2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

A Niching Two-Layered Differential Evolution with Self-adaptive Control Parameters

Yongxin LUO, Sheng HUANG and Jinglu HU

***Abstract*— Differential evolution (DE) is an effective and efficient evolutionary algorithm in continuous space. The setting of control parameters is highly relevant with the convergence efficiency, and varies with different optimization problems even at different stages of evolution. Self-adapting control parameters for finding global optima is a long-term target in evolutionary field. This paper proposes a two-layered DE (TLDE) with self-adaptive control parameters combined with niching method based mutation strategy. The TLDE consists of two DE layers: a bottom DE layer for the basic evolution procedure, and a top DE layer for control parameter adaptation. Both layers follow the procedure of DE. Moreover, to mitigate the common phenomenon of premature convergence in DE, a clearing niching method is brought out in finding efficient mutation individuals to maintain diversity during the evolution and stabilize the evolution system. The performance is validated by a comprehensive set of twenty benchmark functions in parameter optimization and competitive results are presented.**

I. Introduction

Differential evolution (DE), is a well-known simple yet powerful population-based stochastic search technique, which is an efficient and effective global optimizer in the continuous searching domain[1][2]. Since it has presented good convergence and easy modeling features in many real-world cases, DE has been widely studied on its performance improvements and its applications to complicated problems in recent years.

Although DE is a stochastic population based evolution algorithm, appropriate control parameters—amplification factor F and crossover rate CR, will bring good evolution efficiency and result in better solution[3]. Therefore, choosing suitable control parameter values is, commonly, a problem dependent task. In the conventional DE algorithm, the control parameters F and CR are constants during the evolution, while the setting for control parameters depends on problems. The trail-error method applied in tuning control parameters requires multiple optimization runs, which is obviously time consuming. Moreover, during the evolution of iteration, different population coupled with different parameter settings may be required in order to achieve the best performance[4], especially dealing with multimodal functions.

Therefore, researchers have developed several techniques to avoid manual tuning of the control parameters. For example, Janex Brest in [5] summarizes a Self-adaptive DE algorithm with self-adaptive control parameters, in which each individual of generation is extended with parameters F and CR. Sarker R $et\ al.$ proposed a DE-DPS method in [6], where population size NP is dynamically generated during evolution as well as control parameters F, CR. Most of existing methods for parameter control concentrate on the probability to choose random parameters. But Qin $et\ al.$ proposed a Strategy adaptation DE (SaDE) algorithm in [4], in the SaDE both mutation strategies and control parameters can be gradually self-adapted according to their previous experiences of generating promising solutions.

In order to eliminate the time consuming task in fine-tuning control parameters in DE, and to avoid adding too many parameters, we propose a two-layered DE (TLDE) with self-adaptive control parameters based on the previous performance. The TLDE composes of two DE layers: a top DE layer and a bottom DE layer. The top DE layer is a simplified DE without crossover operation, for control parameter adaptation. The bottom DE layer is a doubled, parallel DE with two set control parameters, for the basic evolution task.

There is a natural assumption that in a DE, better control parameters are more likely to generate better offspring to survive, which will also change with different regions during iteration process. However, due to the stochastic characteristics of DE, it is hard to evaluate and then select better control parameters in only one generation. Differ from other greedy strategies which self-adapting parameters by every generation, in this paper a new scheme is introduced, which is designed to evaluate and select good control parameters over a learning period consisting of L_p generations.

On the other hand, although self-learning DE eliminates time for finding appropriate control parameters, it still faces the problem of premature convergence during evolution. A simple yet popular explanation for the occurrence of premature convergence is the loss of diversity[7], which also influences the system stability.

In the literature, a self-adaptive DE known as JADE was proposed by J. Zhang in [8] which implemented a mutation strategy "DE/current-to-p-best" diversifies the population but guarantee the fast convergence property at the same time. Similarly, a dynamic self-adapting parameter F and CR called jDE described in [9], combined with a multi-population method to increase diversity and stabilize the searching procedure. Followed the idea to combine parameter control adaptation and diversity maintenance[10], in this paper we introduce a clearing niching method[11][12]to improve the performance of the TLDE.

The rest of this paper is organized as follows: Section II briefly describes conventional DE and clearing nich-

Yongxin Luo, Sheng Huang and Jinglu Hu are with the Graduate School of Information, Production and Systems, WASEDA University, 2-7 Hibikino, Wakamatsu-ku, Kitakyushu, Fukuoka, Japan. Email: carolyn.lyx@gmail.com; symbi_21@hotmail.com; jinglu@waseda.jp.

978-1-4799-1488-3/14/$31.00 ©2014 IEEE

ing method. Section III describes in detail the proposed TLDE enhanced with niching method. Section IV carries out numerical simulations to show the effectiveness of the proposed approach. Finally, SectionV gives discussions and conclusions.

II. Related Algorithms

A. Differential Evolution

DE is a population based iterative optimization algorithm[1]. Offspring are generated by perturbing the solutions with a scaled difference of selected population vectors, and update population by greedy strategy to remain better individuals.

