Yokohama Publishers
ISSN 1349-8169 ONLINE JOURNAL

Pacific Journal of Optimization
Vol. 10, No. 2, pp. 385–399, April 2014

A SMOOTHING FUNCTION METHOD WITH UNIFORM DESIGN FOR GLOBAL OPTIMIZATION*

Fei Wei, Yuping Wang and Zhiqing Meng

Abstract: A common difficulty for global optimization method is that it is not easy to escape from the local optimal solution and therefore often does not find the global optimal solution. In order to overcome this drawback, first, a smoothing function is constructed that can eliminate all such local optimal solutions worse than the best solution found so far. Second, this smoothing function can keep the local optimal solutions unchanged in the region in which the values of the original function are not worse than its value at the best solution found so far. Third, by making use of the properties of the smoothing function, the uniform design search technique is properly combined into the algorithm, which will make the proposed algorithm converge much faster. Based on all these, a novel effective evolutionary algorithm for global optimization is proposed. At last, the numerical simulations for several standard benchmark problems are made and the simulation results show that the proposed algorithm is very effective.

Key words: *global optimization, evolutionary algorithm, smoothing function, uniform design*

Mathematics Subject Classification: *90-08*

1 Introduction

More and more practical problems in science, economics, engineering and other fields can be formulated as global optimization problems. Lots of researchers have been attracted to the field of global optimization.

In this paper, we consider the following global optimization problem:

$$(P) \quad \begin{cases} \min & f(x) \\ s.t. & x \in R^n. \end{cases}$$

where $f(x) : R^n \to R$ is a twice continuously differentiable function. Suppose $f(x)$ satisfies the condition $f(x) \to +\infty$ as $\|x\| \to +\infty$. Then there exists a closed bounded domain Ω called an operating region that contains all local minimizers of $f(x)$. Then the global optimization problem (P) can be rewritten into an equivalent form as follows.

$$(P1) \quad \begin{cases} \min & f(x) \\ s.t. & x \in \Omega = [l, u] = \{x | l \le x \le u, l, u \in R^n\}. \end{cases}$$

Because Ω can be estimated before problem (P) is solved, so we can assume that Ω is known without loss of generality. We only consider problem (P1) in the following.

*This work is supported by The National Natural Science Foundation of China (No. 61272119).

In recent years, many new theoretical and computational contributions have been reported for solving global optimization problems. Global optimization is mainly concerned with the characteristics and algorithms for the multi-modal functions. In general, the existing approaches can be classified into two categories: deterministic methods [1,3,7,8,13,15,18, e.g.] and probabilistic methods [2,4,11,12,20, e.g.]. The typical examples of the former are the filled function method (FFM) [7,8], the trajectory method [3,18], the tunneling method [13], and the covering method [1, 15], whereas ones of the latter are the clustering method [2], evolutionary algorithms (EAs) [11, 12, 20], and the simulated annealing method [4], where evolutionary algorithms are one of most efficient and popular algorithms, they exploit a set of potential solutions, named a population, and detect the optimal solution through cooperation and competition among individuals of the population. However, for EAs in global optimization, the major challenge is that an algorithm may be trapped in the local optima of the objective function and the convergence speed may be slow. These issues are particularly challenging when the dimension of the problem is high and there are numerous local optima. In order to improve the performance of EAs, researchers have incorporated other techniques to enhance their performance. Among these techniques, function smoothing and local search are two important and effective techniques. In the function smoothing techniques, the objective function is transformed into another function called smoothing function which can smooth the rugged landscapes of the objective function, and alleviate local minimizers. As a result, looking for global minimizers by EAs becomes easier. One of these techniques constructs a smoothing function called "stretching" function (e.g., [17]). But it usually generates the so called "Mexican hat" effect (for details please refer to [17]), and it is sensitive to the parameter values in stretching function. Another one constructs the smoothing function by convolution transformation in which the smoothing function is defined by an integral of multiplication of the objective function and the smoothing kernel (usually Gaussian function, [24]), but to compute the smoothing function, one has to use the Monte-Carlo method. This is computationally expensive and the error of estimating the integral may be large. To overcome these shortcomings, some smoothing functions (e.g., [23]) are proposed. However, they are usually nondifferentiable, which often results in the calculation difficulties. To deal with these problems, a new smoothing function with two parameters, which is continuously differentiable and the parameters are easy to adjust, is proposed in this paper. And a new search method called uniform design search algorithm is proposed. Based on these, a novel effective evolutionary algorithm for global optimization is proposed. The numerical simulations are made and the performance of the proposed algorithm is compared with that of eight evolutionary algorithms published recently. The results indicate that the proposed algorithm is statistically sound and has better performance for the test functions.

2 Smoothing Function

For global optimization methods, one of the most frequently happened situations is that they often trap into some local optimal solutions. Thus, one of the greatest challenges of global optimization methods is how to avoid trapping into local optimal solutions. In order to achieve this purpose, a smoothing function of $f(x)$ at the current local minimizer x_k^* is designed as follows:

$$P(x, x_k^*) = f(x_k^*) + g(f(x) - f(x_k^*)), \tag{2.1}$$

$$g(t) = \begin{cases} 0, & t \geq 0, \\ r \cdot t^{\rho}, & t < 0. \end{cases}$$

where r is an adjustable positive real number as large as possible, used as the weight factor, and $\rho > 1$ is a positive odd integer.

Obviously, this smoothing function has some properties as follows:

Suppose that x_k^* is the current best solution of $f(x)$, then

- $P(x, x_k^*)$ will keep the local optimal solutions of $f(x)$ unchanged at any point x better than x_k^*. This can be proved by the following theorem 2.1.
- $P(x, x_k^*)$ will flatten the landscape at any point no better than x_k^*, i.e., $\forall x \in \Omega$, if $f(x) \geq f(x_k^*)$, then $P(x, x_k^*) = f(x_k^*)$. This can be proved by the following theorem 2.2.
- $P(x, x_k^*)$ is continuously differentiable, that is different from the existing literature (e.g., [23]) and helpful to the local search. This can be proved by the following theorem 2.3.

