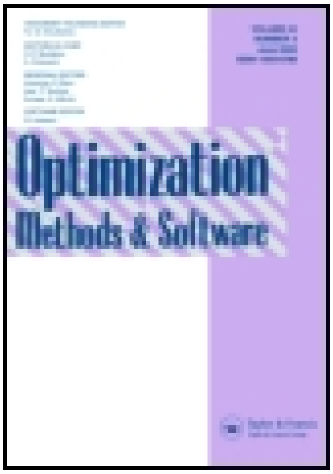

Optimization Methods and Software
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/goms20

Hybrid simulated annealing and direct search method for nonlinear unconstrained global optimization

Abdel-Rahman Hedar [a] & Masao Fukushima [a]
[a] Department of Applied Mathematics and Physics, Graduate School of Informatics , Kyoto University , Kyoto, 606-8501, Japan
Published online: 27 Oct 2010.

To cite this article: Abdel-Rahman Hedar & Masao Fukushima (2002) Hybrid simulated annealing and direct search method for nonlinear unconstrained global optimization, Optimization Methods and Software, 17:5, 891-912, DOI: 10.1080/1055678021000030084

To link to this article: http://dx.doi.org/10.1080/1055678021000030084

Optimization Methods and Software, Vol. 17, pp. 891–912

HYBRID SIMULATED ANNEALING AND DIRECT SEARCH METHOD FOR NONLINEAR UNCONSTRAINED GLOBAL OPTIMIZATION

ABDEL-RAHMAN HEDAR and MASAO FUKUSHIMA*

Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan

(Received 11 December 2001; In final form 12 August 2002)

In this article we give a new approach of hybrid direct search methods with meta-heuristics of simulated annealing for finding a global minimum of a nonlinear function with continuous variables. First, we suggest a Simple Direct Search (SDS) method, which comes from some ideas of other well-known direct search methods. Since our goal is to find global minima and the SDS method is still a local search method, we hybridize it with the standard simulated annealing to design a new method, called Simplex Simulated Annealing (SSA) method, which is expected to have some ability to look for a global minimum. To obtain faster convergence, we first accelerate the cooling schedule in SSA, and in the final stage, we apply Kelley's modification of the Nelder–Mead method on the best solutions found by the accelerated SSA method to improve the final results. We refer to this last method as the Direct Search Simulated Annealing (DSSA) method. The performance of SSA and DSSA is reported through extensive numerical experiments on some well-known functions. Comparing their performance with that of other meta-heuristics shows that SSA and DSSA are promising in practice. Especially, DSSA is shown to be very efficient and robust.

Keywords: Nonlinear unconstrained global optimization; Direct search methods; Meta-heuristics; Simulated annealing

1 INTRODUCTION

In recent years, there has been a great deal of interest in the development of optimization algorithms that deal with the problem of finding

*Corresponding author. Tel.: +81-75-753-5519. Fax: +81-75-753-4756. E-mail: fuku@i.kyoto-u.ac.jp

ISSN 1055-6788 print: ISSN 1029-4937 online © 2002 Taylor & Francis Ltd
DOI: 10.1080/1055678021000030084

a global minimum of a given continuous function [9,10,19,20]. These algorithms were innovated to confront the rapid growth of many optimization problems in science, economics and engineering. In this article, we will focus on the case of unconstrained minimization, i.e., the problem is

$$\min_{x \in R^n} f(x),$$

where f is a real valued function defined on R^n.

Meta-heuristics methods are considered to be acceptably good solvers of this problem [18]. The power of meta-heuristic methods comes from the fact that they are robust and can deal successfully with a wide range of problem areas. However, these methods, especially when they are applied to complex problems, suffer from the high computational cost due to their slow convergence. The main reason for this slow convergence is that these methods explore the global search space by creating random movements without using much local information about promising search direction [18]. In contrast, local search methods have faster convergence due to their using local information to determine the most promising search direction by creating logical movements. However, local search methods can easily be entrapped in local minima.

One approach that recently has drawn much attention is to combine meta-heuristic methods with local search methods to design more efficient methods with relatively faster convergence than the pure meta-heuristic methods. Moreover, these hybrid methods are not easily entrapped in local minima because they still maintain the merits of the meta-heuristic methods. Direct search methods, as local search methods, have got much attention in these combinations. For instance, Nelder–Mead method [17] was hybridized with genetic algorithm in [6,25] and with Tabu search in [7]. In addition, Kvasnicka and Pospichal [14] proposed a hybrid of controlled random search method, which is a generalization of the Nelder–Mead method, and simulated annealing (SA). Moreover, Tabu search was combined with another direct search method called Hooke–Jeeves method in [1].

In this article, we will hybridize SA, as a meta-heuristic method, and direct search methods, as local search methods. First, we suggest a simple direct search (SDS) method, which comes from some ideas of other well-known direct search methods. Since our goal is to find global minima and the SDS method is still a local search method, we hybridize it with the standard simulated annealing to design a new method, called simplex simulated annealing (SSA) method, which is expected to have some ability to look for global minima. The final method, called the direct search simulated annealing (DSSA) method, can be obtained by modifying SSA. To obtain faster convergence, we first accelerate the cooling schedule in SSA, and in the final stage, we apply Kelley's modification of the Nelder–Mead method [11,12] on the best solutions found by the accelerated SSA to improve the final results. These two modifications on SSA will comprise the final method DSSA. The performance of SSA and DSSA is reported through extensive numerical experiments on some well-known functions. Comparing their performance with that of other meta-heuristics methods shows that SSA and DSSA are promising in practice. Especially, DSSA is shown to be very efficient and robust.