1) $Initialization$ Classic DE begins by randomly initializing a population of NP, D-dimensional vectors X_i^g, $i = 1, 2, \ldots, NP$. We denote generations $g \in \{0, 1, \ldots FESmax\}$ in DE and the i^{th} vector of the population at the current generation as

$$X_i^g = [X_{i,2}^g, X_{i,2}^g, \ldots X_{i,D}^g] \tag{1}$$

where $i \in [1, N]$. Population size NP should be at least four and unchanged during the searching process[1].

2) $Mutation$ For each target vector X_i^g, a mutation vector V_i^g is generated by the basic "rand/1/bin" strategy

$$V_i^g = X_{r_1}^g + F(X_{r_2}^g - X_{r_3}^g),\ r_1 \neq r_2 \neq r_3 \neq i \tag{2}$$

where r_i is randomly chosen integers from $[1, NP]$.

3) $Crossover$ In order to increase the diversity of the perturbed parameter vectors, trail vectors generated by

$$U_{i,j}^g = \begin{cases} V_{i,j}^g & \text{if } rand_{i,j}(0,1) \leq CR_{i,j} \text{ or } j = j_{rand} \\ X_{i,j}^g & \text{otherwise.} \end{cases} \tag{3}$$

$rand_{i,j}$ is the j^{th} evaluation of a uniform random number generator with outcome within the range $[0, 1]$. j_{rand} is a randomly chosen index $\in 1, 2, \ldots, D$, which ensures that U_i^g gets at least one dimension from V_i^g

4) $Selection$ Choosing the survive one between target vector X_i^g and trail vector U_i^g, greedy criterion is used as follow equation:

$$X_i^{g+1} = \begin{cases} U_i^g & \text{if } f(U_i^g) < f(X_i^g) \\ X_i^g & \text{otherwise.} \end{cases} \tag{4}$$

B. Niching Method

Niching method is designed to overcome the premature convergence problem due to the loss of diversity. The mechanism of maintain diversity in niching method is keeping several sub-populations within search space during iteration[13]. Since niching method was proposed from 1995[10], various niching techniques derived from it[14][15][16]. This paper gives a short introduce of clearing niching method which is nearly as others.

1) $Principles$ A clearing niche is characterized by a limited amount of resources available for living individuals which share commonalities[11]. Instead of sharing the available resources among all individuals of a subpopulation, which is widely used, the clearing niching means only few best members in each subpopulation possess the limited resources.

Algorithm 1 Clearing Procedure

Input: clearing radius δ, population $\{X_{1,2,\ldots,NP}^g\}$
Output: winners set $\{X_c^g\}$, number of winners n

```
function CLEARING(δ, X_i^g)
    Sort fitness f(X_{1,2,...,NP}^g)
    for i = 1 → NP do
        if fitness f(X_i^g) > 0 then
            {X_c^g} ← X_i^g
            /*Save the winner of this niche*/
            for j = i + 1 → NP do
                if distance (X_i^G, X_j^g) ≤ δ then
                    fitness f(X_j^g) ← 0
                    /*Clear others of this niche*/
                end if
            end for
        end if
    end for
end function
```

2) $Procedure$ For better understanding, a brief pseudo code of clearing procedure is presented (Algorithm 1). Take population $\{X_{1,2,\ldots,NP}^g\}$ in g^{th} generation and radius δ as input, we can get the winner set$\{X_c^g\}$ of this iteration.

III. The Niching Two-layered DE

As shown in Fig.1, the TLDE consists of two layers: a top DE layer and a bottom DE layer. The bottom DE layer is a doubled basic DE for normal evolution procedure, while the top DE layer is a simplified DE for self-adapting control parameters.

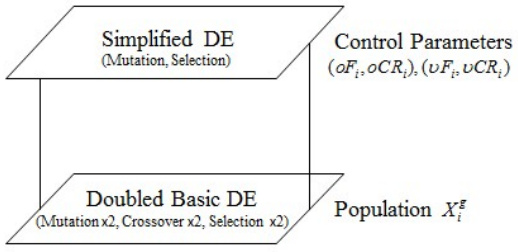

Fig. 1. Structure of the two-layered DE

A. The doubled basic DE

The doubled basic DE is a conventional DE described in Section II.A, evolving parallel with two sets of control parameters. The evolution is divided into multiple evolution periods called learning periods used by the simplified DE in the top layer for control parameter evolution. Each learning period consists of L_p generations with two set of fixed control parameters from the simplified DE in the top layer.

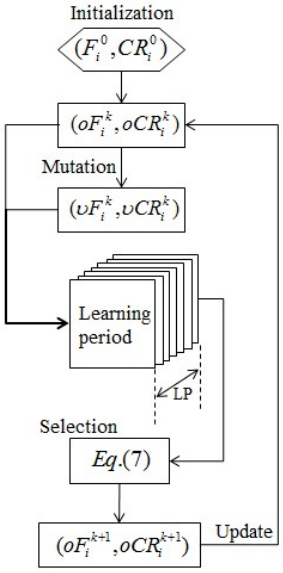

Fig. 2. Evolution of the simplified DE for control parameter adaptation

B. The simplified DE

As shown in Fig.2, the simplified DE is a conventional DE without crossover operation, for control parameter adaptation. The control parameters are evaluated over the learning period by the basic DE in the bottom layer.

