_{r_1}(t) + F \cdot (X_{r_2}(t) - X_{r_3}(t))$, where F is a scaling factor. In the crossover phase, the trial point is generated by the crossover between $X_m(t)$ and $X_i(t)$ of the form

$$x_{trial,\,j}(t) = \begin{cases} x_{m,j}(t), & \text{if rand}[0,1) < C_R \text{ or } j = j_{\text{rand}}; \\ x_{i,j}(t), & \text{otherwise.} \end{cases}$$

The trial point replaces $X_i(t)$ from the population if and only if it is not worse than it. In 2006, P. Kaelo and M. M. Ali proposed a modified version of DE, called differential evolution algorithm with random localization (DERL) [11]. Different from the original DE, the modified DERL requires $X_{r_1}(t)$ to be the best of the three random points and uses a random scaling factor F uniformly drawn from $[-1, -0.4] \cup [0.4, 1]$. DERL has improved the performance of DE significantly [11].

In order to make a fair comparison between LDSE and DE, we choose TE (an LDSE algorithm with a fixed dimension and fixed scaling factors, see subsection. 3.3) as the representative to compete with DERL (an improved version of DE).

Note that TE and DERL share similar method for parent selection. Both algorithms reproduce new individuals from the i-th individual and three randomly selected mutually different parents. In addition, both TE and DERL use the function-value information of selected parents, that is, TE needs to find the best and the worst parents to carry out the simplex operations and DERL also needs to find the best parent to carry out the mutation operation.

The control parameter C_R of DERL used in our experiment follows [11], in which $C_R = 0.5$ is reported to be the best choice for the 50 test problems. The population size N for TE and DERL are all found empirically. We have conducted a series of runs of TE and DERL by varying N from $1.5n$ to $20n$. Only the best results are presented in this paper. The population size N used here might not be optimal, but it should be a good choice at least. The comparison results between DERL and TE are presented in Table 2, where the reduced $nfe = (1 - \frac{nfe_{\text{TE}}}{nfe_{\text{DERL}}}) \cdot 100\%$, and the increased $ps = ps_{\text{DERL}} - ps_{\text{TE}}$. From Table 2, we can see that TE outperforms DERL considerably. TE is superior to DERL by 27.72% and 3.07 in terms of average reduced nfe and average increased ps.

6 Conclusion

We have presented a new hybrid evolutionary algorithm named low dimensional simplex evolution (LDSE) for continuous global optimization. LDSE is a (<ET>)-type hybrid algorithm (see Sect. 2 and subsection. 3.2). It generates new trial points by reflection and contraction operators derived from the Nelder-Mead algorithm after essential modifications, and the

Table 2 Comparison of DERL and TE using 50 problems

Problems			DERL		TE		Reduced	Increased
No.	P	n	nfe	ps	nfe	ps	$nfe(\%)$	ps
1	ACK	10	31146	100	2772	100	91.10	0
2	AP	2	425	100	281	100	33.88	0
3	BL	2	695	100	438	100	36.98	0
4	B1	2	687	100	346	100	49.64	0
5	B2	2	632	100	397	100	37.18	0
6	BR	2	761	100	354	100	53.48	0
7	CB3	2	473	100	266	100	43.76	0
8	CB6	2	494	100	290	100	41.30	0
9	CM	4	1706	100	894	100	47.60	0
10	DA	2	1037	100	543	100	47.64	0
11	EP	2	642	100	328	100	48.91	0
12	EM	10	15707	62	19713	71	−25.50	9
13	EXP	10	1428	100	661	100	53.71	0
14	GP	2	706	100	376	100	46.74	0
15	GW	10	7037	100	1304	100	81.47	0
16	GRP	3	1562	100	719	100	53.97	0
17	H3	3	867	100	449	100	48.21	0
18	H6	6	4353	91	2276	100	47.71	9
19	HV	3	941	100	686	100	27.10	0
20	HSK	2	326	100	194	100	40.49	0
21	KL	4	1244	100	538	100	56.75	0
22	LM1	3	745	100	422	100	43.36	0
23	LM2	10	9189	100	8741	100	4.88	0
24	MC	2	353	100	195	100	44.76	0
25	MR	3	3301	83	1278	100	61.28	17
26	MCP	4	1863	100	1495	94	19.75	−6
27	ML	10	150403	71	138264	100	8.07	29
28	MRP	2	985	65	523	82	46.90	17
29	MGP	2	1254	67	4204	89	−235.25	22
30	NF2	4	35874	100	17901	100	50.10	0
31	NF3	10	56206	100	18419	100	67.23	0
32	OSP	10	–	–	–	–	–	–
33	PP	10	10465	100	8229	100	21.37	0
34	PRD	2	1253	100	926	100	26.10	0
35	PQ	4	2592	100	1091	100	57.91	0
36	PTM	9	–	–	–	–	–	–
37	RG	10	83141	100	1236	100	98.51	0
38	RB	10	96417	100	14831	100	84.62	0
39	SAL	10	–	–	–	–	–	–