To the authors' knowledge, there are two main previous results on hybridizing SA with simplex methods. Press and Teukolsky [21] add a positive logarithmically distributed variable, proportional to the control annealing temperature *T*, to the function associated with every vertex of the simplex. Likewise, they subtract a similar random variable from the function value at every new replacement point. Then, their method may accept a new simplex whose actual function values at its vertices are not better than those at the previous simplex. This method was subsequently studied by Cardoso *et al.* [3,4]. The other main results was presented by Kvasnicka and Pospichal [14]. Their method depends on the use of the SA acceptance in a controlled random search method. More precisely, the controlled random search uses a simplex method on randomly selected simplex sets from the population. So, to avoid being entrapped in local minima, they applied SA acceptance on the updating procedure. The common idea underlying these hybrid approaches and also our approach is to use simplex method to generate new logical movements while applying SA. However, the approach proposed in this article is different from the above mentioned approaches. We try to fix some disadvantages of SA like its slowness and its wandering near the global minimum in the final stage of search. So, we use a new simplex method to generate the movements trying to explore the function domain more carefully while applying accelerated SA and also use another simplex method to accelerate the final stage in the search.

The article is organized as follows. In Section 2, we state the description of the proposed methods. Experimental results along with the initialization of some parameters and the setting of the control parameters of the proposed methods are discussed in Section 3. The conclusion makes up Section 4.

2 THE DESCRIPTION OF THE PROPOSED METHODS

In this section, we describe the SDS, SSA, and DSSA methods and introduce the initial and control parameters that are required by these methods. The values of these parameters used in the experiments will be given in Section 3.

2.1 Simple Direct Search (SDS)

Before we state the steps of the SDS method, we will introduce the main ideas which SDS comes from. The most famous simplex based direct search method was proposed by Nelder and Mead [17] in 1965. Nelder–Mead method has been studied extensively. In 1991, Dennis and Torczon [8] proposed a new form of direct search method, called the multidirectional search method, which can be considered an effective modification of Nelder–Mead method in the parallel computing environment. The main difference between Nelder–Mead method and the multidirectional search method is that the number of points used in the reflection step equals n in the latter method and equals one in Nelder–Mead method. Recently, Tseng [23] proposed a general framework of the simplex based direct search method which contains Nelder–Mead and the multidirectional search methods as subclasses and uses a varying number of reflected points in a flexible manner.

In the SDS method, we will start with an initial simplex with $n+1$ vertices. Then, we will try to get a better movement by reflecting the worst vertex in this simplex with respect to the remaining vertices. If the new vertex is not better than the worst one, we reflect the two worst vertices. If it fails to get a better point, then we reflect the three worst vertices and so on. If we reach the case of reflecting the n worst vertices and we still fail to get any better movement, then we will shrink the simplex.

Algorithm 2.1 below is a formal description of the SDS method. We require that the initial simplex S be a nondegenerate simplex with vertices $x_1, x_2, \dots, x_{n+1}$. We assume throughout that the vertices are sorted according to the objective function values

$$f(x_1) \le f(x_2) \le \cdots \le f(x_{n+1}). \tag{1}$$

We will refer to x_1 as the best vertex and x_{n+1} as the worst. Two scalar parameters ρ and σ that represent coefficients of reflection and shrinkage, respectively, must be specified to define the SDS method. We suppose that these parameters satisfy

$$\rho > 0, \qquad 0 < \sigma < 1. \tag{2}$$

Algorithm 2.1 SDS(S,f, ϵ)

0. Choose parameters ρ and σ satisfying (2). Select an initial simplex S with vertices $x_1, x_2, \dots, x_{n+1}$. Choose a sufficiently small number $\epsilon > 0$.
1. *Order*. Order and relabel the vertices of S so that (1) holds.
2. If $f(x_{n+1}) - f(x_1) \le \epsilon$, then terminate. Otherwise, go to Step 3.
3. Let $k := 1$. (k is the number of reflected points.)
4. If $k \le n$, then go to Step 5 to perform the reflection. Otherwise, go to Step 6 to perform the shrinkage.
5. *Reflect*. Compute the k reflected points $\{x_i^r\}_{i=n-k+2}^{n+1}$ by

$$x_i^r := \overline{x} + \rho(\overline{x} - x_i), \quad i = n+1, n, \dots, n-k+2, \tag{3}$$

where $\overline{x}$ is the centroid of the set $\{x_1, x_2, \dots, x_{n-k+1}\}$, i.e.,

$$\overline{x} := \frac{1}{n-k+1} \sum_{i=1}^{n-k+1} x_i. \tag{4}$$

Evaluate $f(x_i^r)$, $i = n+1, n, \ldots, n-k+2$. If $\min_{n-k+2 \le i \le n+1}\{f(x_i^r)\} < f(x_1)$, then put $x_i := x_i^r$, $i = n+1, n, \ldots, n-k+2$, and go to Step 1. Otherwise, let $k := k+1$ and go to Step 4.

6. *Shrink*. Evaluate the function f at the n new vertices

$$x_i := x_1 + \sigma(x_i - x_1), \quad i = 2, 3, \ldots, n+1. \tag{5}$$

Go to Step 1.

The coefficient of reflection ρ, in Algorithm 2.1, can be randomly chosen from the interval (0.9, 1.1) to make more effective exploration. Algorithm 2.1 terminates when the function values at all the vertices become close to each other. However, if the number of iterations exceeds the predetermined allowed number of iterations, then we may terminate the algorithm.

Simplex methods maintain at each iteration a nondegenerate simplex and the function values at the vertices. When one or more test points, along with their function values, are computed, we proceed to the next iteration with a new simplex. A most general approach of simplex methods was proposed by Tseng [23]. In this general approach, an integer m $(1 \le m \le n)$ is chosen to specify the number of "good" vertices to be retained in constructing the initial trial simplices. The other vertices will be reflected, and then either expanded or contracted, at each iteration. If it fails to get a better point, then the whole simplex will be shrunk with respect to the best vertex. However, in Algorithm 2.1, we simply intensify the search by only repeating the reflection step in many directions and if it fails to get a better point, then we shrink the simplex with respect to the best vertex. It is noteworthy that the main aim of SDS is to enhance the exploration role to be a good seed to generate the global optimization methods SSA and DSSA. So, we will not compare the behavior of SDS with the other simplex based direct search methods in this article.

