Hindawi Publishing Corporation
Discrete Dynamics in Nature and Society
Volume 2015, Article ID 303125, 9 pages
http://dx.doi.org/10.1155/2015/303125

Research Article

A DE-Based Scatter Search for Global Optimization Problems

Kun Li[1] and Huixin Tian[2]

[1]*School of Management, Tianjin Polytechnic University, Tianjin 300387, China*
[2]*School of Electrical Engineering & Automation, Tianjin Polytechnic University, Tianjin 300387, China*

Correspondence should be addressed to Kun Li; lk_neu@163.com

Received 19 September 2014; Accepted 29 January 2015

Academic Editor: Gualberto Solís-Perales

Copyright © 2015 K. Li and H. Tian. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This paper proposes a hybrid scatter search (SS) algorithm for continuous global optimization problems by incorporating the evolution mechanism of differential evolution (DE) into the reference set updated procedure of SS to act as the new solution generation method. This hybrid algorithm is called a DE-based SS (SSDE) algorithm. Since different kinds of mutation operators of DE have been proposed in the literature and they have shown different search abilities for different kinds of problems, four traditional mutation operators are adopted in the hybrid SSDE algorithm. To adaptively select the mutation operator that is most appropriate to the current problem, an adaptive mechanism for the candidate mutation operators is developed. In addition, to enhance the exploration ability of SSDE, a reinitialization method is adopted to create a new population and subsequently construct a new reference set whenever the search process of SSDE is trapped in local optimum. Computational experiments on benchmark problems show that the proposed SSDE is competitive or superior to some state-of-the-art algorithms in the literature.

1. Introduction

In recent years, many research efforts have been devoted to the hybridization of different algorithms because the adoption of the best features of individual algorithm can often result in a new powerful and efficient hybrid algorithm. When designing such hybrid algorithms, one should examine and evaluate the features and search mechanisms of different algorithms, as well as their potential combinations.

Scatter search (SS) was firstly proposed by Glover [1] for integer programming. It is a population-based evolutionary algorithm that combines a set of solutions to construct a new solution. The scatter search template (Glover [2]) provides the foundation for most implementations of SS by now. After population initialization, SS selects a subset of solutions from the population as the reference set and then operates on the reference set to generate new trial solutions by combining solutions in the reference set. With comparison to the other evolutionary algorithms such as genetic algorithm, the memory feature of the reference set can accurately grasp the current optimization state and reduce the frequency of using stochastic search, which in turn greatly improves the ability of global search. In addition, the diversity maintenance mechanism of SS guarantees the search diversity during evolution. Due to its flexibility and effectiveness, SS has been widely used in tackling combination optimization problems such as the graph-based permutation problems (Rego and Aleao [3]), flow shop scheduling problem (Yamada and Nakano [4], Nowicki and Smutnicki [5]), vehicle routing problem (Taillard [6]), quadratic assignment problem (Chung et al. [7]), and vehicle routing (Greistorfer [8], Russell and Chiang [9]). Although SS has achieved good performance in solving combination optimization problems, few literatures focus on the application of SS in continuous optimization problems. Herrera et al. [10] investigated the performance of continuous SS with two combination methods, namely, the *BLX-*α and the average combination. Ugray et al. [11] proposed a multistart framework for global optimization by embedding SS with a nonlinear programming solver.

Although differential evolution (DE) is a new member of the swarm-based evolutionary algorithms, it has become one of the most powerful stochastic search algorithms in the literature (Das and Suganthan [12]). Due to its simplicity and ease of implementation, DE has performed well in

(1) **Begin**
(2) Use the *Diversification generation method* to create a population.
(3) Use the *Improvement method* to enhance the population, and set S_b to be the best solution found so far.
(4) Use the *Reference update method* to build the initial reference set R.
(5) **while** (the termination criterion is not reached) **do**
(6) Generate subsets from R using the *Subset generation method*.
(7) Combine each subset into a new solution using the *Solution combination method*.
(8) Use the *Improvement method* to enhance each combined solution, and update S_b.
(9) Update the reference set based on the union of the current reference set and the new combined solutions using the *Reference update method*.
(10) **end while**
(11) Report the best solution S_b.
(12) **End**

Algorithm 1: Canonical scatter search.

many single-objective optimization problems (Zhang and Sanderson [13], Qin et al. [14]) and engineering optimization problems (Babu and Angira [15], Zhang and Rangaiah [16]). The main advantage of DE is that it enables the solution to learn from better solutions (e.g., the DE/best/1 mutation operator) and at the same time maintain the search diversity (e.g., the DE/rand/1 mutation operator).

Due to the power of SS and DE, some researchers start to investigate their hybridization to construct a new powerful algorithm. Davendra and Onwubolu [17] and Davendra et al. [18] proposed a hybrid DE with SS for discrete optimization problems such as quadratic assignment problem (QAP), traveling salesman problem (TSP), and permutation flow shop scheduling problem (PFSP). However, this hybrid algorithm is in fact a two-phase method, in which the first phase is to obtain a good solution by DE and the second phase is to improve this solution by SS. Shi et al. [19] developed a real hybrid SS with DE by embedding DE into SS to generate the new reference set. This hybrid algorithm is used to solve the discrete project scheduling problem.

In this paper, we propose a new hybrid algorithm based on SS and DE, named hybrid SS with DE (SSDE). The main framework follows that of SS but uses DE to combine solutions in the reference set and then generate new solutions. Different from that of Shi et al. [19], our hybrid algorithm is designed for continuous algorithm. In addition, our hybrid SSDE algorithm adopts multiple mutation operators proposed for classical DE and makes use of an adaptive mechanism to select the most appropriate mutation operators during evolution. This strategy makes the hybrid SSDE very robust for different kinds of optimization problems.