1) *Initialization*

In order to realize a self-adaptation of control parameters by DE, we create a population of control parameters by assigning a set of control parameters to each individual of the basic DE in the bottom layer, which are randomly initialized. As shown in Fig.3, X_i^g denotes the i individual in the gth generation, while F_i^g and CR_i^g the corresponding control parameters.

X_1^G	F_1^G	CR_1^G
X_2^G	F_2^G	CR_2^G
⋮	⋮	⋮
X_{NP}^G	F_{NP}^G	CR_{NP}^G

Fig. 3. Individuals and the corresponding control parameters

2) *Mutation*

The mutation is performed for generating new control parameters.

$$\begin{cases} \upsilon F_i^g = oF_i^g + \omega_i \cdot (F_{r1}^g - F_{r2}^g), \\ \upsilon CR_i^g = oCR_i^g + \omega_i \cdot (CR_{r1}^g - CR_{r2}^g). \end{cases} \tag{5}$$

where (oF_i^g, oCR_i^g) are control parameters updated from last learning period. r_1, r_2 $(r_1 \neq r_2)$ and ω_i are random values uniformly distributed in a range of (0,1).

By this step, we have got two sets of control parameters $[oF_i^g, oCR_i^g]$ and $[\upsilon F_i^g, \upsilon CR_i^g]$ assigned to each individual, see Fig.4. These two sets of control parameters are used by the basic DE during a learning period, and which set to be chosen for the next learning period depends on their performance.

X_1^G	$[oF_1^G, oCR_1^G]$	$[\upsilon F_1^G, \upsilon CR_1^G]$
X_2^G	$[oF_2^G, oCR_2^G]$	$[\upsilon F_2^G, \upsilon CR_2^G]$
⋮	⋮	⋮
X_{NP}^G	$[oF_{NP}^G, oCR_{NP}^G]$	$[\upsilon F_{NP}^G, \upsilon CR_{NP}^G]$

Fig. 4. Two sets of control parameters generated by mutation

3) *Selection*

It is a natural assumption that better control parameters are more likely to generate better offspring to survive, which inverse, good individuals are more likely generated by individuals with good parameters from last iteration. However, because of the stochastic characteristics of DE it is hard to evaluate a better control parameter in just one generation, as done in many existing self-adaptive DE algorithms. To solve this problem, instead of generating new control parameters in each generation, we introduce a learning period with L_p generations ($L_p = 15$ in the simulation), the control parameters remain fixed till the next learning period, and the one with better rate that evolves minimum outcomes will survive.

Moreover, previous parameter selection procedure apply greedy strategy and remain parameters of best individuals. Somewhat differently, we apply a probability P as success rate to evaluate the parameters. Only when the success rate P is higher than a threshold τ, can the parameters be saved. The reason we omit parameters generate better individuals with success rate under τ is that, a "spark" of good control parameters is unstable and has high probability lead to a local optimal.

According this scheme, we intend to keep both local and global search ability to generate potentially good mutation vectors throughout the evolution process. oP_i^k and υP_i^k represent the probability of generating ith individual as optimal value in the kth learning period, which are calculated as follows:

$$\begin{aligned} oP_i^k &= \textstyle\sum_{g=L_p*(k-1)}^{L_p*k} \{f(oU_i^g) \leq Min_i^g\}/L_p \\ \upsilon P_i^k &= \textstyle\sum_{g=L_p*(k-1)}^{L_p*k} \{f(\upsilon U_i^g) \leq Min_i^g\}/L_p \end{aligned} \tag{6}$$

where oP_i^k is the success evolve rate for (oF_i^k, oCR_i^k), and υP_i^k is for $(\upsilon F_i^k, \upsilon CR_i^k)$.

By comparing the success evolve rate, the better one with probability higher than τ ($\tau = 0.3$) will be selected to replace $(oF_i^{k+1}, oCR_i^{k+1})$. For each individual, the pair of parameters (oF_i^k, oCR_i^k) in the k^{th} learning period updates at the same time, according to following equations:

$$(oF_i^{k+1}, oCR_i^{k+1}) = \begin{cases} (oF_i^k, oCR_i^k) & \text{if} \quad oP_i^k \geq Max_i^k \\ (\upsilon F_i^k, \upsilon CR_i^k) & \text{if} \quad \upsilon P_i^k \geq Max_i^k \\ (F_\tau^k, CR_\tau^k) & \text{otherwise} \end{cases} \tag{7}$$