2.2 Simplex Simulated Annealing (SSA)

Since the SDS method is still a local search method, we hybridize it with the standard SA to perform SSA method, which is expected to have the ability to look for a global minimum. The basic idea of the standard SA is that it tries to avoid being trapped in local minima by making uphill move with the probability $p = \exp(-\Delta E/T)$, where ΔE is the amount of increase in the objective value caused by the uphill move and T is a parameter referred to as "annealing temperature". To avoid accepting large uphill move in the later stage of the search, the parameter T will be decreased over time by a schedule which is called "the cooling schedule". We will apply this SA acceptance condition on the reflected points in the SDS method to obtain SSA method. In other words, we allow the possibility of accepting reflected points which do not include any better solution. Algorithm 2.2 describes the steps of SSA method and shows how we apply the SA acceptance and the cooling schedule with the lower limit temperature $T_{\min}$.

Algorithm 2.2 SSA($S, f, \epsilon, T_{\min}, M$)

```
0. Choose parameter $\sigma \in (0,1)$. Select an initial simplex $S$ with
   vertices $x_1, x_2, \ldots, x_{n+1}$. Set the parameters of the cooling schedule:
   the initial temperature $T$, $T_{\min}$, and $M$. Choose a sufficiently small
   number $\epsilon > 0$.
1. Order. Order and relabel the vertices of $S$ so that (1) holds.
2. If $f(x_{n+1}) - f(x_1) \leq \epsilon$ or $T < T_{\min}$, then terminate. Otherwise, go
   to Step 3.
3. Repeat the following Steps 3.1–3.5 $M$ times.
   3.1. Let $k := 1$.
   3.2. If $k \leq n$, then go to Step 3.3 to perform the reflection.
        Otherwise, go to Step 3.4 to perform the shrinkage.
   3.3. Reflect. Compute the $k$ reflected points $\{x_i^r\}_{i=n-k+2}^{n+1}$ by

        $$x_i^r := \overline{x} + \rho(\overline{x} - x_i), \quad i = n+1, n, \ldots, n-k+2, \qquad (6)$$

        where $\rho$ is randomly chosen from the interval $(0.9, 1.1)$ and $\overline{x}$
        is defined by (4). Evaluate $f(x_i^r)$, $i = n+1, n, \ldots, n-k+2$,
        and put $\widehat{f} := \min_{n-k+2 \leq i \leq n+1}\{f(x_i^r)\}$.
   3.3.1. If $\widehat{f} < f(x_1)$, then go to Step 3.3.3.
   3.3.2. Compute $p := \exp\{-(\widehat{f} - f(x_1))/T\}$ and choose $U$ ran-
          domly from the interval $(0,1)$. If $p \geq U$, then go to
          Step 3.3.3. Otherwise, let $k := k+1$ and go to Step 3.2.
        3.3.3. Set $x_i := x_i^r,\ i = n+1, n, \ldots, n-k+2$. Go to Step 3.5.
    3.4. *Shrink.* Shrink the simplex by determining $n$ vertices by (5).
         Go to Step 3.5
    3.5. *Sort.* Sort the vertices of $S$ so that (1) holds.
4. Reduce the temperature $T$ and go to Step 2.
```

In Algorithm 2.2, the coefficient of reflection ρ is determined by choosing a random number from the interval $(0.9, 1.1)$. SSA method terminates when the function values at the vertices are close to each other or the cooling schedule is completed. The main role of M, the number of inner iterations per each temperature, is to get closer to the equilibrium because it has been proved [13,15] that when M is sufficiently large and the temperature T is slowly reduced, the solution x will eventually be frozen at the global minimum.

2.3 Direct Search Simulated Annealing (DSSA)

It is known that the standard SA may quickly approach the neighborhood of the global minimum but has a difficulty in obtaining some required accuracy. So, it is suitable to finish the algorithm with a faster convergent method. According to this idea, we modify SSA method to obtain the DSSA method as follows:

1. Accelerate the cooling schedule in SSA, i.e., use a smaller reduction factor for the temperature T.
2. Set the coefficient of shrinkage σ equals one to maintain the size of the initial simplex large enough. Actually, setting $0 < \sigma < 1$ is effective for achieving good behavior near a minimum in SSA, especially, in the final stage of search. However, in DSSA, the situation is different because we use the simplex simulated annealing part in exploring the whole domain and storing the best visited point in a list. So, perfect behavior near a minimum is not pursued in this part but it will be considered in the last part of DSSA using a complete local search method starting from each point in the best point list. In fact, it is known that local search methods have much better behavior near a minimum than global methods.
3. Store the best solutions found by the accelerated SSA in a list called "best list" as mentioned earlier and apply another local search method starting from each element of the best list to improve further these best solutions.

According to these modifications of SSA, we can state the steps of the DSSA method in Algorithm 2.3.