The remainder of this paper is organized as follows. Section 2 gives the background descriptions on SS and DE, and in Section 3 the procedure and components of the hybrid SSDE are described in details. The computational experiments are carried out in Section 4 and our SSDE is compared to the other state-of-the-art evolutionary algorithms in the literature on benchmark test problems. Finally, this paper is concluded in Section 5.

2. Background

2.1. Canonical Scatter Search. The framework of the SS generally includes five components (shown in Algorithm 1). The SS first generates a population of solutions using a diversification generation method that considers both solution quality and diversity. Then the reference update method is used to build and maintain the reference set by selecting some solutions with good quality and the other solutions with good diversity from the population (note that the size of the reference set is a preset fixed value). Based on the reference set, the subset generation method is used to produce a great number of subsets of solutions as a basis for creating combined solutions (e.g., all the two solutions and the three solutions selected from the reference set are generally taken as the subsets). For each subset, the solution combination method is used to transform the solutions in the subset into one combined solution. At last, the improvement method is applied to further improve each combined solution. If there is no new solution added to the reference set, the algorithm terminates; otherwise, the algorithm goes to the subset generation procedure and starts a new iteration.

2.2. Canonical Differential Evolution. DE algorithm is a simple, fast, and robust evolutionary algorithm. It starts with a population P of NP solutions, and at each generation G the population updates through three main consecutive steps: mutation, crossover, and selection. In the following, the ith solution in the population of generation G is denoted as $X_{i,G}$, $i = 1, 2, \ldots, NP$.

The mutation step first randomly selects three solutions $\{X_{r1,G}, X_{r2,G}, X_{r3,G}\}$ from the population P of generation G for the target individual $X_{i,G}$ and then generates the perturbed vector $V_{i,G}$ based on the three selected solutions as follows:

$$V_{i,G} = X_{r1,G} + F \times \left(X_{r2,G} - X_{r3,G}\right), \quad (1)$$

(1) **Begin**
(2) Set the selection probability of each crossover operator O_k to be the same, that is, $p_k = 1/K$, where K is the sum of all mutation operators.
(3) Set the iteration counter $G = 0$ and the $t = 0$.
(4) Use the *Diversification method* to generate a population P, evaluate each solution in P and then set the best solution found so far S_b to be the best one in P.
(5) **while** (the termination criterion is not reached) **do**
(6) Use the *Reference update method* to update reference set R.
(7) Set $G = G + 1$, and $i = 0$. Set the solution set P' to be empty.
(8) **if** ($t < t_{\max}$) **do**
(9) **while** (**i** < |**P**|) **do** // generate a new solution for each $X_{i,G}$ in P
(10) Set the successful count s_{kG} and failure count f_{kG} of the crossover operator O_k to be zero.
(11) (1) Mutation
(12) Randomly select a mutation operator (namely O_k) using the *Adaptive selection method*.
(13) Randomly select solutions from R according to the requirement of mutation operator O_k.
(14) Perform mutation operator O_k on the selected solutions to generate a new trial solution $V_{i,G}$.
(15) (2) Crossover
(16) Apply crossover operation based on (5) to obtain new solution $U_{i,G}$.
(17) (3) Selection
(18) Use (7) to select the new solution $X_{i,G+1}$. If $f(X_{i,G+1}) < f(X_{i,G})$, set $s_{kG} = s_{kG} + 1$; otherwise, set $f_{kG} = f_{kG} + 1$.
(19) Add s_{kG} and f_{kG} of each crossover operator O_k to the end of a list L whose length is l. If $G > l$, then remove the first node of this list so that only a maximum of l nodes are stored in list L.
(20) Update the selection probability p_k of each operator O_k according to the method described in Section 3.4.
(21) Set $i = i + 1$.
(22) **end while**
(23) **if** (the best solution in P is better than S_b) **do**
(24) Update S_b and set $t = 0$.
(25) **else do**
(26) Set $t = t + 1$.
(27) **end if**
(28) **else do**
(29) Sort the solutions in the P in the ascending order of the objective value, and then randomly reinitialize the latter half of solutions in P.
(30) Set $t = 0$.
(31) **end if**
(32) **end while**
(33) Report the best solution S_b.
(34) **End**

Algorithm 2: Hybrid scatter search with differential evolution (SSDE).

where $i = 1, 2, \ldots, NP$, $r1$, $r2$, $r3$ are three different numbers randomly selected from $\{1, 2, \ldots, NP\}$ and $F \in [0, 1]$ is the control parameter.

The crossover step generates the trial solution $U_{i,G} = (u_{1,i,G}, u_{2,i,G}, \ldots, u_{n,i,G})$ based on the perturbed vector $V_{i,G} = (v_{1,i,G}, v_{2,i,G}, \ldots, v_{n,i,G})$ and the target solution $X_{i,G} = (x_{1,i,G}, x_{2,i,G}, \ldots, x_{n,i,G})$ as follows:

$$u_{j,i,G} = \begin{cases} v_{j,i,G}, & \text{if } \text{rand}_j \le C_r \text{ or } j = j_{\text{rand}} \\ x_{j,i,G}, & \text{otherwise,} \end{cases} \quad (2)$$

where $j = 1, \ldots, n$, $\text{rand}_j \in [0, 1]$ and j_{rand} is a randomly chosen index from $\{1, \ldots, n\}$ to ensure that the trial solution $U_{i,G}$ does not duplicate the target solution $X_{i,G}$. $C_r \in [0, 1]$ is the crossover probability set by the user.