Theorem 2.1. *Suppose x_k^* is a local minimizer of $f(x)$, $P(x, x_k^*)$ is a smoothing function at x_k^*, if there exists a better local minimizer x_{k+1}^* of $f(x)$, then x_{k+1}^* is also a local minimizer of $P(x, x_k^*)$.*

Proof. Suppose x_{k+1}^* is a better local minimizer of $f(x)$, then there exists a small positive real number ε, and a neighborhood $\delta = U(x_{k+1}^*, \varepsilon)$, such that for all $x \in \delta$, when $x \neq x_{k+1}^*$, then $f(x) > f(x_{k+1}^*)$, therefore $f(x) - f(x_{k+1}^*) > 0$.

And for all $x \in \delta$, and $\rho > 1$ is a positive odd integer, set $\rho = \rho_1$, then

$$P(x, x_k^*) = f(x_k^*) + r \cdot (f(x) - f(x_k^*))^{\rho_1}, and$$

$$P(x_{k+1}^*, x_k^*) = f(x_k^*) + r \cdot (f(x_{k+1}^*) - f(x_k^*))^{\rho_1}.$$

Since $f(x) > f(x_{k+1}^*)$, $f(x_{k+1}^*) < f(x_k^*)$, and ρ_1 is a positive odd integer, therefore

$$(f(x) - f(x_{k+1}^*))^{\rho_1} > (f(x_{k+1}^*) - f(x_k^*))^{\rho_1},$$

and r is an adjustable positive real number, then

$$P(x, x_k^*) - P(x_{k+1}^*, x_k^*) = r \cdot [(f(x) - f(x_{k+1}^*))^{\rho_1} - (f(x_{k+1}^*) - f(x_k^*))^{\rho_1}] > 0$$

Therefore $P(x, x_k^*) > P(x_{k+1}^*, x_k^*)$, when $x \in \delta$. Then x_{k+1}^* is also a local minimizer of $P(x, x_k^*)$. The proof is completed. □

It can be seen from theorem 2.1 that the smoothing function keeps the local optimal solutions of $f(x)$ unchanged in the region where the value of $f(x)$ is better than that at the current best solution found by the algorithm. This means that the smoothing function will not destroy the global optimal solution of $f(x)$ in the region containing better than the current best solution found.

Theorem 2.2. *Suppose x_k^* is a local minimizer of $f(x)$, $P(x, x_k^*)$ is a smoothing function at x_k^*, then $P(x, x_k^*)$ will flatten the landscape at any points no better than x_k^*, i.e., $\forall x \in \Omega$, if $f(x) \geq f(x_k^*)$, then $P(x, x_k^*) = f(x_k^*)$.*

Proof. Since x_k^* is a local minimizer of $f(x)$, $P(x, x_k^*)$ is a smoothing function at x_k^*, $\forall x \in \Omega$, if $f(x) \geq f(x_k^*)$, then $P(x, x_k^*) = f(x_k^*)$. $P(x, x_k^*)$ will flatten all such local optimal solutions worse than the best solution found so far. The proof is completed. □

It can be seen from Theorem 2.2 that the smoothing function makes the objective function $f(x)$ become flat in the region where $f(x)$ is no better than that at the current best solution found. This means that the smoothing function can eliminate all the local optimal solutions no better than the current best solution found.

Theorem 2.3. *Suppose x_k^* is a local minimizer of $f(x)$, $P(x, x_k^*)$ is as shown in equation (2.1), which is a smoothing function at x_k^*, then $P(x, x_k^*)$ is continuously differentiable.*

Proof. Since x_k^* is a local minimizer of $f(x)$, $P(x, x_k^*)$ is a smoothing function at x_k^*, $\forall x \in \Omega$,

$$P(x, x_k^*) = \begin{cases} f(x_k^*), & f(x) \geq f(x_k^*), \\ f(x_k^*) + r \cdot (f(x) - f(x_k^*))^\rho, & f(x) < f(x_k^*). \end{cases}$$

Then

$$\nabla P(x, x_k^*) = \begin{cases} 0, & f(x) \geq f(x_k^*), \\ r \cdot \rho \cdot (f(x) - f(x_k^*))^{\rho-1}, & f(x) < f(x_k^*). \end{cases}$$

(i) When $f(x) \geq f(x_k^*)$, $\nabla P(x, x_k^*)$ is continuously;

(ii) When $f(x) < f(x_k^*)$, since $f(x)$ is continuously differentiable, then $\nabla f(x)$ is continuously, and $\nabla P(x, x_k^*)$ is continuously;

(iii) $\lim\limits_{f(x) \to f(x_k^*)^-} \nabla P(x, x_k^*) = 0$, and $\lim\limits_{f(x) \to f(x_k^*)^+} \nabla P(x, x_k^*) = 0$.

Therefore $\nabla P(x, x_k^*)$ is continuously at the points that $f(x) = f(x_k^*)$. The proof is completed. □

It is known from the definition of a continuously differentiable function and the above (i)-(iii) that $P(x, x_k^*)$ is continuously differentiable.

It can be seen from the third property that the smoothing function is continuously differentiable. This makes it flexible to select a local optimization method for the algorithm.

These three properties can be intuitively illustrated by a function with one variable as an example in the following Figure 1, where the solid line represents the original function, and the dotted line represents the smoothing function.

It can be seen from Figure 1 that if a proper search scheme is used to the smoothing function, the search will quickly go out the flattening region and can find a better solution fast.