Algorithm 2.3. DSSA($S, f, \epsilon, T_{\min}, M$)

```
0. Select an initial simplex $S$ with vertices $x_1, x_2, \ldots, x_{n+1}$. Set the parameters of the cooling schedule: the initial temperature $T$, $T_{\min}$, and $M$. Set the size of the best list. Choose a sufficiently small number $\epsilon > 0$.
1. *Order.* Order and relabel the vertices of $S$ so that (1) holds.
2. *Best list.* Store the $m$ best points in the best list.
3. If $f(x_{n+1}) - f(x_1) \le \epsilon$ or $T < T_{\min}$, then go to Step 5. Otherwise, go to Step 4.
4. Repeat the following Steps 4.1–4.4 $M$ times.
   4.1. Let $k := 1$.
   4.2. *Reflect.* Compute the $k$ reflected points $\{x_i^r\}_{i=n-k+2}^{n+1}$ by

        $$x_i^r := \overline{x} + \rho(\overline{x} - x_i), \quad i = n+1, n, \ldots, n-k+2, \qquad (7)$$

        where $\rho$ is randomly chosen from the interval $(0.9, 1.1)$ and $\overline{x}$ is defined by (4). Evaluate $f(x_i^r)$, $i = n+1, n, \ldots, n-k+2$, and put $\widehat{f} := \min_{n-k+2 \le i \le n+1}\{f(x_i^r)\}$.
        4.2.1. If $\widehat{f} < f(x_1)$, then go to Step 4.2.3.
        4.2.2. Compute $p := \exp\{-(\widehat{f} - f(x_1))/T\}$ and choose $U$ randomly from the interval $(0, 1)$. If $p \ge U$, then go to Step 4.2.3. Otherwise, let $k := k + 1$ and go to Step 4.4.
        4.2.3. Set $x_i := x_i^r$, $i = n+1, n, \ldots, n-k+2$. Go to Step 4.3.
   4.3. *Sort.* Sort the vertices of $S$ so that (1) holds and update the best list.
   4.4. If $k \le n$, then go to Step 4.2.
5. Reduce the temperature $T$ and go to Step 2.
6. From each point in the best list, construct a smaller simplex. Then, apply Kelley's modification of the Nelder–Mead method on each of these simplices.
```

In the DSSA method, we use the Kelley's modification [11] of the Nelder–Mead method to refine the points stored in the best list. We prefer to use this method instead of the original Nelder–Mead method because after more than thirty years of studying and applying the Nelder–Mead method, McKinnon [16] shows that the Nelder–Mead algorithm can stagnate and converge to a nonoptimal point even for very simple problems. However, Kelley [11,12] proposes a test for sufficient decrease which, if passed for all iterations, will guarantee convergence of the Nelder–Mead iteration to a stationary point under some appropriate conditions.

3 EXPERIMENTAL RESULTS

The performance of SDS, SSA, and DSSA methods has been evaluated to show how SA can affect the local search method SDS toward its generalization in global optimization. Moreover, the comparison between the results of SSA and DSSA shows the effect of the acceleration of convergence to improve the final results. Finally, the performance of our final method DSSA has been compared with some other meta-heuristics methods. The comparison was made using a set of some well-known functions, which are listed in Appendix A.

3.1 Setting of Parameters

Some initial parameters and control parameters must be specified to define the complete implementation of the methods SDS, SSA, and DSSA.

3.1.1 Choosing the Initial Simplex

First, we randomly choose an initial orientation x_1 from some predetermined range of initial points for each function. Then, we take a step in each coordinate direction, called the edge of the simplex, to construct a right-angled simplex with vertices $x_1, x_2, \ldots, x_{n+1}$. The edge length of the simplex is chosen to befit the range of initial points for each function. For all test functions the edge length is varied from 0.125 to 4 depending on the range of initial points of each function. Moreover, we start with some suitable edge length and if the difference of the functional values at the simplex vertices is very small, then this edge length will be doubled until we get an improvement on the condition of the initial simplex or reach the maximum allowed edge length. This method of choosing the initial simplex is applied on all methods SDS, SSA, and DSSA.

3.1.2 The Cooling Schedule

The cooling schedule consists of the initial temperature $T_{\max}$, the cooling function, the epoch length M and the stopping condition. As in Kirkpatrick *et al.* [13], we choose the value of $T_{\max}$ large enough to make the initial probability of accepting transition close to 1. We set the initial probability equal to 0.9. Then, $T_{\max}$ is calculated from the equation

$$T_{\max} = -\frac{f(x_{n+1}) - f(x_1)}{\ln(0.9)}. \tag{8}$$

The temperature is reduced with a so-called cooling function F, i.e., the temperature at the kth epoch is determined with $T_k = F(T_{k-1})$. In the standard SA, this equation will be $T_k = \alpha T_{k-1}$, where $\alpha \in [0.5, 0.99]$ is a parameter called cooling ratio. In SSA algorithm, we set $\alpha = 0.9$. Since the DSSA algorithm is designed by accelerating SSA, we set $\alpha = 0.5$, and the computational experience shows that this value of α gives good results for most of the test functions. However, for some hard functions; Shekel functions, Shubert function and Griewank function, we have observed that it is more effective to slow down the cooling schedule by setting $\alpha = 0.7$. Epoch length M is the number of trials allowed at each temperature and we set it equal to $10n$ in SSA and n in DSSA. Finally, the stopping condition is comprised of the minimum allowed temperature $T_{\min}$ which equals $10^{-5} \times T_{\max}$ in both methods SSA and DSSA.

3.1.3 Termination Criteria

The termination criteria of SDS, SSA, and DSSA algorithms are intended to reflect the progress of these algorithms. So, we terminate these algorithms when the function values at all the vertices become close to each other, i.e.,

$$f(x_{n+1}) - f(x_1) \leq \epsilon, \tag{9}$$

where the tolerance ϵ is a small positive number and we set it 10^{-6} in SDS, SSA and 10^{-8} in DSSA. Moreover, SSA algorithm and the SA part of the DSSA algorithm can also be terminated if the cooling schedule is completed. However, if the number of iterations exceeds the predetermined allowed number of iterations, then we may terminate the algorithms. This maximum number equals $50n$ in SDS and DSSA and equals $1000n$ in SSA. We remark that, for Easom function, DSSA has had some difficulty in finding its minimum because it lies in a very narrow hole and outside this narrow hole of the graph the function is almost flat. Termination before reaching this narrow hole could be avoided by repeating the algorithm with reducing the edge length of the simplex in each time until we get a very small edge length equal to 10^{-4}.