Once the trial solution is generated, the selection step compares it with its target solution. If the trial solution $U_{i,G}$ has an equal or lower objective function value than that of its target solution $X_{i,G}$, it replaces the target vector in the next generation; otherwise, the target solution retains its place in the population for at least one generation. That is, the solution in the next generation is selected according to the following rule:

$$X_{i,G+1} = \begin{cases} U_{i,G}, & \text{if } f(U_{i,G}) \le f(X_{i,G}) \\ X_{i,G}, & \text{otherwise.} \end{cases} \quad (3)$$

3. Proposed SSDE Algorithm

In this section, we first present the overall framework of our hybrid SSDE algorithm in Algorithm 2 and then describe each component of it in details in the following sections. In Algorithm 2, $|P|$ denotes the size of population.

3.1. Initial Population Generation Method. To obtain an initial population with good diversity, we developed a diversification method following the main ideas of [20]. The details of this method are as follows, in which D is the sum of dimensions and LB_j and UB_j are the lower bound and upper bound of dimension j. Based on this method, the diversification of initial population can be guaranteed.

Step 1. Divide the range of each dimension $[\mathrm{LB}_j, \mathrm{UP}_j]$ ($j = 1, \ldots, D$) into R equal subranges.

Step 2. From dimension $j = 1$ to $j = D$, do the following steps.

Step 2.1. Set $i = 1$ and the selection probability of each subrange r to be $p_r = 1/R$.

Step 2.2. Randomly select a subrange, namely, r, and then within this range generate a random value for x_{ij}.

Step 2.3. For the selected subrange r, set $p_r = p_r - 1/n$, and for the other unselected subranges set $p_k = p_k + 1/(n \times (R-1))$.

Step 2.4. Set $i = i + 1$. If $i > n$, $j = j + 1$ and then go to Step 2.1; otherwise, go to Step 2.2.

3.2. Reference Update Method. In canonical SS algorithm, the reference set R generally consists of two parts of solutions: one part is a set of solutions with good quality in objective function value and the other part is a set of solutions with good diversity. This is a good choice for discrete solutions; however, for continuous optimization problems such a setting will often deteriorate the search efficiency. If two solutions are far from each other in the solution space, then the offspring generated from them is often inferior to them. In our preliminary experiment, the results show that the adoption of this setting often results in a very slow convergence speed, which in turn makes the algorithm unable to reach a good solution within a given number of function evaluations. Therefore, in our hybrid SSDE, the reference set R consists of only the best b solutions in the population. At the end of each generation, the reference set R is updated with the best b solutions in the new population.

3.3. Mutation Operators Used in Hybrid SSDE. As mentioned above, several mutation operators proposed for DE are adopted in our hybrid SSDE algorithm. The definitions of these mutation operators are as follows. In these equations, X_{best} denotes the best solution found so far by the algorithm:

(1) DE/rand/1: $V_{i,G} = X_{r1,G} + F \times (X_{r2,G} - X_{r3,G})$;

(2) DE/best/1: $V_{i,G} = X_{\mathrm{best},G} + F \times (X_{r1,G} - X_{r2,G})$;

(3) DE/rand-to-best/1: $V_{i,G} = X_{i,G} + F \times (X_{\mathrm{best},G} - X_{i,G}) + F \times (X_{r1,G} - X_{r2,G})$;

(4) DE/best/2: $V_{i,G} = X_{\mathrm{best},G} + F \times (X_{r1,G} - X_{r2,G}) + F \times (X_{r3,G} - X_{r4,G})$.

3.4. Adaptive Selection Method for Mutation Operators. To select the mutation operator, we present an adaptive mechanism by introducing the concept of solution survival used in the JADE [13] and SaDE [14]. The success and failure memories in SaDE are also used in the hybrid SSDE. In our SSDE, the application of a selected mutation operator for solution $X_{i,G}$ is viewed as successful only if the new solution resulted from this operator ($X_{i,G+1}$ in Algorithm 2) outperforms $X_{i,G}$. A list L with a fixed length l (also called the learning period) is established to store the successful counter s_{kG} and unsuccessful counter f_{kG} of each crossover operator O_k. During the first l generations, the selection probability of each mutation operator is equal to $1/K$ (K is the sum of mutation operators). This warm-up period is used to train the algorithm to analyze the performance and suitability of each mutation operator for the current optimization problem. At the end of each generation, the number of s_{kG} and f_{kG} for each crossover operator is counted and then added to the end of the list L. Then from the generation $G = l+1$, the head node of list L will be removed so that the new record of s_{kG} and f_{kG} can be added to the end of L.

The selection probability of each mutation operator k at generation G (denoted as p_{kG}) is calculated as follows according to SaDE [14]:

$$p_{kG} = \frac{S_{kG}}{\sum_{k=1}^{K} S_{kG}},$$
$$\text{where } S_{kG} = \frac{\sum_{g=G-l}^{G-1} s_{kg}}{\sum_{g=G-l}^{G-1} \left(s_{kg} + f_{kg}\right) + 0.01}, \quad (4)$$
$$k = 1, \ldots, K; \quad G > l.$$

3.5. Reinitialization of Population. During the evolution process, the hybrid SSDE algorithm may be trapped in local optimal regions, which in turn will deteriorate the performance of the hybrid SSDE. To further improve the ability of getting out of local optimal regions, a reinitialization method is adopted. During the evolution, we will record the number of consecutive generations that the best solution found so far has not been updated. Whenever this number exceeds a limit, then we will reinitialize the population P as follows. First, sort the solutions in the population P in the ascending order of their objective function values. Second, delete the latter half of the population in which the solutions have worse objective function values. Third, randomly select two solutions from the left half of P and then perform the *SBX* operator [20] to generate two offspring solutions. We will repeat the third step for $|P|/2$ times to generate $|P|$ new solutions, and then we select the best $|P|/2$ solutions and add them to P.