Table 2 continued

Problems			DERL		TE		Reduced	Increased
No.	P	n	nfe	ps	nfe	ps	$nfe(\%)$	ps
40	SF1	2	2891	53	1429	100	50.57	47
41	SF2	2	1598	100	1631	100	−2.07	0
42	SBT	2	1795	100	1346	100	25.01	0
43	SWF	10	18525	100	16045	97	13.39	−3
44	S5	4	4159	100	3754	100	9.74	0
45	S7	4	3720	100	3603	100	3.15	0
46	S10	4	4556	100	4230	100	7.16	0
47	FX	10	–	–	–	–	–	–
48	SIN	20	14608	100	60341	100	−313.07	0
49	ST	9	37864	100	12807	100	66.18	0
50	WP	4	6841	100	3176	100	53.57	0
Average							27.72	3.07

new individuals survive by the rule of natural selection in a framework of try-try-test. The simplices therein are real-time constructed and low dimensional. The simplex operators are applied selectively and conditionally. The expansion and reduction operators in the Nelder-Mead algorithm are discarded. A new simplex operator, the local learning, is introduced to increase the diversity of the population. The contraction will not be applied if the reflection is successful, and the local learning will not be applied if the reflection or contraction is successful.

Numerical investigation shows that LDSE is much more superior to full dimensional simplex evolution (FDSE) for the problems with a higher dimension such as $n \geq 10$ (see subsection 5.1). The performance of LDSE has been demonstrated with an extensive testbed of 50 test problems from the reference [1]. The comparison results show that LDSE outperforms an improved version of differential evolution (DE) considerably with respect to the convergence speed and reliability (see subsection 5.2).

Although the numerical results of LDSE are very encouraging, its working mechanism is not clear yet. LDSE needs a large population size N to ensure its global convergence. This might decelerate its convergence speed. Therefore, some more efficient variants of LDSE with less population size are expected in future work. In addition, some algorithm-specific constraint handling and multi-objective treatment techniques are also expected.

Acknowledgments The authors would like to thank the anonymous reviewers for their constructive comments related to earlier manuscript versions of this work.

References

1. Ali, M.M., Khompatraporn, C., Zabinsky, Z.B.: A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. J. Glob. Optim. **31**, 635–672 (2005)
2. Coello, C.A.C.: List of references on constraint-handling techniques used with evolutionary algorithms. http://www.cs.cinvestav.mx/~constraint (2010) Accessed 15 April 2010

