Multi-population Modified L-SHADE for Single Objective Bound Constrained Optimization

Yann-Chern Jou
Department of Computer Science and Information Engineering, National Taiwan Normal University, Taipei, Taiwan
danney9512@gmail.com

Shuo-Ying Wang
Department of Computer Science and Information Engineering, National Taiwan Normal University, Taipei, Taiwan
kevin168w@gmail.com

Jia-Fong Yeh
Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan
jiafongyeh@ieee.org

Tsung-Che Chiang
Department of Computer Science and Information Engineering, National Taiwan Normal University, Taipei, Taiwan
tcchiang@ieee.org

***Abstract*—In this paper, we extend a previous algorithm mL-SHADE by running the evolutionary process through multiple populations and adding dynamic control of mutation intensity and hyper-parameters. The whole population is partitioned into sub-populations by a random clustering method. Mutation intensity and hyper-parameters are adjusted based on the consumption of fitness function evaluations. Performance of the proposed algorithm is verified by ten benchmark functions in the CEC2020 Competition on Single Objective Bound Constrained Optimization. The results show the competitiveness of the proposed algorithm.**

Keywords—differential evolution, multi-population, clustering, adaptive, success history

I. Introduction

In recent years, evolutionary computation is widely applied in many fields for solving optimization problems. Differential evolution (DE) is one of the most popular evolutionary algorithms and shows high potential particularly in solving real-parameter optimization problems. As an evolutionary algorithm, DE searches for the optimal solution through repeating mutation, crossover, and selection in each generation. A standard DE has three main parameters, the population size (*NP*), the scaling factor (*F*), and the crossover rate (*CR*). The setting of these three key parameters has a great impact on the performance of DE. Therefore, developing a DE with adaptive control of parameters has been an important topic in the literature.

In CEC2020, there is a competition on Single Objective Bound Constrained Optimization [1]. The goal is to develop an algorithm to solve ten given numerical optimization problems. Solving a *D*-dimensional real-parameter optimization problem is to find the best decision vector $X = [x_1, x_2, \ldots, x_D]$ to minimize or maximize an given objective function $f(X)$. The bound constraints define the ranges, i.e. the maximal and minimal values, of all decision variables. Many adaptive DE algorithms have been proposed for real-parameter optimization in the last decade. In this paper we develop an algorithm based on a modified version of L-SHADE [2], which is a state-of-the-art evolutionary algorithm and the winner in the Competition on Real-Parameter Single Objective Optimization in CEC2014. The rest of the paper is organized as follows. Section II reviews the previous studies of DE with adaptive control and multi-population mechanisms. Section III elaborates the proposed multi-population mL-SHADE (mpmL-SHADE). Section IV presents experiments and results. Section V gives conclusions and future directions.

II. Literature Review

A. Success History Based Adaptive Differential Evolution

Most adaptive control mechanisms adjust parameter values according to feedback information obtained during the evolutionary process. When we generate an offspring and the offspring survives to the next generation, the parameter setting used to generate the offspring is usually regarded as "successful". It is natural to record and re-use the successful parameter values.

SaDE [3] is a representative adaptive DE that follows the above concept. It generates *CR* values by Gaussian distribution. These *CR* values are used in the crossover operations to generate new offspring. During a learning period (e.g. 25 generations) successful *CR* values are recorded, and the mean of these successful values will be taken as the mean of the Gaussian distribution in the next period. JADE [4] is another well-known DE that works in a similar manner. JADE controls both *CR* and *F* values, and it uses Gaussian distribution and Cauchy distribution to generate *CR* and *F*, respectively. In addition, JADE uses the cur-to-*p*best mutation strategy, in which top $100 \cdot p\%$ individuals in the population are regarded as *p*best.