The procedure of the *SBX* operator can be described as follows according to Deb and Agrawal [20]. For two solutions $X = (x_1, \ldots, x_D)$ and $Y = (y_1, \ldots, y_D)$, the *SBX* operator can result in two new solutions $Z_1 = (z_{1,1}, \ldots, z_{1,D})$ and $Z_2 = (z_{2,1}, \ldots, z_{2,D})$ as follows. First, generate a random uniform

Table 1: Definition of benchmark problems.

Test function	Solution space	Optimal value
$f_1(x) = \sum_{i=1}^{D} x_i^2$	$[-100, 100]^D$	0
$f_2(x) = \sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} \lvert x_i \rvert$	$[-10, 10]^D$	0
$f_3(x) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_j \right)^2$	$[-100, 100]^D$	0
$f_4(x) = \max \lvert x_i \rvert$	$[-100, 100]^D$	0
$f_5(x) = \frac{\pi}{D} \left\{ 10\sin^2(\pi y_1) + \sum_{i=1}^{D-1} (y_i - 1)^2 \left[1 + 10\sin^2(\pi y_{i+1}) \right] \right\} + (y_D - 1)^2 + \sum_{i=1}^{D} u(x_i, 10, 100, 4)$, $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & x_i > a \\ 0 & -a \le x_i \le a \\ k(-x_i - a)^m & x_i < -a \end{cases}$, $y_i = 1 + (x_i + 1)/4$	$[-50, 50]^D$	0
$f_6(x) = \sum_{i=1}^{D} (\lfloor x_i + 0.5 \rfloor)^2$	$[-100, 100]^D$	0
$f_7(x) = \sum_{i=1}^{D} i x_i^4 + \text{rand}[0, 1)$	$[-1.28, 1.28]^D$	0
$f_8(x) = \sum_{i=1}^{D} -x_i \sin\left(\sqrt{\lvert x_i \rvert} \right) + D \times 418.98288727243369$	$[-500, 500]^D$	0
$f_9(x) = \sum_{i=1}^{D} \left[x_i^2 - 10\cos(2\pi x_i) + 10 \right]$	$[-5.12, 5.12]^D$	0
$f_{10}(x) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2} \right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D} \cos 2\pi x_i \right) + 20 + e$	$[-32, 32]^D$	0

number $u \in [0, 1]$ and then generate a new random number β based on u and a distribution index η according to

$$\beta = \begin{cases} (2u)^{1/(1+\eta)}, & \text{if } u \le 0.5 \\ \left(\frac{1}{2(1-u)} \right)^{1/(1+\eta)}, & \text{otherwise.} \end{cases} \quad (5)$$

At last each dimension of the two new solutions is generated using the following equations: $z_{1,i} = 0.5[(1+\beta)x_i + (1-\beta)y_i]$ and $z_{2,i} = 0.5[(1-\beta)x_i + (1+\beta)y_i]$, $(i = 1, 2, \ldots, D)$.

4. Computational Experiments and Results

In this section, we start with the introduction of the experimental environment in Section 4.1 and then the test problems in Section 4.2. Finally, Section 4.5 is devoted to carrying out the comparative studies between our SSDE and the other state-of-the-art evolutionary algorithms.

4.1. Experimental Settings. The proposed SSDE algorithm is implemented in C++, and all the experiments are carried out on a personal computer with the Intel Core 2 Duo 2.8 GHz CPU and 4 GB memory.

The parameters are set as follows based on our preliminary experiments. The population size is set to be 100, the size of reference set R is set to be 35, the control parameter F is generated by $N(0.8, 0.3)$, the crossover probability Cr is generated by $N(0.3, 0.3)$, and the number of consecutive generations $t_{\max}$ is set to be 50.

4.2. Test Problems. To test the performance of the proposed SSDE algorithm, 10 benchmark problems are selected from the literature. The definition of these problems is given in Table 1, in which the last column is the optimal objective function value for each test problem. In our experiments, we use $D = 30$.

In the experiment, a total of 50 independent runs are performed for each test problem to collect the statistical performance of each algorithm. The independent *t-test* is used to show the statistical difference between different algorithms and the result for each problem is given in the last column of Tables 2, 3, and 4. In the last column denoted as Sig, the signal "+" denotes that the performance difference between our SSDE and the best one among the other algorithms in the table is significant with a confidence level of 95%, while

Table 2: Comparison results between SSDE algorithms with and without adaptive selection method.