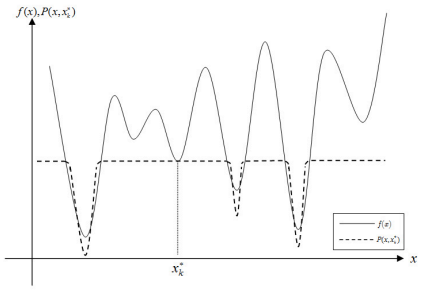

Figure 1: $P(x, x_k^*)$ is indicated by dotted line and has only three local optimal solutions, and three local optimal solutions of $f(x)$ were eliminated, where x_k^* is the best solution found by the algorithm so far.

3 Uniform Design

Note that there will be a large flat area on the landscape of the smoothing function. In order to get a better solution than the best one found so far quickly, it is necessary to go through this area fast and fall in a lower area quickly. For this purpose, we design a uniform design search method.

3.1 Uniform Design

Experimental design method is a sophisticated branch of statistics [9,16]. In this section, we briefly describe an experimental design method called uniform design. The main objective of uniform design is to sample a small set of points from a given set of points, such that the sampled points are uniformly scattered. We describe the main features of uniform design in the following, and we refer the readers to [9, 22] for more details.

Suppose the yield of a chemical depends on the temperature, the amount of catalyst, and the duration of the chemical process. These three quantities are called the factors of the experiment. If each factor has ten possible values, we say that each factor has ten levels. There are $10^3 = 1000$ combinations of levels. To find the best combination for a maximum yield, it is necessary to do 1000 experiments. When it is not possible or cost-effective to do all these experiments, it is desirable to select a small but representative sample of experiments. The uniform design was developed for this purpose [9, 22].

Let there be n factors and q levels per factor. When n and q are given, the uniform design selects q combinations out of q^n possible combinations, such that these q combinations are scattered uniformly over the space of all possible combinations. The selected q combinations are expressed in terms of a uniform array $U(q, n) = [U_{i,j}]_{q\times n}$, where $U_{i,j}$ is the jth level of the ith factor in the combination.

Uniform arrays can be constructed as follows. Consider a unit hypercube over a n-dimensional space. We denote this hypercube by the set of points in it

$$C = \{(c_1, c_2, ..., c_n), 0 \leq c_i \leq 1, i = 1, 2, ..., n\}.$$

Consider any point in C, say $r = (r_1, r_2, ..., r_n)$. We form a hyper-rectangle between 0 and r, and we denote it by the set of points in it

$$C(r) = \{(c_1, c_2, ..., c_n), 0 \leq c_i \leq r_i, i = 1, 2, ..., n\}.$$

We select a sample of q points such that they are scattered uniformly in the hypercube. Suppose $q(r)$ of these points are in the hyper-rectangle $C(r)$. Then the fraction of points in the hyper-rectangle is $q(r)/q$. The volume of the unit hypercube is 1, and hence the fraction of the volume of this hyper-rectangle is $r_1r_2...r_n$. The uniform design is to determine points in C such that the following discrepancy is minimized:

$$\sup_{r\in C} q(r)/q - r_1r_2...r_n,$$

Then we map these q points in the unit hypercube to the space with n factors and q levels. When q is prime and $q > n$, it has been proven that $U_{i,j}$ is given by [5],

$$U_{i,j} = (i\sigma^{j-1} mod q) + 1$$

where σ is a parameter given in Table 1.

Example 1: We construct a uniform array with five factors and seven levels as follows. From Table 1, we see that is equal to 3. We compute $U_{7,5}$ and we get

$$\begin{pmatrix} 2 & 4 & 3 & 7 & 5 \\ 3 & 7 & 5 & 6 & 2 \\ 4 & 3 & 7 & 5 & 6 \\ 5 & 6 & 2 & 4 & 3 \\ 6 & 2 & 4 & 3 & 7 \\ 7 & 5 & 6 & 2 & 4 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Table 1: Values of the parameter for different number of factors and different number of levels per factor

Number of levels per factor	Number of factors	σ
5	2-4	2
7	2-6	3
11	2-10	7
13	2	5
	3	4
	4-12	6
17	2-16	10
19	2,3	8
	4-18	14
23	2,13,14,20-22	7
	8-12	15
	3-7,15-19	17
29	2	12
	3	9
	4-7	16
	8-12,16-24	8
	13-15	14
	25-28	18
31	2,5-12,20-30	12
	3,4,13-19	22

In the first combination, the five factors have respective levels 2, 4, 3, 7, 5; in the second combination, the five factors have respective levels 3, 7, 5, 6, 2, etc.

When the solution space is large, it is desirable to sample more points for a better coverage. In principle, we can apply the uniform array with a larger number of levels. However, only the uniform arrays with at most 31 levels have been tabulated in Table 1, and it is very time consuming to compute the larger uniform arrays. To bypass this difficulty, we divide the solution space into multiple subspaces, and then apply the uniform array to sample some points in each subspace.

Denote that $\Omega = [l, u] = \{x | l \leq x \leq u, l, u \in R^n\}$ is the solution space, we divide $[l, u]$ into subspaces $[l(1), u(1)], [l(2), u(2)], \ldots, [l(S), u(S)]$, where the design parameter S can be assumed to be the values 2, or 2^2, or 2^3, etc. First, we divide the solution space into two subspaces as follows. We select the dimension with the largest domain and divide the solution space into two equal subspaces along this dimension. Then we divide the two subspaces into four subspaces as follows. For any subspace, say $[l(1), u(1)]$, we select the dimension with the largest domain, and then divide the two subspaces along this dimension into four equal subspaces. We repeat this step in a similar manner, until the solution space has been divided into S subspaces. The details are as follows:

Algorithm 3.1 (Dividing the Solution Space)**.**

Step 1. Let $a = l$ and $z = u$. Repeat the following computation $\log_2 S$ times: select the sth dimension such that $z_s - a_s = \max_{1 \leq i \leq N} \{z_i - a_i\}$, and then compute $z_s = (a_s + z_s)/2$.