Max_i^k defines as $\max\{oP_i^k, \upsilon P_i^k, \tau\}$ to find the parameters with highest probability to generate good individuals. Since a certain success rate of parameters are required to ensure individual quality, as similar referred in [8], those control parameters with success rate less than τ will be regenerated :

$$\begin{cases} F_\tau^k = randc_i(\mu F^k, randn) \\ CR_\tau^k = randn_i(\mu CR^k, randn) \end{cases} \tag{8}$$

where $randc_i$ is Cauchy distribution and $randn_i$ is normal distribution. Parameters μF and μCR represent the location parameter of distribution.

$$\begin{cases} \mu F^k = (1-c) \times \mu F^0 + c \times mean_L(S_F^k) \\ \mu CR^k = (1-c) \times \mu CR^0 + c \times mean_A(S_{CR}^k) \end{cases} \tag{9}$$

where parameters μF^0 and μCR^0 are initialized location as 0.5. c is a positive constant between 0 and 1, which we set to 0.5. Eq.9 and Eq.10 are similar equations were used in [8]. $mean_A()$ is the usual arithmetic mean, S_{CR}^k is the set of successful crossover probabilities in current learning period. S_F^k is the set collect for the F of success evolved population during the learning period, and $mean_L()$ is the Lehmer mean:

$$mean_L S_F = \frac{\sum_{F \in S_F} F^2}{\sum_{F \in S_F} F} \tag{10}$$

In this way the control parameter (oF_i^{k+1}, CR_i^{k+1}) are self-learning and updating after every learning period by choosing better success rate between (oF_i^k, oCR_i^k) and $(\upsilon F_i^k, \upsilon CR_i^k)$. The kth learning period is shown in Algorithm 2.

Algorithm 2 Selection of parameters

Input: control parameters(oF_i^k, oCR_i^k)
Output: parameters for next period $(oF_i^{k+1}, oCR_i^{k+1})$
success rate$oP_i^k \leftarrow (oF_i^k, oCR_i^k)$, $\upsilon P_i^k \leftarrow [\upsilon F_i^k, \upsilon CR_i^k]$
for $i = 1 \rightarrow NP$ **do**
  if $oP_i^k > \upsilon P_i^k > \tau$ **then**
    $F_i^{k+1} \leftarrow oF_i^k$, $CR_i^{k+1} \leftarrow oCR_i^k$
  else if $\upsilon P_i^k > oP_i^k > \tau$ **then**
    $F_i^{k+1} \leftarrow \upsilon F_i^k$, $CR_i^{k+1} \leftarrow \upsilon CR_i^k$
  else
    $F_i^{k+1} \leftarrow randni(\mu F, rand(0,1))$
    $CR_i^{k+1} \leftarrow randni(\mu CR, rand(0,1))$
  end if
end for
/*Update backup set $\{\hat{X}_b^g\}$ and $B \leq NP$*/
$\mu F \leftarrow (1-c) \times \mu F + c \times mean_L(S_F^k)$
$\mu CR \leftarrow (1-c) \times \mu CR + c \times mean_A(S_{CR}^k)$

4) *Evolution of the basic DE*

Fig.5 shows the evolution of the basic DE from $X_i^g \rightarrow X_i^{g+1}$. The basic DE is a doubled and parallel one, in which each individual is assigned with two set of control parameters described in Fig.4. The two sets of control parameters $(\upsilon F_i^g, \upsilon CR_i^g)$ and (oF_i^g, oCR_i^g) are fixed in L_p generations of a learning period. After a learning period, the control parameters of each individual are evaluated with a success rate, which indicates whether and which set of the control parameters are suitable for the next learning period.

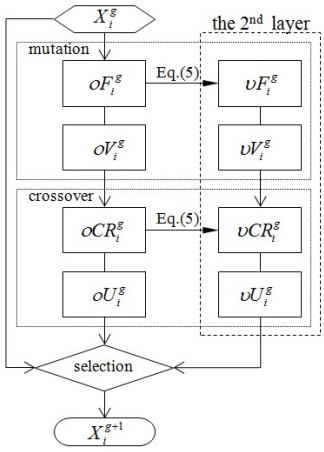

Fig. 5. Evolution of the basic DE

Searching procedure with both mutation vectors are computed by the mutation scheme "DE/rand/1/bin"[17]:

$$\begin{cases} oV_i^g = X_{C_r}^g + oF_i^g \cdot (X_{r_1}^g - \hat{X}_{r_2}^{g-1}) \\ \upsilon V_i^g = X_{C_r}^g + \upsilon F_i^g \cdot (X_{r_1}^g - \hat{X}_{r_2}^{g-1}) \end{cases} \tag{11}$$

where the index c_r,r_1,r_2 are integers chosen among $[1, NP]$, X_{c_r} is randomly chosen from winner set $\{X_c^g\}$ after clearing procedure, which we will describe later in detail. $X_{r_1}$is a random vector from current population $\{X_{1,2,\ldots,NP}^g\}$, while $\hat{X}_{r_2}^{g-1}$ is chosen from vectors of backup set from last learning period $\{X_b^g\}$, randomly.