SHADE [5] is an algorithm descendant of JADE. The new feature is that SHADE records multiple pairs of means of successful *CR* and *F* values. It enables SHADE to use different ranges of parameter values on different individuals in a generation. Every time an offspring is to be produced, SHADE randomly selects one pair of mean *CR* and mean *F* from the success memory. Then, actual *CR* and *F* values are generated through Gaussian and Cauchy distributions, respectively. In each generation, all successful *CR* and *F* values are recorded. Weighted arithmetic mean and weighted Lehmer mean of successful *CR* and *F* values will update the success memory in a round-robin way. The authors of SHADE later added a linear reduction mechanism of population size in a new version – L-SHADE [2]. They also added a terminal value to stop updating *CR* values and to keep the zero value of *CR*.

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

Brest *et al.* proposed iL-SHADE [6], which won the third place in a CEC2016 competition. It is an improved version of L-SHADE. The cur-to-*p*best mutation strategy is applied, but the value of p linearly decreases from p^{max} to p^{min} based on the consumption of fitness function evaluations. Moreover, the success memory update mechanism was modified; iL-SHADE considers both the old and the incoming mean values, not just replacing the old mean by the new mean. A pair of large CR and F values is kept in the memory to maintain the chance of using large CR and F values to generate offspring. In the next year, Brest *et al*. [7] proposed jSO, which was the second place in a competition in CEC2017. jSO applies a weighted version of cur-to-*p*best mutation strategy, named cur-to-*p*best-w. This strategy uses different weights to control the mutation strength. In addition, values of some parameters such as the success memory size and the initial memory values were changed.

Awad *et al.* [8] proposed another variant of L-SHADE, called L-SHADE-cnEpSin. In L-SHADE-cnEpSin, the mutation strategy and the control mechanism of CR are the same as those used in L-SHADE. The control mechanism of F is different and is divided into two phase. In the first phase, the value of F is generated by two sinusoidal functions, chosen by the learning mechanism in SaDE. In the second phase, the control mechanism is switched back to that used in L-SHADE. L-SHADE-cnEpSin was the 3rd place in a CEC2017 competition. In L-SHADE-RSP [9], again an extension of L-SHADE, a rank-based selection is added into the cur-to-*p*best mutation strategy. The proposed current-to-*p*best/r mutation selects the individuals as the *p*best in the probability proportional to their ranks. The parameter setting heuristics in jSO were also applied. It won the second place in a competition in CEC2018.

Yeh *et al*. proposed mL-SHADE [10], where several modifications were made to L-SHADE. They removed the CR terminal value to keep the capability of exploration. (The terminal value significantly increases the chance of using zero as the CR value, and zero CR means only one decision variable is changed.) They also invoked a memory perturbation mechanism when the history memory did not update for a period of generations to avoid stagnation of population. Besides, polynomial mutation was applied in a certain probability to increase population diversity. No algorithm dominated mL-SHADE in terms of solution quality and computational effort simultaneously in a competition in CEC2019.

B. Multi-population of DE

The concept of "multiple (sub-)populations" has been introduced to improve the performance of DE and particle swarm optimization (PSO) in several studies [11][12]. It has been shown that the multi-population method could be an effective approach to enhance the diversity and the search ability for DE to solve some optimization problems. These studies mainly partition the initial population into multiple equal-size smaller sub-populations. As the algorithm proceeds, the sub-populations evolve independently or cooperatively. Some mechanisms for exchanging information among sub-populations include migration, shuffle, split, and regrouping. It is important to maintain diversity and balance exploitation and exploration [13].

Zaharie proposed a multiresolution multi-population DE (MMDE) [14]. MMDE divides the population into equal-size sub-populations. A classic DE is applied to evolve each sub-population for a given number of generations. A migration process is carried out in a predefined probability to maintain the diversity and avoid premature convergence. When a sub-population converges, the found optima are collected into an archive. Then, the sub-populations will be re-initialized based on the archived solutions and re-start the evolutionary process.

Gao *et al*. proposed a cluster-based DE with a self-adaptive strategy for solving multimodal optimization problems [15]. They used a clustering partition strategy to form sub-populations and improved the performance of two DEs. The strategy randomly chooses an individual from the population as a "reference point". The individuals in the population closest to the reference point in terms of Euclidean distance in the decision space are clustered with the reference point one by one until the predefined size of a sub-population is reached. The above process is repeated to form the remaining sub-populations.