Step 2. Compute $\delta_i = z_i - a_i$ and $n_i = (u_i - l_i)/\delta_i$, for all $i = 1, 2, ..., N$. Then compute the subspace $[l(k), u(k)]$, for all $1 \leq j_i \leq n_i$, and $1 \leq i \leq N$ as follows:

$$\begin{cases} l(k) = l + ((j_1 - 1)\delta_1, (j_2 - 1)\delta_2, ..., (j_N - 1)\delta_N) \\ u(k) = l + (j_1\delta_1, j_2\delta_2, ..., j_N\delta_N) \end{cases}$$

where $k = (j_1 - 1)n_2n_3...n_N + (j_2 - 1)n_3...n_N + ... + (j_{N-1} - 1)n_N + j_N$.
After dividing the solution space into S subspaces, we select a sample of points from each subspace as follows. Consider any subspace, say the kth subspace, and denote it by

$$[l(k), u(k)] = [(l_1(k), l_2(k), ..., l_N(k)), (u_1(k), u_2(k), ..., u_N(k)].$$

In this subspace, we quantize the domain $[l_i(k), u_i(k)]$ of x_i into Q_0 levels $\alpha_{i,1}(k)$, $\alpha_{i,2}(k)$, ..., $\alpha_{i,Q_0}(k)$, where the design parameter Q_0 is prime and $\alpha_{i,j}(k)$ is given by

$$\alpha_{i,j}(k) = \begin{cases} l_i(k), & j = 1 \\ l_i(k) + ((j-1)(u_i(k) - l_i(k))/(Q_0 - 1), & 2 \le j \le Q_0 - 1 \\ u_i(k), & j = Q_0 \end{cases} \tag{3.1}$$

In other words, the difference between any two successive levels is the same.
We let $\alpha_i(k) = (\alpha_{i,1}(k), \alpha_{i,2}(k), ..., \alpha_{i,Q_0}(k))$.After quantization, the subspace consists of Q_0^N points. We apply the uniform array $U(N, Q_0)$ to sample the following Q_0 points:

$$\begin{cases} (\alpha_{1,U_{11}}(k), \alpha_{2,U_{12}}(k), ..., \alpha_{N,U_{1N}}(k)) \\ (\alpha_{1,U_{21}}(k), \alpha_{2,U_{22}}(k), ..., \alpha_{N,U_{2N}}(k)) \\ ... \\ (\alpha_{1,U_{Q_0 1}}(k), \alpha_{2,U_{Q_0 2}}(k), ..., \alpha_{N,U_{Q_0 N}}(k)) \end{cases} \tag{3.2}$$

We repeat the above steps for each of the N subspaces, so that we get a total of SQ_0 points.

3.2 Uniform Design Local Search Algorithm (UDA)

Step 1. x_k^* is the current best solution, and M is the set of all points that have been used so far, and Ω is the operating region, given a positive integer I.

Step 2. Find the better point than the current best solution x_k^* in M, denoted as $x_{k+1}^{'}$, and go to step 5; else, $i = 1$, go to step 3.

Step 3. Generate N points by using uniform design and generate N points random strategy, and put them into S_1 and S_2 respectively, where $N = i \cdot n$, and n is selected in Table 1, if the number of generating points is not enough, then subdivide the operator region into several sub-region by using Algorithm 3.1, and then generated points by using the uniform design in each sub-region, until that the number of generating points is satisfied. Then go to step 4.

Step 4. Random select $[N/2]$ points in S_1, and $N - [N/2]$ points in S_2, and put them into M, find the better point than the current best solution x_k^* in M, denoted as $x_{k+1}^{'}$.
if $x_{k+1}^{'}$ is found
    go to step 5;
elseif $i \le I$,
    $i = i + 1$, and go to step 3;
else
    $x_{k+1}^* = x_k^*$, stop.

Step 5. Starting from the point $x_{k+1}^{'}$, minimize $f(x)$ by using a local optimization method to obtain x_{k+1}^*, stop.

4 Crossover Operator and Mutation Operator

4.1 Crossover Operator

In this section, the uniform design method [5] is used to design a new crossover operator. This operator can explore the search space effectively. The detail is as follows.

Denote $C_n = \{(x_1, ..., x_n)|0 \leq x_1, ..., x_n \leq 1\}$ and let $\{\alpha\}$ denote the decimal part of a real number α. Three widely used methods to generate q approximately uniformly distributed points in C_n are as follows [5], where q is a positive integer.

• Denote $(\gamma_1, ..., \gamma_n) = (\sqrt{p_1}, ..., \sqrt{p_n})$, where $p_1, ..., p_n$ are the first n positive primes. Then

$$\{(\{k\gamma_1\}, \{k\gamma_2\}, ..., \{k\gamma_n\})|k = 1 \sim q\} \tag{4.1}$$

is a set of q approximately uniformly distributed points in C_n.

• Let p be a positive prime. Denote $\lambda = 1/(n+1)$ and $(\gamma_1, ..., \gamma_n) = (p^\lambda, p^{2\lambda}, ..., p^{n\lambda})$, then

$$\{(\{k\gamma_1\}, \{k\gamma_2\}, ..., \{k\gamma_n\})|k = 1 \sim q\} \tag{4.2}$$

is a set of q approximately uniformly distributed points in C_n.

• Let p be a positive prime satisfying $p \geq 2n + 3$. Denote $\omega = 2\pi/p$ and $(\gamma_1, ..., \gamma_n) = 2(\cos(\omega), \cos(2\omega), ..., \cos(n\omega))$, then

$$\{(\{k\gamma_1\}, \{k\gamma_2\}, ..., \{k\gamma_n\})|k = 1 \sim q\} \tag{4.3}$$

is a set of q approximately uniformly distributed points in C_n.