3.1.4 Best List

The remaining parameter is the number of the best points stored during the search in the DSSA method. This parameter is set equal to n except for the two types hard functions, Shekel functions and Griewank function, for which we set it equal to $2n$.

3.2 Numerical Results

To examine the performance of our algorithms, we tested them on some well-known functions [3,5,24]. The behavior of these test functions varies; we have functions with some studded local minima such as Goldstein and Price function, functions with many crowded local minima such as Shubert function, functions with a global minimum lying in a very narrow hole such as Easom function, functions with a narrow valley such as Rosenbrock function, and smooth functions such as De Joung function and Zakharov function. For each function we made 100 trials with different starting points. The average number of function evaluations and the average error are related to only successful trials.

First, to demonstrate the effect of hybridizing SA with SDS to design SSA and DSSA, we show in Table I the percentage of successful trials. From Table I, we see that the rate of success for SSA is generally better than that for SDS. However, the behavior of SDS

TABLE I Percentage of successful trials for SDS, SSA, and DSSA

Function	*SDS*	*SSA*	*DSSA*	*Function*	*SDS*	*SSA*	*DSSA*
RC	100	99	100	$S_{4,5}$	14	61	81
ES	12	68	93	$S_{4,7}$	19	60	84
GP	41	91	100	$S_{4,10}$	15	59	77
RT	76	100	100	R_5	4	67	100
HM	100	100	100	Z_5	100	87	100
SH	59	57	94	$H_{6,4}$	52	49	92
R_2	12	93	100	*GR*	36	82	90
Z_2	100	99	100	R_{10}	0	78	100
DJ	100	100	100	Z_{10}	0	66	100
$H_{3,4}$	88	86	100				

for Zakharov function Z_5 is better than that of SSA due to the fixed cooling schedule in SSA for all functions. We note that the behavior of these methods has changed drastically when the dimension n of function Z_n is increased to 10. Moreover, Table I clearly shows that the behavior of DSSA is the best of the three methods in terms of the rate of success.

In Table II, we show the effect of accelerating the cooling schedule in SSA and applying a local search method on the final results obtained by the accelerated SSA to design DSSA. The results in Table II reveal that the acceleration procedure successfully affects the rate of success, the number of function evaluations, and the average error.

To show to what extent DSSA succeed in accelerating SA, we compare its results with other simplex SA methods like SIMPSA and NE-SIMPSA [3]. Table III shows the average number of function evaluations obtained by each method starting from the same starting point as in [3]. The data for SIMPSA and NE-SIMPSA are taken from [3]. Actually [3] reports many results for SIMPSA and NE-SIMPSA depending on the search domain but we prefer to make our method more general without any constrains during the search. Moreover, we have chosen the best results obtained by SIMPSA and NE-SIMPSA from Table III in [3] to make the comparison simpler and fair.

Next we compare the DSSA method with three other meta-heuristics methods based on SA, Tabu search, and genetic algorithm. These methods are:

1. Enhanced Continuous Tabu Search (ECTS) [5].
2. Enhanced Simulated Annealing (ESA) [22].
3. Real-value Coding Genetic Algorithm (RCGA) [2].

TABLE II Results of SSA and DSSA

Function	*Rate of success*		*Average number of function evaluations*		*Average error*	
	SSA	*DSSA*	*SSA*	*DSSA*	*SSA*	*DSSA*
RC	99	100	12225	118	9E-3	4E-7
ES	68	93	4318	1442	4E-3	3E-9
GP	91	100	11238	261	5E-3	4E-9
RT	100	100	4564	252	7E-3	5E-9
HM	100	100	10157	225	0.01	5E-8
SH	57	94	10237	457	0.1	9E-6
R_2	93	100	7387	306	3E-3	4E-9
Z_2	99	100	5868	186	8E-3	4E-9
DJ	100	100	6743	273	3E-3	5E-9
$H_{3,4}$	86	100	17756	572	0.1	2E-6
$S_{4,5}$	61	81	7856	993	6E-3	2E-6
$S_{4,7}$	60	84	9047	932	0.01	6E-7
$S_{4,10}$	59	77	9062	992	0.01	1E-5
R_5	67	100	11115	2685	0.03	3E-9
Z_5	87	100	11527	914	0.03	5E-9
$H_{6,4}$	49	92	37467	1737	0.02	2E-6
GR	82	90	12208	1830	0.1	5E-9
R_{10}	78	100	22306	16785	0.02	7E-9
Z_{10}	66	100	23883	12501	0.04	7E-9

TABLE III Average number of function evaluations in DSSA and other simplex SA methods

Function	*DSSA*	*SIMPSA*	*NE-SIMPSA*
R_2	306	10780	4508
R_4	1682	21177 (99%)	3053 (94%)
CV	1592	22615	3443
DX	6941	52556 (93%)	8613 (94%)

Table IV shows the average number of function evaluations needed by each algorithm. The results of ECTS, ESA, and RCGA are taken from their original papers [2,5,22]. For all test functions, we use the same condition as that used by ECTS [5] to judge the success of a trial which is given by

$$|f^* - f_{\text{DSSA}}| < \epsilon_1 |f^*| + \epsilon_2, \tag{10}$$

where f_{DSSA} refers to the best function value obtained by DSSA and f^* refers to the exact global minimum. We set $\epsilon_1 = 10^{-4}$ and

TABLE IV Average number of function evaluations in DSSA and other meta-heuristics