Test functions	Gen	Mean (Std Dev)					Sig
		$SSDE_{rand1}$	$SSDE_{best1}$	$SSDE_{r\text{-}best}$	$SSDE_{best2}$	SSDE	
f_1	1500	$3.2e-18$ ($2.7e-18$)	$6.0e-31$ ($9.7e-27$)	$8.1e-31$ ($1.2e-32$)	$4.2e-27$ ($3.6e-24$)	**$5.96e-36$** (**$1.33e-36$**)	+
f_2	2000	$1.2e-15$ ($4.2e-16$)	**$2.4e-40$** (**$2.0e-40$**)	$2.1e-29$ ($1.5e-28$)	$1.1e-17$ ($1.6e-17$)	$3.77e-27$ ($1.33e-36$)	+
f_3	5000	6.5148 (12.8856)	$6.5e-8$ (12.16)	$6.3e-8$ ($1.2e-17$)	$1.7e-11$ ($1.2e-13$)	**$9.83e-12$** (**$2.18e-11$**)	+
f_4	5000	$5.2e-3$ ($3.8e-2$)	$4.2e-2$ ($8.1e-2$)	$2.7e-3$ ($1.4e-3$)	$4.1e-10$ ($1.2e-11$)	**$6.44e-13$** (**$2.67e-12$**)	+
f_5	1500	$5.3e-19$ ($5.1e-19$)	$2.0e-3$ ($1.2e-3$)	$1.8e-32$ ($1.9e-43$)	$3.2e-23$ ($2.4e-21$)	**$1.57e-32$** (**$2.75e-47$**)	+
f_6	1500	$2.6e-18$ ($1.8e-19$)	$1.3e-32$ ($3.0e-33$)	0 (0)	$2.7e-24$ ($3.3e-23$)	**$3.08e-35$** (**$3.08e-34$**)	+
f_7	3000	$2.7e-46$ ($3.1e-53$)	**0** (**0**)	**0** (**0**)	**0** (**0**)	**0** (**0**)	–
f_8	9000	**0** (**0**)	6.0237 (17.4291)	**0** (**0**)	1.1 (6.2506)	**0** (**0**)	–
f_9	5000	**0** (**0**)	3.2942 (2.1396)	$4.4e-2$ ($2.5e-1$)	**0** (**0**)	**0** (**0**)	–
f_{10}	2000	$1.3e-13$ ($1.1e-17$)	$1.2e-14$ ($4.1e-12$)	$1.3e-13$ ($2.3e-15$)	$7.8e-1$ (1.7688)	**$4.51e-15$** (**$3.55e-15$**)	+

Table 3: Comparison results between SS-SBX and SSDE.

Test functions	Gen	Mean (Std Dev)		Sig
		SS-SBX	SSDE	
f_1	1500	**0** (**0**)	$5.96e-36$ ($1.33e-36$)	–
f_2	2000	**0** (**0**)	$3.77e-27$ ($1.33e-36$)	–
f_3	5000	$9.74e+00$ ($1.5e+01$)	**$9.83e-12$** (**$2.18e-11$**)	+
f_4	5000	**0** (**0**)	$6.44e-13$ ($2.67e-12$)	+
f_5	1500	$3.31e-01$ ($3.09e-03$)	**$1.57e-32$** (**$2.75e-47$**)	+
f_6	1500	$1.21e-01$ ($1.18e-01$)	**$3.08e-35$** (**$3.08e-34$**)	+
f_7	3000	**0** (**0**)	**0** (**0**)	–
f_8	9000	$2.33e-01$ ($1.70e-00$)	**0** (**0**)	+
f_9	5000	**0** (**0**)	**0** (**0**)	–
f_{10}	2000	**$4.44e-16$** (**0**)	$7.51e-15$ ($3.55e-15$)	+

the signal "–" denotes that their performance difference is not significant.

4.3. Efficiency Analysis of Adaptive Selection Method. In this section, we first carried out an experiment to show the efficiency of the adaptive selection method. We compared our proposed SSDE algorithm with four SSDE algorithms each of which adopts only one mutation operator. These four algorithms are, respectively, denoted as $SSDE_{rand1}$, $SSDE_{best1}$, $SSDE_{r\text{-}best}$, and $SSDE_{best2}$ according to the four mutation operators.

The comparison results for the proposed SSDE algorithm and the other SSDE with only one mutation operator are presented in Table 2, in which a better result is shown in the bold type. The second column *Gen* denotes the generations used in the algorithm as the stopping criterion, and *Mean* (*Std Dev*) represents the average value and the standard deviation obtained by each algorithm for each test problem.

From Table 2, it can be seen that the SSDE algorithm with the adaptive selection method can obtain the best results and show the best robustness for almost all the test problems. It is significantly better than the other four algorithms for 9 out of the 10 test problems. The reason behind this phenomenon is that this strategy can adaptively select the most appropriate mutation operators for the current problem. In addition, the adoption of multiple kinds of mutation operators also help to improve the exploration ability of SSDE because such a strategy can help to enhance the algorithm's ability of getting out of local optimal regions.

4.4. Efficiency Analysis of Hybrid Architecture of SSDE. To test the impact of adopting DE as the generation method

Table 4: Comparison results between SSDE and the other state-of-the-art algorithms.