Suppose that $x = (x_1, x_2, ..., x_n)$ and $y = (y_1, y_2, ..., y_n)$ are any two parents chosen for crossover. Let $l_i = \min\{x_i, y_i\}$ and $u_i = \max\{x_i, y_i\}$ for $i = 1 - n$. Now a new crossover operator is designed to generate q approximately uniformly distributed points in the set $[l, u] = \{x|l_i \leq x_i \leq u_i, i = 1 \sim n\}$.

Algorithm 4.1 (Crossover operator)**.**

1) Generate q approximately uniformly distributed points in C_n by one of the formulas (4.1), (4.2) and (4.3), and denote the set of these points by

$$\{(\{c_{k1}, ..., c_{kn}\})|k = 1 \sim q\} = \{(\{k\gamma_1\}, \{k\gamma_2\}, ..., \{k\gamma_n\})|k = 1 \sim q\}.$$

In simulation formula (4.2) is used and the parameter values are $p = 7$ and $q = 5$.

2) Generate q uniformly distributed points in set $[l, u]$ by $B = \{(\{b_{k1}, ..., b_{kn}\})|b_{kj} = l_j + c_{kj}(u_j - l_j), k = 1 \sim q\}$. Then the points in B are offspring of x and y.

4.2 Mutation Operator

For each individual generated by the local search, the mutation is executed on it. For example, suppose $z = (z_1, z_2, ..., z_n)$ is any individual generated by the local search, its offspring $z = (o_1, o_2, ..., o_n)$ of the mutation is as follows:

$$o = z + \triangle z, \triangle z = (\triangle z_1, \triangle z_2, ..., \triangle z_n)$$

where $\triangle z_i \sim N(0, \sigma_i)$, i.e., $\triangle z_i$ is a value of stochastic variable obeying Gaussian distribution with mean zero and variance σ_i^2, and $\triangle z_1, \triangle z_2, ..., \triangle z_n$ are mutually independent.

If $o_i, i \in (1, 2, ..., n)$ run out of the boundary, then $o_i - z_i$.

5 Smoothing with Uniform Design Evolutionary Algorithm(SUDEA)

Step 1. (Initialization) Given population size N, crossover probability $p_c > 0$, mutation probability $p_m > 0$, and a positive integer K. Generate N_1 points by using a uniform design method, and generate $N - N_1$ points randomly, put them into an initial population $POP(0)$. Let $k = 0$.

Step 2. (Local search) Find out the best individual $x_k^{'}$ in $POP(k)$. Starting from the point $x_k^{'}$, minimize $f(x)$ by using a local optimization method BFGS to obtain x_k^*.

Step 3. (Fitness) Define the fitness function $P(x, x_k^*)$ by formula (1), then go to step 8.

Step 4. (Jump out of the local minimizer) Use UDA in section 3.2 to find a better solution of $P(x, x_k^*)$, denoted as x_{k+1}^*, $k = k + 1$, and go to step 3; if the x_{k+1}^* is not found, then go to step 5.

Step 5. (Crossover) Randomly choose $[p_c \times N/2]$ pairs of parents from $POP(k)$. For each pair, denote a and b, if $rand(0, 1) < p_m$, then generate offspring $a^{'}$ and $b^{'}$, where $a^{'} = \alpha a + (1 - \alpha)b$, $b^{'} = (1 - \alpha)a + \alpha b$, and $\alpha \in (0, 1)$; else use Algorithm 4.1 to generate offspring. The set of all these offspring is denoted as O_1.

Step 6. (Mutation) For each individual $z \in O_1$, use the mutation operator to get an offspring O. The set of all these offspring is denoted as O_2.

Step 7. (Selection) Select best $[N/2]$ individuals among $POP(k) \cup O_1 \cup O_2 \cup \{x_k^*\}$ to put into $POP(k + 1)$, and randomly select $N - [N/2]$ individuals among $POP(k) \cup O_1 \cup O_2$ to put into $POP(k + 1)$, $k = k + 1$, and go to step 2.

Step 8. (Termination) If $k > K$, and $|x_k^* - x_{k-1}^*| \le \varepsilon$, or the gradient of the fitness function $P(x, x_k^*)$ is zero, the algorithm stop, and output $x^* = x_k^*$; otherwise, go to step 4.

6 Numerical Experiments

6.1 Test Problems

In this section, the proposed algorithm is tested on problems 1-8 taken from [21].

Problem 1. $F1 = \sum_{i=1}^{n} -x_i \sin(\sqrt{|x_i|})$

Problem 2. $F2 = \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10)$

Problem 3. $F3 = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1$

Problem 4. $F4 = \frac{\pi}{n}\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1} z_i^2 \cdot [1 + 10\sin^2(\pi y_{i+1})] + z_n^2\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4)$,

where $y_i = 1+(x_i+1)/4$, $z_i = y_i-1$, and $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a, \\ 0, & -a \le x_i \le a, \\ k(-x_i - a)^m, & x_i < -a. \end{cases}$

Problem 5. $F5 = \frac{1}{10}\{10\sin^2(3\pi x_1) + \sum_{i=1}^{n-1} y_i^2 \cdot [1 + \sin^2(3\pi x_{i+1})] + y_n^2 \cdot [1 + \sin^2(2\pi x_n)]\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$,

where $y_i = x_i - 1$.

Problem 6. $F6 = \sum_{i=1}^{n}\{\sum_{j=1}^{n}(\chi_{ij}\sin\omega_j + \psi_{ij}\cos\omega_j) - \sum_{j=1}^{n}(\chi_{ij}\sin x_j + \psi_{ij}\cos x_j)\}^2$,

where χ_{ij} and ψ_{ij} are random integers in $[-100, 100]$, and ω_j is a random number in $[-\pi, \pi]$.