As we have got a pair of trail vectors oV_i^g and υV_i^g, the next step crossover with oCR_i and υCR_i is described as follows:

$$oU_{i,j}^g = \begin{cases} oV_{i,j}^g & \text{if } rand_{i,j}(0,1) \leq oCR_{i,j}^k \text{ or } j = j_{rand} \\ X_{i,j}^g & \text{otherwise.} \end{cases} \tag{12}$$

$$\upsilon U_{i,j}^g = \begin{cases} \upsilon V_{i,j}^g & \text{if } \quad rand_{i,j}(0,1) \leq \upsilon CR_{i,j}^k \text{ or } j = j_{rand} \\ X_{i,j}^g & \text{otherwise.} \end{cases} \tag{13}$$

After mutation, crossover procedure in the second layer, those vectors get from equation (12)(13) supply a richer set to select target vectors:

$$X_i^{g+1} = \begin{cases} oU_i^g & \text{if } \quad f(oU_i^g) \leq Min_i^g \\ \upsilon U_i^g & \text{if } \quad f(\upsilon U_i^g) \leq Min_i^g \\ X_i^g & \text{otherwise.} \end{cases} \tag{14}$$

where Min_i^g defines as $\min\{f(oU_i^g), f(\upsilon U_i^g), f(X_i^g)\}$ for we need to find out the global minima. But if the trail vectors oU_i^g or υU_i^g is not the best one but better than X_i^g, it will be reserved to backup solution set $\{X_b^g\}$ and participate mutation in next generation by equation (11).

C. Niching mutation strategy

As we mentioned dominate individuals (target vectors $X_{r_1}^g$), are highly relevant with convergence efficiency. The

target vector in each iteration of searching should cover various niches and represent different niches.

Pseudo code of clearing procedure is shown in Algorithm 1. At the beginning of the g^{th} iteration, rank the fitness value of population in order, draw a circle C_1^g with center at the best fitness individual X_1^g and radius equal to δ ("Hamming" distance in simulation). The circle C_1^g may contain other individuals, but only remain the best individual as "the winner", then "clear" (delete) other individuals in this circle. So far we have got a first niche N_1^g with a winner X_1^g, the rest individuals are circling exactly the same as the first one. When there are no other individuals except winners, the clearing procedure completes and we get a smaller population group $\{X_c^g\}$.

By finding the efficient individuals of niches in each generation, we intended to speed up the convergence. Although we narrow down the domain of target individuals and in each iteration generate as X_{cr}^g, the step vector $X_{r_2}^g - X_{r_3}^g$ are still randomly chosen from population $\{X_i^g\}$ to maintain the variety of searching steps. This clearing procedure has no relationship with following mutation, crossover and selection procedures, it can be parallel executed with TLDE.

IV. Numerical Simulations

A. Benchmark Functions

To assure a fair comparison, twenty complicate benchmark functions from [18][19] are chose to test the performance of our proposed method. The benchmarks we adopt include several unimodal and multimodal functions, most of them have some global optima and many local optima in high dimension (30D) , which are difficult to find global minima especially shifted and rotated functions $f_{14} - f_{20}$. Benchmark functions are shown in Table I, for a intuitive result, the global minima of all these functions are all set to 0.

B. Comparing with conventional DE

Table II testify the performance of DE is sensitive to the choice of control parameters. For all those 20 benchmark functions, the success rate by fixed control parameters varies from different functions.

TABLE II
COMPARE TLDE AND CONVENTIONAL DE WITH FIXED F,CR

	0101	0503	0505	0509	0909	TLDE
$num\ of\ best$	0	3	5	2	1	14
$success\ rate$	46.0%	69.5%	69.8%	60.0%	50.3%	72.3%

Considering the commonly used parameter settings, we choose (F, CR) as (0.1,0.1), (0.5,0.3), (0.5,0.5), (0.5,0.9)and (0.9,0.9) to investigate the influence of DE parameters. In fairness, we compared to the proposed two-layer structured parameter learning DE without niching procedure, due to the parameters are the only part of adjustment. For conventional DE, results are not surprisingly vary from different parameters, while TLDE with self-learning parameters shows a significant improvement.

C. Comparing with Self-adaptive DE

In order to confirm the proposed method is extensive comparable, we compare the proposed method with some improved DE algorithms that use adaptive or self-adaptive parameter strategies. Specifically, the JADE algorithm proposed by Zhang in 2009[8], the jDE algorithm proposed by Brest in 2006[5], and the SDE algorithm proposed by Salman in 2007[20]. Moreover, we use the same population size(NP) as 100, also the same maximum function evolution(FEs) (5000 for JADE, TLDE and NTLDE, 10000 for jDE and SDE) for a fair comparison in solution quality.

Table III shows mean value and standard deviation of benchmark functions calculated by JADE, jDE, SDE, TLDE and NTLDE algorithm. Some results such as f_1, f_5, $f_9 - f_{13}$, f_{16}, f_{17} are omitted from the table, which because the results of these functions calculate by algorithms upon are similar to each other. Even there exist difference, our proposed method NTLDE is as good as the best result of other algorithms.