Test functions	Gen	Mean (Std Dev)			Sig
		SaDE	JADE	SSDE	
f_1	1500	$4.5e-20$ ($6.9e-20$)	**$1.3e-54$** (**$9.2e-54$**)	$5.96e-36$ ($1.33e-36$)	+
f_2	2000	$1.9e-14$ ($1.05e-14$)	$3.9e-22$ ($2.7e-21$)	**$3.77e-27$** (**$1.33e-36$**)	+
f_3	5000	$9.0e-37$ ($5.43e-36$)	**$6.0e-87$** (**$1.9e-86$**)	$9.83e-12$ ($2.18e-11$)	+
f_4	5000	$7.4e-11$ ($1.82e-10$)	**$4.3e-66$** (**$1.2e-65$**)	$6.44e-13$ ($2.67e-12$)	+
f_5	1500	$1.2e-19$ ($2.0e-19$)	$1.6e-32$ ($5.5e-48$)	**$1.57e-32$** (**$2.75e-47$**)	−
f_6	1500	**0** (**0**)	**0** (**0**)	$3.08e-35$ ($3.08e-34$)	+
f_7	3000	$4.8e-03$ ($1.2e-03$)	$6.8e-04$ ($2.5e-04$)	**0** (**0**)	+
f_8	9000	$4.7e+00$ ($3.3e+01$)	$7.1e+00$ ($2.8e+01$)	**0** (**0**)	+
f_9	5000	**0** (**0**)	**0** (**0**)	**0** (**0**)	−
f_{10}	2000	$4.3e-14$ ($2.6e-14$)	**$4.4e-15$** (**0**)	$4.51e-15$ ($3.55e-15$)	−

for new solutions, in this section we implemented another kind of SS in which the new solutions were generated by the simulated binary crossover (*SBX*). That is, lines from 10 to 20 in Algorithm 2 are replaced by the following steps: first, randomly select two solutions from the reference R; second, apply the *SBX* operator to the two solutions to generate two new solutions; third, select the best one as the new solution $X_{i,G+1}$. This algorithm is denoted as *SS-SBX*.

The comparison results for the two kinds of SS algorithms are presented in Table 3, in which a better result is shown in the bold type. From this table, it can be seen that although *SS-SBX* obtains the best results for f_1, f_2, f_4, f_7, f_9, and f_{10}, for the other problems its performance is much worse. The SSDE algorithm obtains the best results for 6 out of the 10 problems, and its performance for the other four problems is also very good. Therefore, it can be concluded that the SSDE algorithm is superior the *SS-SBX*, especially with respect to the robustness.

4.5. Comparisons with Other State-of-the-Art Algorithms. In the above sections, we have shown that the adaptive selection mechanism based on multiple mutation operators and the adoption of DE into SS can help to improve both the search performance and robustness of SS. In this section, we further compare our SSDE with the other state-of-the-art evolutionary algorithms in the literature. These algorithms include the SaDE proposed in [14] and the JADE with archive proposed in [13]. Both the evolutionary algorithms have adaptive strategies in parameter control or mutation strategy selection during the evolution process. The comparison results are given in Table 4.

From Table 4, it can be found that our SSDE obtains better results for 7 out of the 10 problems with comparison to the SaDE algorithm in [14] (the performance differences are significant for all of the 7 problems) and for 4 out of the 10 problems with comparison to the JADE algorithm in [13]. For problems f_7 and f_8, our SSDE algorithm can reach the optimal solutions, while the other two algorithms SaDE and JADE show a much worse performance. In addition, it also appears that our SSDE algorithm is more robust for different kinds of problems. On average, it can be concluded that our SSDE is competitive to the JADE algorithm but superior to the SaDE algorithm for the 10 benchmark test problems.

To give a graphical illustration of this analysis, we reimplemented the SaDE algorithm and compared the evolution processes between our SSDE and the SaDE for the first six problems. The comparison results are shown in Figure 1, in which the ordinate is $\log_{10}(f(X))$ in which $f(X)$ is the objective function value of the best solution found in each iteration. Please note that whenever the algorithm reaches the optimal solution, the evolution process line will terminate in the figures because the optimal objective function value of each problem is zero. From this figure, it can be seen that the convergence speed of our SSDE is much faster than that of the SaDE for all the six test problems.

5. Conclusions

In this paper, we developed a hybrid scatter search algorithm by incorporating the differential evolution algorithm to act as the new solution generation method. To make the proposed hybrid algorithm more robust for different kinds of optimization problems, an adaptive selection mechanism