Problem 7. $F7 = \frac{1}{n}\sum_{i=1}^{n}(x_i^4 - 16x_i^2 + 5x_i)$

Problem 8. $F8 = \sum_{i=1}^{n-1}[100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2]$

6.2 Parameters Setting for SUDEA

The proposed algorithm SUDEA is executed 50 independent runs for each test problem. In experiments, we adopted the parameters as follows:
• Population size: $N = 20$.
• Crossover and mutation parameters: $p_c = 0.5$, $p_m = 0.1$.
• Parameters in Algorithms UDA and SUDEA: $I = 30$, $K = 50$, $N_1 = 10$, and $\varepsilon = 1.0e - 9$.
• Stopping criterion: when the best solution cannot be improved further in successive 50 generations, the execution of the algorithm is stopped.

6.3 Simulation Results

8 widely used and challenging benchmarks are selected to test the proposed algorithm SUDEA. F1-F8 are chosen from [21], and the basic properties of these 8 problems are listed in Table 2.

Table 2: Basic Characteristics of Test Functions F1-F8

No.	Search space	Globally minimal function value	Number of local minima
F1	$[-500, 500]^n$	-12569.5	NA
F2	$[-5.12, 5.12]^n$	0	NA
F3	$[-600, 600]^n$	0	NA
F4	$[-50, 50]^n$	0	NA
F5	$[-50, 50]^n$	0	NA
F6	$[-\pi, \pi]^n$	0	2^n
F7	$[-5, 5]^n$	-78.33236	2^n
F8	$[-5, 10]^n$	0	NA

In each table from Tables 2 to 4, "NA" represents the results are not provided in the related reference, and "−" means the experiment is not carried out on the related function. We choose 8 algorithms for comparison: Minimum-Elinination-Escape Memetic(MEEM) [6], Fast Evolutionary Programming with CauchyMutation (FEP) [25], Orthogonal Genetic Algorithm with Quantization (OGA/Q) [12], Hybrid Taguchi-Genetic Algorithm (HTGA) [19], Hybrid Estimation of Distribution Algorithm (EDA/L) [26], Evolutionary Programming with Adaptive LevyMutation (ALEP) [10], Comprehensive Learning Particle Swarm Optimizer (CLPSO) [14], Evolutionary Algorithmbased on the Level-set Evolution and Latin Squares (LEA) [21].

Table 3: the results of SUDEA

No.	n	M-fun	Best	Worst	M-best	std
F1	30	2.4904e+04	-12573.45226	-12573.45226	-12573.45226	2.4253e-012
F2	30	3.0426e+03	0	0	0	0
F3	30	2.9706e+03	0	0	0	0
F4	30	5.9948e+03	3.5875e-019	3.5875e-019	3.5875e-019	5.0753e-035
F5	30	4.7016 e+03	7.1593e-023	7.1593e-023	7.1593e-023	1.2391e-038
F6	30	47050.0	4.3867e-018	4.3867e-018	4.3867e-018	5.6653e-026
F7	100	2.8280e+03	-78.33233	-78.33233	-78.33233	7.5731e-012
F8	100	2.8140e+03	1.4378e-023	1.4378e-023	1.4378e-023	0

Each of the above algorithms is executed to solve some of the test functions, and the results are reported in related articles. We use these existing results for a direct comparison, and the results obtained are listed in Tables 3 and 4. The convergence rate of algorithms for solving test functions F1-F8 is shown in Figures 2-9, where horizontal axis shows the times of iterations and the vertical axis shows the fitness value of the original objective function.

The symbols used in Tables 3 and 4 are given as follows:
No.: the number of the test problems;
n: the dimension of the test problems;
M-fun: the total number of function and gradient evaluations of $f(x)$ and $P(x, x_k^*)$;
Best: the best function value in 50 runs;
Worst: the worst function value in 50 runs;
M-best: the mean best function values in 50 runs;
std: the standard deviation of function values of the best solutions found in 50 runs;

Table 4: Results and Comparisons

No.		SUDEA	MEEM	FEP	OGA/Q	HTGA	EDA/L	ALEP	CLPSO	LEA
	M-fun	**2.4904e+04**	49,783	900,000	302,116	163,468	52,216	150,000	–	287365
F1	M-best	**-12573.45226**	-12569.49	-12554.5	-12569.4537	-12569.46	-12569.48	-11469.2	–	-12569.4542
	Std	2.4253e-012	4.532e-06	52.6	6.447e-04	**0**	NA	58.2	–	4.831e-04
	M-fun	**3.0426e+03**	25,467	500,000	224,710	16,267	75,014	150,000	200,000	223,803
F2	M-best	**0**	1.635e-15	0.046	0	0	0	5.85	4.85e-10	2.103e-18
	Std	**0**	3.002e-15	0.012	0	0	NA	2.07	3.63e-10	3.359e-18
	M-fun	**2.9706e+03**	76,825	200,000	134,000	20,999	79,096	150,000	200,000	130,498
F3	M-best	**0**	1.954e-14	0.016	0	0	0	0.024	3.14e-10	6.104e-16
	Std	**0**	1.856e-14	0.022	0	0	NA	0.028	4.64e-10	3.001e-17
	M-fun	**5.9948e+03**	56,482	150,000	134,556	66,457	89,925	150,000	–	132,642
F4	M-best	3.5875e-019	6.860e-19	9.2e-06	6.019e-06	1.000e-06	**3.654e-21**	6.0e-06	–	2.482e-06
	Std	5.0753e-035	8.042e-19	3.6e-06	1.159e-06	**0**	NA	1.0e-06	–	2.276e-06
	M-fun	**4.7016e+03**	64,946	150,000	134,143	59,003	114,570	150,000	–	130,213
F5	M-best	**7.1593e-023**	7.341e-18	1.6e-04	1.869e-04	1.000e-04	3.485e-21	9.8e-05	–	1.734e-04
	Std	1.2391e-038	1.250e-17	7.3e-05	2.615e-05	**0**	NA	1.2e-05	–	1.205e-04
	M-fun	**4.7050e+04**	165,468	–	190,031	186,816	124,417	–	–	189,427
F6	M-best	**4.3867e-018**	1.001e-09	–	4.672e-07	5.869e-05	3.294e-08	–	–	1.627e-06
	Std	**5.6653e-026**	1.237e-09	–	1.293e-07	8.325e-05	NA	–	–	6.527e-03
	M-fun	**2.8280e+03**	95,977	–	245,930	216,535	153,116	–	–	243,895
F7	M-best	**-78.33233**	-78.3323	–	-78.3000296	-78.303	-78.31077	–	–	-78.31
	Std	7.5731e-012	5.469e-12	–	6.288e-03	**0**	NA	–	–	6.127e-04
	M-fun	**2.8140e+03**	99,532	–	167,863	60,737	128,140	–	200,000	168,910
F8	M-best	**1.4378e-023**	5.210e-06	–	0.752	0.7	4.324e-03	–	2.1e+01	0.5609
	Std	**0**	1.483e-05	–	0.114	0	NA	–	2.98e+00	0.1078