Since the population is randomly generated at the beginning, all twenty benchmark functions runs over 50 times by compared algorithms independently. In this table, we put the best two algorithms with best results in bold font. By observing the results table III listed, we can find out that although the result of NTLDE is not as precise as JADE algorithm for unimodal functions $f_1 - f_6$, but the average outcomes are still within the acceptable range. For multimodal function f_8, the proposed method get the best acceptable result. The advantages of proposed method are obvious when calculate $f_{14} - f_{20}$, which are complex shifted and rotated functions. Moreover, it's worth noticed that the two-layer structured DE has better performance when add with clearing procedure, especially in f_4, f_{15}, f_{19} and f_{20}. This better performance shows smaller average outcomes and smaller standard deviation, which confirms that the system is more stable as expected.

D. Comparing the success rate

Besides, it is necessary to compare the success rate of best result in several independent runs. For stable performance functions with small standard deviation we run about 30 times, other functions we run 60 times for each algorithm to ensure the probability. Different real world optimization problems has different acceptable error, when an optimal solution calculated by algorithms is within the acceptable error range, we count it as success. Any value beyond the error range in table I will be counted as a failure. In this case, we keep the results of all twenty benchmark functions.

The average success rate of TLDE is higher than jDE and SDE but lower than JADE, while NTLDE shows the best average success rate. For unimodal functions $f_1 - f_6$, it is easy to calculate the acceptable result in 100% by JADE and NTLDE . For multimodal functions $f_7 - f_{13}$ JADE performs a little bit better than NTLDE, in function f_7. When dealing with high dimensional shifted and rotated functions $f_{14} - f_{20}$, the proposed method shows a small improvement. Both TLDE and NTLDE make it possible to calculate the

TABLE I

THE TWENTY GLOBAL OPTIMIZATION BENCHMARK FUNCTIONS

Function	Search Space	Global Opt.x	Global Min	Accept	Name
f1	$[-100, 100]^D$	$\{0\}^D$	0	1×10^{-6}	Sphere[18]
f2	$[-100, 100]^D$	$\{0\}^D$	0	1×10^{-6}	Schwefel's2.21[18]
f3	$[-10, 10]^D$	$\{0\}^D$	0	1×10^{-6}	Schwefel's2.22[18]
f4	$[-100, 100]^D$	$\{0\}^D$	0	1×10^{-6}	Quadric[18]
f5	$[-100, 100]^D$	$\{0\}^D$	0	0	Step[18]
f6	$[-1.28, 1.28]^D$	$\{0\}^D$	0	1×10^{-2}	Noise[18]
f7	$[-10, 10]^D$	$\{0\}^D$	0	1×10^{-6}	Rosenbrock[18]
f8	$[-500, 500]^D$	$\{420.9687\}^D$	0	1×10^{-2}	Schwefel[18]
f9	$[-5.12, 5.12]^D$	$\{0\}^D$	0	1×10^{-6}	Rastrigin[18]
f10	$[-32, 32]^D$	$\{0\}^D$	0	1×10^{-6}	Ackley[18]
f11	$[-600, 600]^D$	$\{0\}^D$	0	1×10^{-6}	Griewank[18]
f12	$[-50, 50]^D$	$\{1\}^D$	0	1×10^{-6}	Generalized Penalized1.1[18]
f13	$[-50, 50]^D$	$\{1\}^D$	0	1×10^{-6}	Generalized Penalized1.2[18]
f14	$[-100, 100]^D$	o	390	1×10^{-2}	Shifted Rotated Rosenbrock[19]
f15	$[-600, 600]^D$	o	-180	1×10^{-2}	Shifted Rotated Griewank[19]
f16	$[-32, 32]^D$	o	140	1×10^{-2}	Shifted Rotated Ackley[19]
f17	$[-5, 5]^D$	o	-330	1×10^{-2}	Shifted Rastrigin[19]
f18	$[-5, 5]^D$	o	-330	1×10^{-2}	Shifted Rotated Rastrigin[19]
f19	$[-0.5, 0.5]^D$	o	90	1×10^{-2}	Shifted RotatedWeierstrass[19]
f20	$[-pi, pi]^D$	o	-460	1×10^{-2}	Schwefel's2.13[19]

All the test functions are with dimension D-30. Search space cross a wide range from 1 to 1200. The $f_{14} - f_{19}$ contains multiple individuals to the minimum. Accept is the acceptable error within the predefined minima value in 50 independent runs.