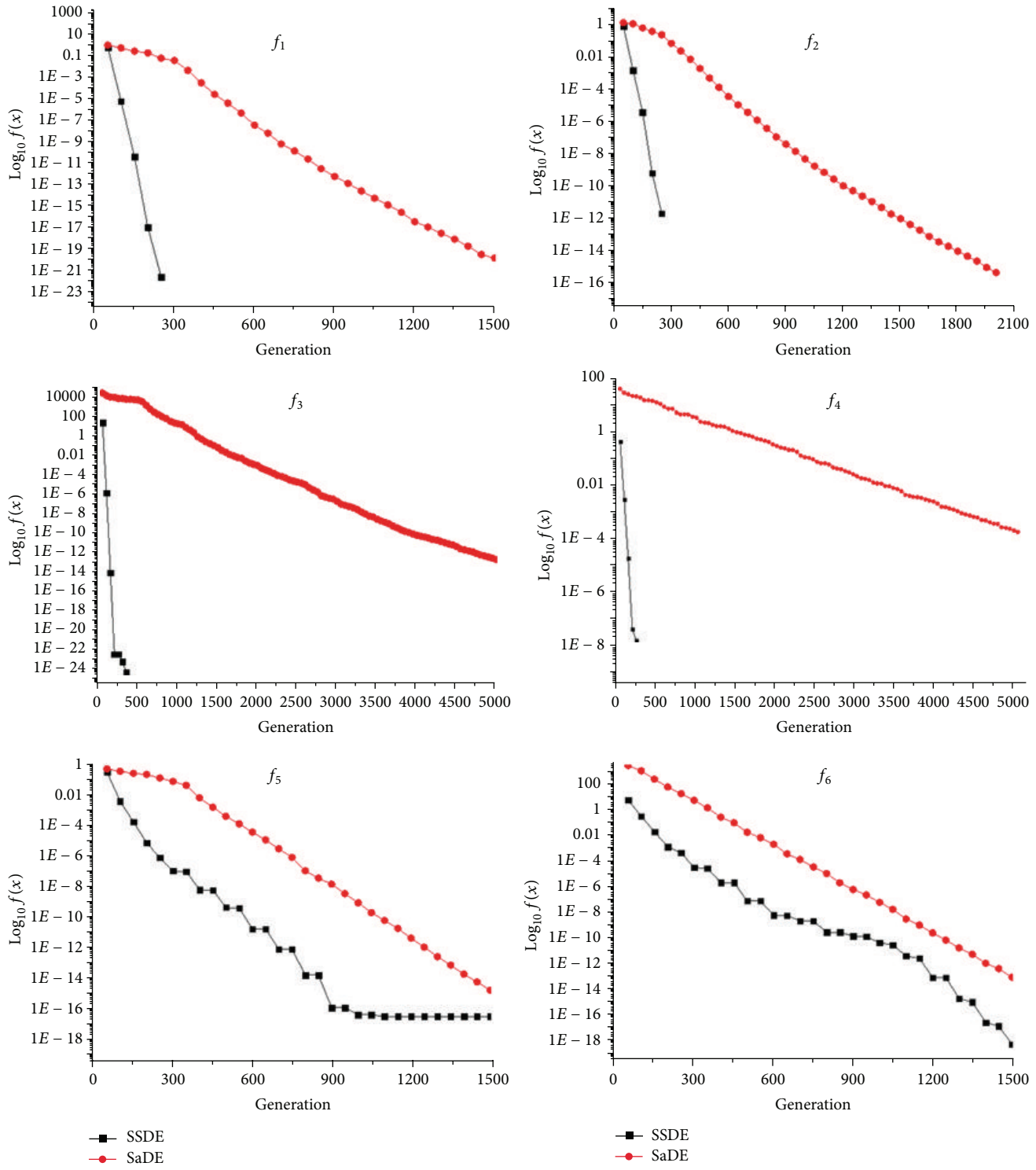


Figure 1: Evolution process of SSDE and SaDE for functions f_1–f_6.

is developed for multiple mutation strategies. A population reinitialization method is also used to help the algorithm to get out from local optimums. The computational results based on benchmark test problems show that the proposed hybrid SSDE algorithm is very efficient and robust and that its performance is competitive or even superior to two state-of-the-art evolutionary algorithms in the literature.

Conflict of Interests

The authors declare that this paper has no conflict of interests.

Acknowledgments

This research was partly supported by the National Natural Science Foundation of China (Grant no. 61403277) and the Humanities and Social Science Research Projects for Colleges in Tianjin (Grant no. 20132151).

References

[1] F. Glover, "Heuristics for integer programming using surrogate constraints," *Decision Sciences*, vol. 8, no. 1, pp. 156–166, 1977.

[2] F. Glover, "A template for scatter search and path relinking," in *Artificial Evolution*, vol. 1363 of *Lecture Notes in Computer Science*, pp. 1–51, Springer, Berlin, Germany, 1998.

[3] C. Rego and P. Aleao, "A scatter search tutorial for graph-based permutation problems," in *Metaheuristic Optimization via Memory and Evolution: Tabu Search and Scatter Search*, vol. 30, pp. 1–24, Kluwer Academic, Norwell, Mass, USA, 2005.

[4] T. Yamada and R. Nakano, "Scheduling by genetic local search with multi-step crossover," in *Parallel Problem Solving from Nature—PPSN IV: International Conference on Evolutionary Computation—The 4th International Conference on Parallel Problem Solving from Nature Berlin, Germany, September 22–26, 1996 Proceedings*, vol. 1141 of *Lecture Notes in Computer Science*, pp. 960–969, Springer, Berlin, Germany, 1996.

[5] E. Nowicki and C. Smutnicki, "Some aspects of scatter search in the flow-shop problem," *European Journal of Operational Research*, vol. 169, no. 2, pp. 654–666, 2006.

[6] E. D. Taillard, "A heuristic column generation method for the heterogeneous fleet VRP," CRT-96-03, Centre de recherché sur les transports, Université de Montréal, 1996.

[7] V. Chung, T. Mautor, P. Michelon, and A. Tavares, "Scatter search based approach for the quadratic assignment problem," in *Proceedings of the IEEE International Conference on Evolutionary Computation (ICEC '97)*, pp. 165–169, Indianapolis, Ind, USA, April 1997.

[8] P. Greistorfer, "A tabu scatter search metaheuristic for the arc routing problem," *Computers & Industrial Engineering*, vol. 44, no. 2, pp. 249–266, 2003.

[9] R. A. Russell and W. C. Chiang, "Scatter search for the vehicle routing problem with time windows," *European Journal of Operational Research*, vol. 169, no. 2, pp. 606–622, 2006.

[10] F. Herrera, M. Lozano, and D. Molina, "Continuous scatter search: an analysis of the integration of some combination methods and improvement strategies," *European Journal of Operational Research*, vol. 169, no. 2, pp. 450–476, 2006.

[11] Z. Ugray, L. Lasdon, J. Plummer, F. Glover, J. Kelly, and R. Marti, "Scatter search and local NLP solvers: a multistart framework for global optimization," *INFORMS Journal on Computing*, vol. 19, no. 3, pp. 328–340, 2007.

[12] S. Das and P. N. Suganthan, "Differential evolution: a survey of the state-of-the-art," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, 2011.