Bošković and Brest proposed a DE with a clustering mechanism for multimodal optimization [16]. In every generation, the algorithm clusters the whole population into several sub-populations and performs jDE [17] for each sub-population. There are two differences in its clustering method from that in [15]: first, it selects the best un-clustered individual as the reference point; second, it clusters the individuals with the reference point in the probability proportional to the Euclidean distance (instead of in a greedy manner). It also carries out extra mutation for the stagnant sub-populations, in which the best individual stops improving after a predefined number of generations. When half of sub-populations are stagnant, the population size is doubled by adding new randomly-generated individuals.

Bujok and Polaková proposed mSO [18] based on jSO using a parallel migration model. They initialized a number of sub-populations and ran jSO for each sub-population. Sub-populations are connected with a unidirectional ring topology. Periodically, the migration policy selects the best individual and some random individuals from each sub-population and uses them to replace the worst individual and some random individuals in the succeeding sub-population.

III. MULTI-POPULATION ML-SHADE (MPML-SHADE) ALGORITHM

A. Overview

In this paper, we extend our previous algorithm mL-SHADE by several improvements. We divide the population into sub-populations; we apply the polynomial mutation with two different parameter values for exploration and exploitation; we adjust several hyper-parameters dynamically based on the consumption of fitness function evaluations. We brief the main improvements here and will describe them in detail in the following sub-sections. Algorithm I gives the pseudo code of our multi-population mL-SHADE (mpmL-SHADE).

ALGORITHM I. MULTI-POPULATION ML-SHADE

Notations
C: the number of sub-populations
M_c^F, M_c^{CR}: the history memory of mean F, CR values of the c^{th} sub-population
$\boldsymbol{A}_c$: the archive of inferior solutions of the c^{th} sub-population
nfe_c^{ns}: the number of fitness evaluations since last improvement of the c^{th} sub-population
k_c: the index of the to-be-updated history memory of the c^{th} sub-population
pop: initial population
subpop: sub-populations
N^{init}: the size of the initial population
N^{subpop}: the size of a sub-population

```
01  for c = 1 to C do
02      Initialize values in M_c^F and M_c^CR
03      Set A_c = Ø, nfe_c^ns = 0, k_c = 1
04  end for
05  nfe = 0
06  pop = Initialize(N^init)
07  subpop = Partition(pop, C)
08  N^subpop = N^init / C
09  while the termination criterion is not satisfied do
10      for c = 1 to C do
11          subpop[c] = Evolve(subpop[c], A_c, M_c, k_c nfe, nfe_c^ns)
12      end for
13      Update N^subpop using the LPSR strategy by Eq. (7)
14      for c = 1 to C do
15          if N^subpop < |subpop[c]|
16              Resize subpop[c] by removing the worst individuals
17              Resize A_c by removing individuals randomly
18          end if
19      end for
20  end while
21  return the best solution in subpop
```

1) Multiple populations: We follow the clustering method in [15] to form multiple sub-populations. In line 7 in Algorithm I, a function "Partition" is called to separate the population *pop* into C sub-populations *subpop*. Pseudo code of this function will be given in Algorithm II. After initialization and partition, each sub-population evolves by the improved mL-SHADE independently until the terminating criterion is satisfied.

2) Polynomial mutation with two intensity: In mL-SHADE the polynomial mutation [19] was carried out in a predefined probability m_r. Here we apply the polynomial mutation when it is necessary – when multiple individuals have the same fitness value. In this way, we remove the hyper-parameter m_r. We apply the polynomial mutation with two intensity, controlled by two values of the parameter η. A small (resp. large) value of η leads to large (resp. small) modification [20]. We intend to use small η more frequenly at earlier stage of the evolutionary process for exploration and to use large η value more frequently at later stage for exploitation. Lines 20–26 in Algorithm III explains our mechanism.