For the proposed algorithm SUDEA in a fixed problem, the BFGS method is used in the local minimization of the function $f(x)$ and the smoothing function $P(x, x_k^*)$, and the step length is determined by using the *Armijo* condition.

It can be seen from Tables 3 and 4 that the proposed algorithm has the following advantages:

1. A better solution is obtained

In SUDEA, the uniform design is helpful to find a better global optimum solution more quickly in the 50 runs. For example, in Table 4, for problems 1,2,3,5,6 and 8, better optimal solutions are obtained by SUDEA than those obtained by other algorithms.

2. More effective and efficient

The optimal solutions of all test problems can be found by SUDEA. This indicates the effectiveness of SUDEA. In addition, it can be seen from Table 4 that whether the calculation accuracy of function value, or the total number of function and gradient evaluations of $f(x)$ and $P(x, x_k^*)$ used by SUDEA is fewer than that those used by other algorithms in Table 4. This indicates SUDEA is more efficient.

3. More stability

It can be seen from column *Std* in Tables 3 and 4 that the *Std* for the successful runs for all problems are very close to zero, which shows that SUDEA is stable. In addition, it can be seen from columns *M-best* and *Std* that difference between *M-best* and *Best* is small. This also indicates SUDEA is stable and robust to the initial points and parameter variation.

4. Applicable to multidimensional problems

In Table 3, SUDEA is tested on F1-F8 with different dimensions, and the numerical results indicate that SUDEA can find optimal solutions for different dimension problems. Thus SUDEA is suitable for solving multidimensional problems.

5. Fast convergence to the optimal solution

It can be seen from Figures 2 to 9 that the proposed algorithm can quickly converge to the optimal solution.

7 Conclusions

The smoothing function method is an approach to find the global minima of multi-modal functions. The existing smoothing functions are often nondifferentiable at some point in search domain. In this paper, first, a smoothing function with two parameters was designed, which was continuously differentiable, and which could eliminate all such local optimal solutions worse than the best solution found so far. Second, this smoothing function could keep the local optimal solutions unchanged in the region in which the values of the original function were not worse than its value at the best solution found so far. Third, by making use of the properties of the smoothing function, the uniform design search technique was properly combined into the algorithm, which would make the proposed algorithm converge much faster. Based on all these, a novel effective evolutionary algorithm with uniform design scheme was proposed, and the algorithm was numerically stable. At last, the numerical simulations for several standard benchmark problems were made and the simulation results showed that the proposed algorithm was very effective and efficient.

References

[1] P. Basso, Iterative methods for the localization of the global maximum, *SIAM Journal on Numerical Analysis* 19 (1982) 781–792.

[2] L. Bai, J.Y. Liang, C.Y. Dang and F.Y. Cao, A cluster centers initialization method for clustering categorical data, *Expert Syst. Appl.* 39 (2012) 8022–8029.

[3] F.H. Branin, Jr, Widely convergent methods for finding multiple solutions of simultaneous nonlinear equations, *IBM Journal of Research Developments* 16 (1972) 504–522.

[4] C.Y. Dang, W. Ma and J.Y. Liang, A deterministic annealing algorithm for approximating a solution of the min-bisection problem, *Neural Networks* 22 (2009) 58–66.

[5] K. Fang and Y. Wang, *Number-Theoretic Methods in Statistics*, London, U.K.: Chapman & Hall, 1994.

[6] L. Fan and Y. Wang, A Minimum-Elimination-Escape Memetic Algorithm for Global Optimization: MEEM. *International Journal of Innovative Computing, Information and Control* 7 (2012) 3689–3703.

[7] R.P. Ge, A filled function method for finding a global minimizer of a function of several variables, *Mathematical Program.* 46 (1990) 191–204.

[8] R.P. Ge and Y.F. Qin, A class of filled functions for finding global minimizers of a function of several variables, *Journal of Optimization Theory and Applications* 54 (1987) 241–252.

[9] C.R. Hicks, *Fundamental Concepts in the Design of Experiments*, New York: Saunders, 1993.

[10] C.Y. Lee and X. Yao, Evolutionary programming using mutations based on the Lvy probability distribution. *IEEE Transactions on Evolutionary Computation* 8 (2004) 1–13.

[11] Y.W. Leung and Y.P. Wang, Multiobjective programming using uniform design and genetic algorithm, *IEEE Transactions on Systems, Man, and Cybernetics, Part C* 30 (2000) 293–304.