Function	*DSSA*	*ECTS*	*ESA*	*RCGA*
RC	118	$245^{\otimes}$	–	490
ES	1442 (93%)	$1284^{\otimes}$	–	642
GP	261	$231^{\otimes}$	$783^{\otimes}$	270
SH	457 (94%)	370	–	946
R_2	306	$480^{\otimes}$	796	596
Z_2	186	195	15820	437
DJ	273	338	–	395
$H_{3,4}$	572	$548^{\otimes}$	$698^{\otimes}$	324
$S_{4,5}$	993 (81%)	825 (75%)	$1137^{\otimes}$ (54%)	1158 (62%)
$S_{4,7}$	932 (84%)	910 (80%)	$1223^{\otimes}$ (54%)	1143 (70%)
$S_{4,10}$	992 (77%)	989 (75%)	$1189^{\otimes}$ (50%)	1235 (58%)
R_5	2685	$2142^{\otimes}$	5364	4150 (60%)
Z_5	914	$2254^{\otimes}$	96799	1115
$H_{6,4}$	1737 (92%)	$1520^{\otimes}$	$2638^{\otimes}$	937
R_{10}	16785	15720 (85%)	$12403^{\otimes}$	8100 (70%)
Z_{10}	12501	4630	$15820^{\otimes}$	2190

$\epsilon_2 = 10^{-6}$. The ESA method used the same condition for testing the successful trials with smaller ϵ_1 and ϵ_2. However, for the results marked by ($^{\otimes}$) in Table IV, their original corresponding data in Table II in [5] and Table I in [22] seem to contain some inconsistencies. Since the authors of [5] used the same condition as (10) to test the successful trials, the average errors for the functions R_2, R_5, and Z_5 must be less than 10^{-6} because $f^* = 0$ for all these functions. However, the average errors corresponding to these functions are reported to be greater than 10^{-6}. For instance, the average error corresponding to the function R_5 in Table II in [5] is 0.08, i.e., there are some trials that did not satisfy the successful trial condition but the authors reported that the rate of success equals 100%. Moreover, the results corresponding to the functions $RC, ES, GP, H_{3,4}$, and $H_{6,4}$ in Table II in [5] also contain the same kind of inconsistencies. For the same reasons, the ESA results marked by ($^{\otimes}$) suffer from the same inconsistencies.

The comparison given in Table IV shows the DSSA outperforms the others for some functions and has similar behavior for other functions. However, Table V shows that DSSA generally produces more accurate solutions than the others. It is noteworthy that the efficiency of the simplex method dwindles with dimensionality, which explains

TABLE V Average errors in function value in DSSA and other meta-heuristics

Function	*DSSA*	*ECTS*	*ESA*	*RCGA*
RC	4E-7	5E-2	–	3E-3
ES	3E-9	1E-2	–	3E-9
GP	4E-9	2E-3	9E-3	1E-9
SH	9E-6	1E-3	–	6E-4
R_2	4E-9	2E-2	–	1E-12
Z_2	4E-9	2E-7	–	1E-10
DJ	5E-9	3E-8	–	6E-4
$H_{3,4}$	2E-6	9E-2	5E-4	7E-3
$S_{4,5}$	2E-6	1E-2	4E-3	1E-3
$S_{4,7}$	6E-7	1E-2	8E-3	1E-4
$S_{4,10}$	1E-5	1E-2	4E-2	4E-3
R_5	3E-9	8E-2	–	1E-1
Z_5	5E-9	4E-6	–	9E-4
$H_{6,4}$	2E-6	5E-2	6E-2	3E-2
R_{10}	7E-9	2E-2	4E-2	1E-1
Z_{10}	7E-9	2E-7	2E-3	3E-3

the greatest margin of superiority for DSSA on Z_5 while it does not outperform the others on Z_{10}.

4 CONCLUSION

The SA method usually suffers from slow convergence due to its random nature of movements. Moreover, SA also suffers from the difficulty in obtaining some required accuracy although it may quickly approach the neighborhood of the global minimum. In this article, we have focused on the importance of creating direct-search-based logical movements while applying SA and the importance of accelerating the final stage of SA by using a faster convergent method. The obtained results demonstrate that these two concepts can be successfully realized by effectively combining direct search methods with SA. Moreover, the experimental results show that the DSSA method is very efficient and robust.

Acknowledgment

This research was supported in part by a Grant-in-Aid for Scientific Research from the Ministry of Education, Science, Sports and Culture of Japan.

References

[1] K.S. Al-Sultan and M.A. Al-Fawzan (1997). A tabu search Hooke and Jeeves algorithm for unconstrained optimization. *European J. of Operational Research*, **103**, 198–208.

[2] M. Bessaou and P. Siarry (2001). A genetic algorithm with real-value coding to optimize multimodal continuous functions. *Struct. Multidisc. Optim.*, **23**, 63–74.

[3] M.F. Cardoso, R.L. Salcedo and S.F. de Azevedo (1996). The simplex-simulated annealing approach to continuous non-linear optimization. *Comput. Chem. Eng.*, **20**, 1065–1080.

[4] M.F. Cardoso, R.L. Salcedo, S.F. de Azevedo and D.Barbosa (1997). A simulated annealing approach to the solution of minlp problems. *Comput. Chem. Eng.*, **21**, 1349–1364.

[5] R. Chelouah and P. Siarry (2000). Tabu search applied to global optimization. *European J. of Operational Reasearch*, **123**, 256–270.

[6] R. Chelouah and P. Siarry. An hybrid method using genetic algorithm and Nelder-Mead simplex algorithms for the global optimization, submitted.

[7] R. Chelouah and P. Siarry. An hybrid method combining continuous Tabu Search and Nelder-Mead simplex algorithms for the global optimization of multiminima functions, submitted.

[8] J.E. Dennis and V. Torczon (1991). Direct search methods on parallel machines. *SIAM J. Optim.*, **1**, 448–474.

[9] C.A. Floudas and P.M. Pardalos (Eds.) (1992). *Recent Advances in Global Optimization*. Princeton University Press, Princeton, NJ.

[10] R. Horst and P.M. Pardalos (Eds.) (1995). *Handbook of Global Optimization*. Kluwer Academic Publishers, Boston, MA.