TABLE III

SOME EXPERIMENTAL RESULTS OF NICHING TWO-LAYER DE WITH OTHER SELF-ADAPTIVE DE FOR SELECTED BENCHMARK FUNCTIONS

Function		JADE	jDE	SDE	TLDE	NTLDE
$f2$	$mean$	2.04000E-39	2.93095E-41	1.53133E-38	**1.54301E-48**	**6.86622E-56**
	std	7.38514E-39	1.21711E-41	3.57502E-38	1.04599E-48	7.7398E-56
$f3$	$mean$	**6.60117E-70**	1.30181E-07	7.37197E-09	6.04986E-08	**2.12986E-17**
	std	1.3709E-69	5.02363E-07	1.18436E-08	4.24645E-08	2.99457E-17
$f4$	$mean$	**1.91456E-90**	0.001073383	9.276981772	1.123664137	**1.26035E-07**
	std	6.44987E-90	0.001130412	15.19739945	0.731622424	1.31483E-07
$f6$	$mean$	**0.000420116**	0.002599212	0.003565094	0.007139964	**0.001872842**
	std	0.00018554	0.000477625	0.000680896	0.001708176	0.000518219
$f8$	$mean$	7.896270801	0.000381827	0.000381827	**0.000381827**	**0.000381827**
	std	30.5806465	4.08274E-13	3.82746E-12	1.12226E-19	0
$f14$	$mean$	17.70968785	0.669693725	0.009761017	**0.000280592**	**0.002311603**
	std	37.08245913	1.414356385	0.000182752	0.000525148	0.007928373
$f15$	$mean$	0.000821399	0.001971457	0.000657152	**0.000657152**	**1.33227E-16**
	std	0.003181266	0.004081304	0.00254514	0.00254514	3.30945E-16
$f18$	$mean$	**22.11378903**	**35.76486345**	139.2023223	47.11462041	44.79645601
	std	5.948578981	10.41515826	16.89396392	8.516641994	10.93542304
$f19$	$mean$	25.07342387	**12.87762592**	33.53525092	23.03053569	**9.688993264**
	std	1.51756617	5.200141785	1.476459176	2.013521791	3.972020795
$f20$	$mean$	3057.418347	2048.220986	3692.160258	**1548.603555**	**997.1170613**
	std	2449.695545	1781.331182	2750.489196	1854.102311	1155.611716

Average over 50 independent runs, with 20 benchmark functions in total, this table shows 10 different results of NTLDE with other methods, while other results of NTLDE are as good as the best results of JADE, jDE, SDE, TLDE.

best result, although in a low probability 6.67%. By compare the average success rate from table IV, clearing niching procedure has been proved effective.

TABLE IV

ACCEPTABLE SUCCESS RATE IN ABOUT 30-60 INDEPENDENT RUNS

Function	JADE	jDE	SDE	TLDE	NTLDE
$f1$	100.00%	100.00%	100.00%	100.00%	100.00%
$f2$	100.00%	100.00%	100.00%	100.00%	100.00%
$f3$	100.00%	100.00%	100.00%	100.00%	100.00%
$f4$	100.00%	0.00%	0.00%	0.00%	100.00%
$f5$	100.00%	100.00%	100.00%	100.00%	100.00%
$f6$	100.00%	100.00%	100.00%	100.00%	100.00%
$f7$	100.00%	0.00%	80.00%	40.00%	86.67%
$f8$	93.33%	100.00%	100.00%	100.00%	100.00%
$f9$	100.00%	100.00%	80.00%	100.00%	100.00%
$f10$	100.00%	100.00%	100.00%	100.00%	100.00%
$f11$	100.00%	100.00%	100.00%	100.00%	100.00%
$f12$	100.00%	100.00%	100.00%	100.00%	100.00%
$f13$	100.00%	100.00%	100.00%	100.00%	100.00%
$f14$	66.67%	100.00%	100.00%	100.00%	93.33%
$f15$	93.33%	100.00%	100.00%	100.00%	100.00%
$f16$	0.00%	0.00%	0.00%	0.00%	0.00%
$f17$	100.00%	100.00%	86.67%	100.00%	100.00%
$f18$	0.00%	0.00%	0.00%	0.00%	0.00%
$f19$	0.00%	0.00%	0.00%	0.00%	0.00%
$f20$	0.00%	0.00%	0.00%	6.67%	6.67%
$average$	77.67%	70.00%	73.00%	72.33%	79.33%

The f_5 acceptable error is 0, while f_6, f_8 and $f_{14} - f_{19}$ acceptable error is $10e^{-2}$, others are all $10e^{-6}$.

E. Discussions

Our experiments are running on OS:$Win7$, 2 Inter Core $i5$ CPU: $2.5GHz$, RAM: $4GB$, Language: $Matlab2012$. Fig.6 shows the converge curves of algorithms above. Although TLDE and NTLDE runs double basic DE in candidate layer, i.e. 300 iterations of TLDE is actually 600 iterations of SDE, it still converge faster than jDE, SDE and JADE. The mutation strategy of JADE is "DE/Current-to-pbest" which is differ from other algorithms with classic mutation strategy "DE/rand/1/bin", which costs more time in each iteration.