[12] Y.W. Leung and Y.P. Wang, An orthogonal genetic algorithm with quantization for global numerical optimization. *IEEE Transactions on Evolutionary Computation* 5 (2001) 41–53.

[13] A.V. Levy and A. Montalvo, The tunneling algorithm for the global minimization of functions, *SIAM Journal on Scientific and Statistical Computing* 6 (1985) 15–29.

[14] J. Liang, A. Qin, P. N. Suganthan et al., Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE Transactions on Evolutionary Computation* 10 (2006) 281–295.

[15] R. Mladineo, An algorithm for finding the global maximum of a multimodal, multivariate function, *Math. Program* 34 (1986) 188–200.

[16] D.C. Montgomery, *Design and Analysis of Experiments*, NewYork: Wiley, 1991.

[17] K.E. Parsopoulos and M.N. Vrahatis, On the computation of all global minimizers through particle swarm optimization. *IEEE Transactions on Evolutionary Computation* 8 (2004) 211–224.

[18] J. Snyman and L. Fatti, A multi-start global minimization algorithm with dynamic search trajectories. *Journal of Optimization Theory and Applications* 54 (1987) 121–141.

[19] J.T. Tsai, T.K. Liu and J.H. Chou, Hybrid Taguchi-genetic algorithm for global numerical optimization. *IEEE Transactions on Evolutionary Computation* 8 (2004) 365–377.

[20] Y.Y. Wang, A uniform enhancement approach for optimization algorithms: smoothing function method, *International Journal of Pattern Recognition and Artificial Intelligence* 24 (2010) 1111–1131.

[21] Y. Wang and C. Dang, An evolutionary algorithm for global optimization based on level-set evolution and Latin squares. *IEEE Transactions on Evolutionary Computation* 11 (2007) 579–595.

[22] Y. Wang and K. Fang, A note on uniform distribution and experimental design. *Ke Xue Tong Bao* 26 (1981) 485–489.

[23] Y. Wang and D. Liu, A global optimization evolutionary algorithm and its convergence based on a smooth scheme and line search, *Chinese Journal of Computers* 29 (2006) 670–675.

[24] D. Yang and S.J. Flockton, Evolutionary algorithms with a coarse-to-fine function smoothing, *in Proc. of IEEE Intern. Conf. Evolutionary Computation, Perth, Australia* 2 (1995) 657–662.

[25] X. Yao, Y. Liu and G. Lin, Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation* 3 (1999) 82–102.

[26] Q. Zhang, J. Sun, E. Tsang et al., Hybrid estimation of distribution algorithm for global optimization. *Engineering Computations* 21 (2004) 91–107.

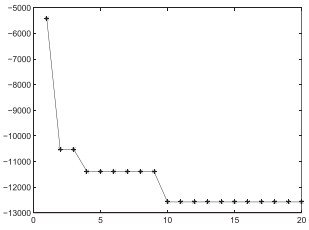

Figure 2: The convergence curve of fitness value over iterations for F1

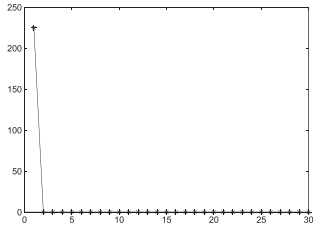

Figure 3: The convergence curve of fitness value over iterations for F2

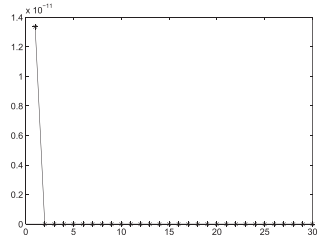

Figure 4: The convergence curve of fitness value over iterations for F3

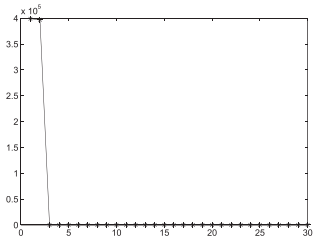

Figure 5: The convergence curve of fitness value over iterations for F4

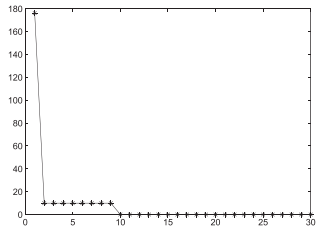

Figure 6: The convergence curve of fitness value over iterations for F5

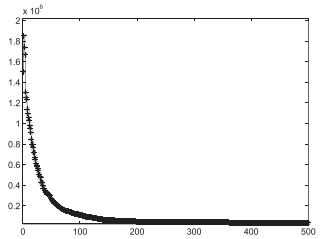

Figure 7: The convergence curve of fitness value over iterations for F6

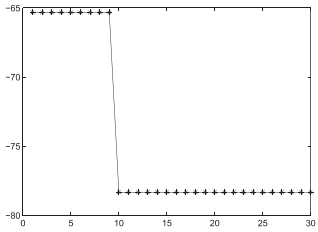

Figure 8: The convergence curve of fitness value over iterations for F7

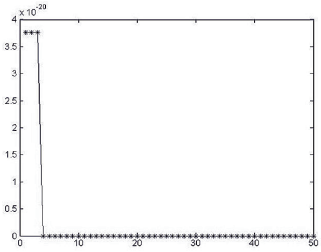

Figure 9: The convergence curve of fitness value over iterations for F8

Manuscript received 2 October 2012
revised 22 May 2013
accepted for publication 11 June 2013

Fei Wei
School of Computer Science and Technology, Xidian University
Xi'an, 710071, China
E-mail address: feiweixjf@gmail.com

Yuping Wang
School of Computer Science and Technology, Xidian University
Xi'an, 710071, China
E-mail address: ywang@xidian.edu.cn

Zhiqing Meng
College of Business and Administration
Zhejiang University of Technology
Zhejiang 310023, China
E-mail address: mengzhiqing@zjut.edu.cn