[11] C.T. Kelley (1999). Detection and remediation of stagnation in the Nelder-Mead algorithm using a sufficient decrease condition. *SIAM J. Optim.*, **10**, 43–55.

[12] C.T. Kelley (1999). Iterative methods for optimization. *Frontiers Appl. Math.*, Vol. 18. SIAM, Philadelphia, PA.

[13] S. Kirkpatrick, C.D. Gelatt Jr. and M.P. Vecchi (1983). Optimisation by simulated annealing. *Science*, **220**, 671–680.

[14] V. Kvasnicka and J. Pospichal (1997). A hybrid of simplex method and simulated annealing. *Chemometrics and Intelligent Laboratory Systems*, **39**, 161–173.

[15] P.J. Laarhoven and E.H. Aarts (1987). *Simulated Annealing: Theory and Applications*. D. Reidel Publishing Company, Dordrecht, Holland.

[16] K.I.M. McKinnon (1999). Convergence of the Nelder-Mead simplex method to a nonstationary point. *SIAM J. Optim.*, **9**, 148–158.

[17] J.A. Nelder and R. Mead (1965). A simplex method for function minimization. *Comput. J.*, **7**, 308–313.

[18] I.H. Osman and J.P. Kelly (1996). *Meta-Heuristics: Theory and Applications*. Kluwer Academic Publishers, Boston, MA.

[19] P.M. Pardalos and H.E. Romeijn (Eds.) (2002). *Handbook of Global Optimization*. Kluwer Academic Publishers, Boston, MA.

[20] P.M. Pardalos and M.G.C. Resende (Eds.) (2002). *Handbook of Applied Optimization*. Oxford University Press, Oxford.

[21] W.H. Press and S.A. Teukolsky (1991). Simulated annealing optimization over continuous spaces. *Comput. Phys.*, **5**, 426–429.

[22] P. Siarry, G. Berthiau, F. Durbin and J. Haussy (1997). Enhanced simulated annealing for globally minimizing functions of many continuous variables. *ACM Transactions on Mathematical Software*, **23**, 209–228.

[23] P. Tseng (1999). Fortified-descent simplicial search method: a general approach. *SIAM J. Optim.*, **10**, 269–288.

[24] P.P. Wang and D.S. Chen (1996). Continuous optimization by a variant of simulated annealing. *Computational Optimization and Applications*, **6**, 59–71.
[25] J. Yen, J.C. Liao, B. Lee and D. Randolph (1998). A hybrid approach to modeling metabolic systems using a genetic algorithm and simplex method. *IEEE Trans. on Syst., Man, and Cybern. B*, **28**, 173–191.

APPENDIX

A List of Test Functions

A.1 Branin RCOS Function (RC)

- Number of variables: $n = 2$.
- Definition: $RC(x_1, x_2) = (x_2 - (5.1/4\pi^2)x_1^2 + (5/\pi)x_1 - 6)^2 + 10(1 - (1/8\pi))\cos(x_1) + 10$.
- Range of initial points: $-5 < x_1 < 10,\ 0 < x_2 < 15$.
- Number of local minima: no local minimum.
- Global minima: $(x_1, x_2)^* = (-\pi, 12.275), (\pi, 2.275), (9.42478, 2.475)$; $RC((x_1, x_2)^*) = 0.397887$.

A.2. Easom Function (ES)

- Number of variables: $n = 2$.
- Definition: $ES(x_1, x_2) = -\cos(x_1)\cos(x_2)\exp(-(x_1 - \pi)^2 - (x_2 - \pi)^2)$.
- Range of initial points: $-10 < x_j < 10,\ j = 1, 2$.
- Number of local minima: several local minima.
- The global minimum: $(x_1, x_2)^* = (\pi, \pi)$; $ES((x_1, x_2)^*) = -1$.

A.3 Goldstein and Price Function (GP)

- Number of variables: $n = 2$.
- Definition: $GP(x_1, x_2) = u * v$, where $u = 1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)$, and $v = 30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)$.
- Range of initial points: $-2 < x_j < 2,\ j = 1, 2$.
- Number of local minima: 4 local minima.
- The global minimum: $(x_1, x_2)^* = (0, -1)$; $GP((x_1, x_2)^*) = 3$.

A.4 Rastrigin Function (RT)

- Number of variables: $n = 2$.
- Definition: $RT(x_1, x_2) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$.

- Range of initial points: $-1 < x_j < 1,\ j = 1, 2$.
- Number of local minima: many local minima.
- The global minimum: $(x_1, x_2)^* = (0, 0);\ RT((x_1, x_2)^*) = 0$.

A.5 Hump Function (HM)

- Number of variables: $n = 2$.
- Definition: $HM(x_1, x_2) = 1.0316285 + 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$.
- Range of initial points: $-5 < x_j < 5,\ j = 1, 2$.
- Number of local minima: no local minima.
- Global minima: $(x_1, x_2)^* = (0.0898, -0.7126), (-0.0898, 0.7126);$ $HM((x_1, x_2)^*) = 0$.

A.6 Shubert Function (SH)

- Number of variables: $n = 2$.
- Definition: $SH(x_1, x_2) = (\sum_{j=1}^{5} j\cos((j+1)x_1 + j))\times (\sum_{j=1}^{5} j\cos((j+1)x_2 + j))$.
- Range of initial points: $-10 < x_j < 10,\ j = 1, 2$.
- Number of local minima: 760 local minima.
- Global minima: 18 global minima and $SH((x_1, x_2)^*) = -186.7309$.

A.7 De Joung Function (DJ)

- Number of variables: $n = 3$.
- Definition: $DJ(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2$.
- Range of initial points: $-5 < x_j < 5,\ j = 1, 2, 3$.
- Number of local minima: no local minima.
- The global minimum: $(x_1, x_2, x_3)^* = (0, 0, 0);\ DJ((x_1, x_2, x_3)^*) = 0$.