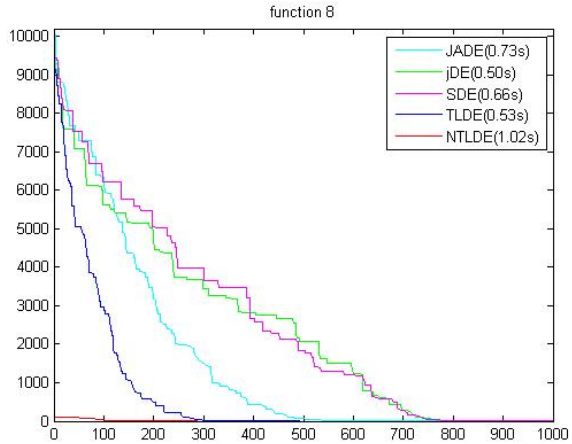

Fig. 6. Converge curves of function 8

Time consuming of each evolution iteration in converge curves varies from different algorithms, table V shows time consuming in function 8. The average time cost of TLDE is good enough with other self-adaptive DEs, while niching method calculating distance in each generation costs time, almost as twice consuming time as TLDE. Therefore, in general functions, the TLDE will be efficient enough to compute an acceptable optimum and the niching method is only required to optimize complex functions.

TABLE V

TIME CONSUMING OF EACH METHOD

F8	JADE	jDE	SDE	TLDE	NTLDE
$time(s)$	0.73	0.50	0.66	0.53	1.02

V. CONCLUSIONS

Parameter learning is beneficial for performance improvement of complex real-world applications. This paper presents an enhanced niching Two-Layered DE with self-adaptive control parameters, which eliminates the time consuming trial and error procedure for finding perfect parameters. A clearing procedure of niching method is proposed to maintain the diversity of the population, and prevent the premature convergence to local optimal.

The proposed modification is tested on commonly used benchmark problems for unconstrained optimization. Empirical results indicates that the proposed niching mutation strategy with two-layer DE structured parameter control is competitive and very promising. It exhibits a robust and stable behavior during the optimal-searching procedure.

Future target is to extend current work and improve this system, the entry point may start from different mutation strategies instead of the original "DE/rand/1/bin" we applied in this paper.

REFERENCES

[1] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[2] J. Brest, A. Zamuda, B. Boskovic, M. S. Maucec, and V. Zumer, "Dynamic optimization using self-adaptive differential evolution," in *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*. IEEE, 2009, pp. 415–422.

[3] R. Gämperle, S. D. Müller, and P. Koumoutsakos, "A parameter study for differential evolution," *Advances in intelligent systems, fuzzy systems, evolutionary computation*, vol. 10, pp. 293–298, 2002.

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

[5] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer, "Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems," *Evolutionary Computation, IEEE Transactions on*, vol. 10, no. 6, pp. 646–657, 2006.

[6] R. Sarker, S. Elsayed, and T. Ray, "Differential evolution with dynamic parameters selection for optimization problems," 2012.

[7] B. Chen and J. Hu, "An adaptive niching eda based on clustering analysis," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, 2010, pp. 1–7.

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

[9] S.-Z. Zhao, P. N. Suganthan, and S. Das, "Self-adaptive differential evolution with multi-trajectory search for large-scale optimization," *Soft Computing*, vol. 15, no. 11, pp. 2175–2185, 2011.
[10] S. W. Mahfoud, "Niching methods for genetic algorithms," *Urbana*, vol. 51, no. 95001, 1995.
[11] A. Petrowski, "A clearing procedure as a niching method for genetic algorithms," in *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*. IEEE, 1996, pp. 798–803.
[12] B. Sareni and L. Krahenbuhl, "Fitness sharing and niching methods revisited," *Evolutionary Computation, IEEE Transactions on*, vol. 2, no. 3, pp. 97–106, 1998.
[13] Z.-f. Liu and J.-h. Zhang, "Optimal planning of substation locating and sizing based on refined multi-team pso algorithm," in *Zhongguo Dianji Gongcheng Xuebao(Proceedings of the Chinese Society of Electrical Engineering)*, vol. 27, no. 1, 2007, pp. 105–111.
[14] M. G. Epitropakis, X. Li, and E. K. Burke, "A dynamic archive niching differential evolution algorithm for multimodal optimization," in *Evolutionary Computation (CEC), 2013 IEEE Congress on*. IEEE, 2013, pp. 79–86.
[15] K. A. De Jong, "Analysis of the behavior of a class of genetic adaptive systems," 1975.
[16] J.-P. Li, M. E. Balazs, G. T. Parks, and P. J. Clarkson, "A species conserving genetic algorithm for multimodal function optimization," *Evolutionary computation*, vol. 10, no. 3, pp. 207–234, 2002.
[17] H. A. Abbass, "The self-adaptive pareto differential evolution algorithm," in *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, vol. 1. IEEE, 2002, pp. 831–836.
[18] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *Evolutionary Computation, IEEE Transactions on*, vol. 3, no. 2, pp. 82–102, 1999.
[19] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization," *KanGAL Report*, vol. 2005005, 2005.
[20] A. Salman, A. P. Engelbrecht, and M. G. Omran, "Empirical analysis of self-adaptive differential evolution," *European Journal of operational research*, vol. 183, no. 2, pp. 785–804, 2007.