3. Gutknecht, M.H.: A brief introduction to Krylov space methods for solving linear systems. In: Proceedings of Frontiers of Computational Science, pp. 53–62 (2007)
4. Hansenand, N., Ostermeier, A.: Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In: Proceedings of IEEE International Conference on Evolutionary Computation, pp. 312–317 (1996)
5. Hedar, A., Fukushima, M.: Minimizing multimodal functions by simplex coding genetic algorithm. Optim. Method Softw. **18**, 265–282 (2003)
6. Horst, R., Tuy, H.: Global Optimization: Deterministic Approaches, 3rd edn. Springer, Berlin (1996)
7. Huyer, W., Neumaier, A.: Global optimization by multilevel coordinate search. J. Glob. Optim. **14**, 331–355 (1999)
8. Ingber, L.: Simulated annealing: practice versus theory. J. Math. Comput. Modell. **18**, 29–57 (1993)
9. Jones, D.R., Perttunen, C.D., Stuckman, B.E.: Lipschitzian optimization without the Lipschitz constant. J. Opt. Theor. Appl. **79**, 157–181 (1993)
10. Jones, D.R.: Direct global optimization algorithm. In: Floudas, C.A., Pardalos, P.M. (eds.) Encyclopedia of Optimization, 2nd edn. Part 4, pp. 725–735 (2009)
11. Kaelo, P., Ali, M.M.: A numerical study of some modified differential evolution algorithms. Eur. J. Oper. Res. **169**, 1176–1181 (2006)
12. Kennedy, J., Eberhart R.: Particle swarm optimization. In: Proceedings of IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948. Piscataway, NJ (1995)
13. Nelder, J.A., Mead, R.: A simplex method for function minimization. Comput. J. **7**, 308–313 (1965)
14. Neumaier, A.: Complete search in continuous global optimization and constraint satisfaction. Acta Numerica **13**, 271–369 (2004)
15. Pardalos, P.M., Resende, M. (eds.): Handbook of Applied Optimization. Oxford University Press, New York (2002)
16. Pardalos, P.M., Romeijn, E. (eds.): Handbook of Global Optimization—Volume 2: Heuristic Approaches. Kluwer, Dordrecht (2002)
17. Pardalos, P.M., Mavridou, T.D.: Simulated annealing. In: Floudas, C.A., Pardalos, P.M. (eds.) Encyclopedia of Optimization, 2nd edn., Part 19, pp. 3591–3593 (2009)
18. Pintér, J.D.: Global Optimization in Action. Kluwer, Dordrecht (1996)
19. Pintér, J.D.: Computational Global Optimization in Nonlinear Systems: An Interactive Tutorial. Lionheart Publishing Inc., Atlanta (2001)
20. Pintér, J.D.: Nonlinear optimization with GAMS/LGO. J. Glob. Optim. **38**, 79–101 (2007)
21. Pintér, J.D.: Continuous global optimization: models, algorithms and software. In: Floudas, C.A., Pardalos, P.M. (eds.), Encyclopedia of Optimization, Part 3, 2nd edn, pp. 486–493 (2009)
22. Renders, J.M., Bersini, H.: Hybridizing genetic algorithms with hill-climbing methods for global optimization: two possible ways. In: Proceedings of the First IEEE Conference on Evolutionary Computation, pp. 312–317 (1994)
23. Price, K., Storn, R., Lampinen, J.: Differential Evolution—A Practical Approach to Global Optimization. Springer, Berlin (2005)
24. Sahinidis, N.V.: BARON: a general purpose global optimization software package. J. Glob. Optim. **8**, 201–205 (1996)
25. Spendley, W., Hext, G.R., Himsworth, F.R.: Sequential application of simplex designs in optimization and evolutionary operation. Technometrics **4**, 441–461 (1962)
26. Storn, R., Price, K.: Differential evolution—a simple and efficient heuristic for global optimization over continuous space. J. Glob. Optim. **11**, 341–359 (1997)
27. Tuy, H.: Cutting plane methods for global optimization. In: Floudas, C.A., Pardalos, P.M. (eds.) Encyclopedia of Optimization, 2nd edn., Part 3, pp. 590–594 (2009)
28. Zhigljavsky, A.A., Zilinskas, A.G.: Stochastic Global Optimization. Springer, Berlin (2008)