A.8 Hartmann Function ($H_{3,4}$)

- Number of variables: $n = 3$.
- Definition: $H_{3,4}(\mathbf{x}) = -\sum_{i=1}^{4} c_i \exp[-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2]$.
- Range of initial points: $0 < x_j < 1,\ j = 1, 2, 3$.
- Number of local minima: 4 local minima.
- The global minimum: $\mathbf{x}^* = (0.114614, 0.555649, 0.852547);$ $H_{3,4}(\mathbf{x}^*) = -3.86278$.

i		a_{ij}		c_i		p_{ij}	
1	3.0	10.0	30.0	1.0	0.689	0.1170	0.2673
2	0.1	10.0	35.0	1.2	0.4699	0.4387	0.7470
3	3.0	10.0	30.0	3.0	0.1091	0.8732	0.5547
4	0.1	10.0	35.0	3.2	0.0381	0.5743	0.8828

A.9 Colville Function (CV)

- Number of variables: $n = 4$.
- Definition: $CV(\mathbf{x}) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90(x_3^2 - x_4)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1)$.
- The global minimum: $\mathbf{x}^* = (1, 1, 1, 1)$; $CV(\mathbf{x}^*) = 0$.

A.10 Shekel Functions ($S_{4,m}$)

- Number of variables: $n = 4$.
- Definition: $S_{4,m}(\mathbf{x}) = -\sum_{i=1}^{m}[\sum_{i=1}^{4}(x_i - a_i)^2 + c_i]^{-1}$.
- Three functions were considered: $S_{4,5}, S_{4,7}$ and $S_{4,10}$.
- Range of initial points: $0 < x_j < 10$, $j = 1, \ldots, 4$.
- Number of local minima: m local minima.
- Same global minimum for three functions $S_{4,5}$, $S_{4,7}$ and $S_{4,10}$: $\mathbf{x}^* = (4, 4, 4, 4)$;

$S_{4,5}(\mathbf{x}^*) = -10.1532$, $S_{4,7}(\mathbf{x}^*) = -10.4029$ and $S_{4,10}(\mathbf{x}^*) = -10.5364$.

i		a_{ij}			c_i
1	4.0	4.0	4.0	4.0	0.1
2	1.0	1.0	1.0	1.0	0.2
3	8.0	8.0	8.0	8.0	0.2
4	6.0	6.0	6.0	6.0	0.4
5	3.0	7.0	3.0	7.0	0.4
6	2.0	9.0	2.0	9.0	0.6
7	5.0	5.0	3.0	3.0	0.3
8	8.0	1.0	8.0	1.0	0.7
9	6.0	2.0	6.0	2.0	0.5
10	7.0	3.6	7.0	3.6	0.5

A.11 Hartmann Function ($H_{6,4}$)

- Number of variables: $n = 6$.
- Definition: $H_{6,4}(\mathbf{x}) = -\sum_{i=1}^{4} c_i \exp[-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2]$.
- Range of initial points: $0 < x_j < 1$, $j = 1, \ldots, 6$.

- Number of local minima: 6 local minima.
- The global minimum: $\mathbf{x}^* = (0.201690, 0.150011, 0.476874, 0.275332, 0.311652, 0.657300)$; $H_{6,4}(\mathbf{x}^*) = -3.32237$.

i	a_{ij}						c_i
1	10.00	3.00	17.00	3.50	1.70	8.00	1.0
2	0.05	10.00	17.00	0.10	8.00	14.00	1.2
3	3.00	3.50	1.70	10.0	17.00	8.00	3.0
4	17.00	8.00	0.05	10.00	0.10	14.00	3.2

i	p_{ij}					
1	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
2	0.2329	0.4135	0.8307	0.3736	0.1004	0.9991
3	0.2348	0.1451	0.3522	0.2883	0.3047	0.6650
4	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

A.12 Griewank Function (GR)

- Number of variables: $n = 6$.
- Definition: $GR(\mathbf{x}) = \sum_{j=1}^{6} x_j^2/4000 - \prod_{j=1}^{6} \cos(x_j/\sqrt{j}) + 1$.
- Range of initial points: $-1 < x_j < 1,\ j = 1, 2, \ldots, 6$.
- Number of local minima: many local minima.
- The global minimum: $\mathbf{x}^* = (0, \ldots, 0),\ GR(\mathbf{x}^*) = 0$.

A.13 Dixon Function (DX)

- Number of variables: $n = 10$.
- Definition: $DX(\mathbf{x}) = (1 - x_1)^2 + (1 - x_{10})^2 + \sum_{j=1}^{9} (x_i^2 - x_{i+1})^2$.
- The global minimum: $\mathbf{x}^* = (1, 1, 1, 1)$; $DX(\mathbf{x}^*) = 0$.

A.14 Rosenbrock Functions (R_n)

- Number of variables: $n = 2, 5, 10$.
- Definition: $R_n(\mathbf{x}) = \sum_{j=1}^{n-1} [100(x_j^2 - x_{j+1})^2 + (x_j - 1)^2]$.
- Range of initial points: $-5 < x_j < 10,\ j = 1, 2, \ldots, n$.
- Number of local minima: no local minimum.
- The global minimum: $\mathbf{x}^* = (1, \ldots, 1),\ R_n(\mathbf{x}^*) = 0$.

A.15 Zakharov functions (Z_n)

- Number of variables: $n = 2, 5, 10$.
- Definition: $Z_n(\mathbf{x}) = \sum_{j=1}^{n} x_j^2 + (\sum_{j=1}^{n} 0.5jx_j)^2 + (\sum_{j=1}^{n} 0.5jx_j)^4$.
- Range of initial points: $-5 < x_j < 10,\ j = 1, 2, \ldots, n$.
- Number of local minima: no local minimum.
- The global minimum: $\mathbf{x}^* = (0, \ldots, 0),\ Z_n(\mathbf{x}^*) = 0$.