[13] J. Q. Zhang and A. C. Sanderson, "JADE: adaptive differential evolution with optional external archive," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 5, pp. 945–958, 2009.

[14] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 398–417, 2009.

[15] B. V. Babu and R. Angira, "Modified differential evolution (MDE) for optimization of non-linear chemical processes," *Computers and Chemical Engineering*, vol. 30, no. 6-7, pp. 989–1002, 2006.

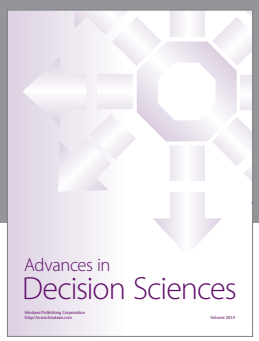
[16] H. B. Zhang and G. P. Rangaiah, "An efficient constraint handling method with integrated differential evolution for numerical and engineering optimization," *Computers & Chemical Engineering*, vol. 37, pp. 74–88, 2012.

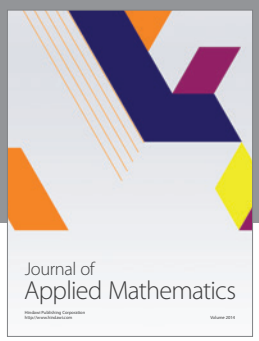
[17] D. Davendra and G. Onwubolu, "Enhanced differential evolution hybrid scatter search for discrete optimization," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '07)*, pp. 1156–1162, Singapore, September 2007.

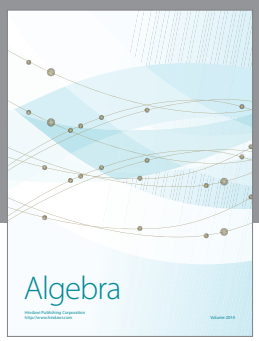
[18] D. Davendra, I. Zelinka, and G. Onwublu, "Hybrid differential evolution—scatter search algorithm for permutative optimization," in *Evolutionary Computation*, pp. 273–296, InTech, Vienna, Austria, 2009.

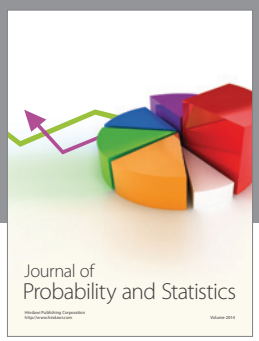
[19] Y. J. Shi, F. Z. Qu, W. Chen, and B. Li, "A differential evolution with scatter search for project scheduling," *Applied Mechanics and Materials*, vol. 26-28, pp. 724–727, 2010.

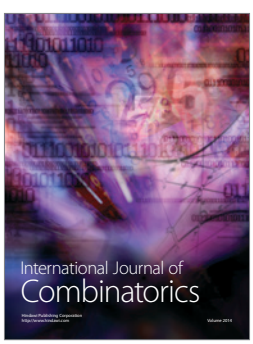
[20] K. Deb and R. B. Agrawal, "Simulated binary crossover for continuous search space," *Complex Systems*, vol. 9, no. 2, pp. 115–148, 1995.

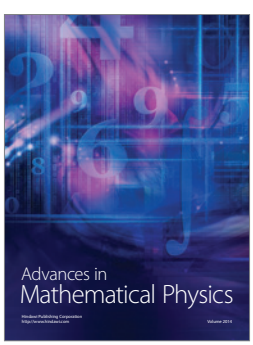


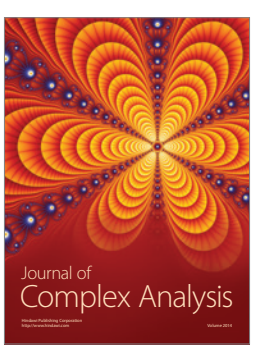


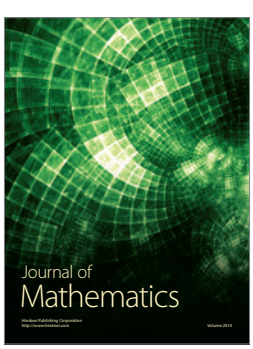


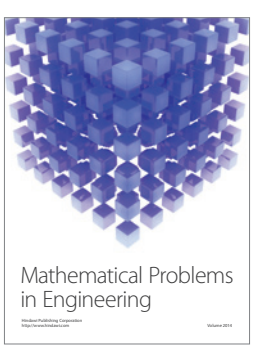


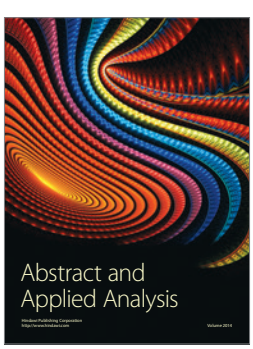


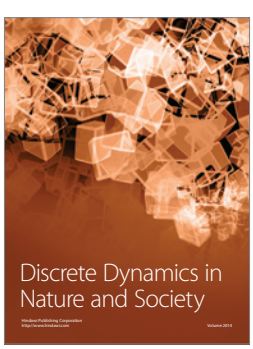


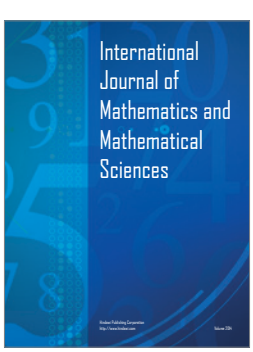












Hindawi

Submit your manuscripts at
http://www.hindawi.com