3) Dynamic control of hyper-parameters: As we mentioned, in JADE and its descendants, the actual F values are generated by Cauchy distribution, which has two paramters. Most parameter contorl mechanisms adjust the first parameter, the location parameter, and fix the second one, the scale parameter. We found that the value of the scale parameter could also affect the algorithm performance. Thus, we adjust it by a linear scale parameter increment (LSPI) mechanism (see line 2 in Algorithm III). In mL-SHADE, the success memory is perturbed when the search process gets stuck for N^{stuck} continous generations. We keep the memory perturbation strategy but activate it in a probabilistic way. Again it helps to remove one more hyper-parameter N^{stuck}. (See line 33 in Algorithm III.)

B. Initialization

Our algorithm initializes the population by uniformly sampling random solutions in the feasible solution space, following the rules of the CEC2020 Single Objective Bound Constrained Optimization Competition. Then, we divide the whole population into C sub-populations by the Partition function, shown in Algorithm II. The Partition function randomly selects a reference point for each sub-population and then repeats assigning the nearest individual in the population to the reference point until the sub-population contains NP/C individuals.

ALGORITHM II. PARTITION FUNCTION

Function Partition(*pop*, C)
Output: *subpop*, C sub-populations, each containing equal number of individuals

Notations
pop: a population of individuals
C: the number of sub-populations
NP: the number of individuals in *pop*

```
01  NP = |pop|
02  for c = 1 to C do
03      Randomly select a reference point R_c from pop
04      subpop[c] = {R_c}
05      pop = pop \ {R_c}
06  end for
07  for c = 1 to C do
08      for j = 2 to NP/C do
09          Find the nearest individual x ∈ pop to R_c
10          subpop[c] = subpop[c] ∪ {x}
11          pop = pop \ {x}
12      end for
13  end for
14  return subpop
```

C. Mutation Strategy

The mutant vector v_i is generated by using the current-to-*p*best/1 mutation strategy [4] shown in (1). This strategy selects one of the top $100{\cdot}p\%$ individuals in the population as x_{pbest}. Comparing with a single best individual in the current-to-best/1 strategy, multiple *p*best individuals help to search toward multiple directions.

$$v_i = x_i + F \cdot (x_{pbest} - x_i) + F \cdot (x_{r1} - x_{r2}) \tag{1}$$

When the value $v_{j,i}$ of dimension j in the mutant vector v_i goes outside of the feasible range $[x_j^{min}, x_j^{max}]$, we repair the value by (2).

$$v_{j,i} = \begin{cases} (x_j^{min} + x_{j,i})/2, & if\ v_{j,i} < x_j^{min} \\ (x_j^{max} + x_{j,i})/2, & if\ v_{j,i} > x_j^{max} \end{cases} \tag{2}$$

We implement the current-to-*p*best/1 strategy with archive. At the beginning of evolution, the archive A is empty. During evolution, the replaced (i.e. inferior) solutions are added into A.

In (1), x_{r2} is randomly chosen from the union of the population and the archive. We set the same size for the archive and the population. When the size of A exceeds the limit, one individual is randomly deleted.

ALGORITHM III. EVOLVE FUNCTION

Function Evolve(*pop, A, M, k, nfe, nfe*^{ns})
Output: The result of evolving *pop* after one generation

Notations
pop: a population of individuals
$\boldsymbol{A}$: the archive of inferior solutions
k: the index of the to-be-updated history memory
M_F, M_{CR}: the history memory of mean *F*, *CR* values
nfe: the number of fitness evaluations already consumed
nfe^{ns}: the number of fitness evaluations since last improvement
S_F, S_{CR}, $S_{\Delta f}$: the archive of successful *F*, *CR*, and fitness improvement
$scale_{min}$, $scale_{max}$: minimum and maximum values for *scale*
MAX_NFE: the maximum number of function evaluations
NP: the number of individuals in *pop*
H: the size of history memory
randn, randc, *randu*: normal, Cauchy, and uniform distribution
D: problem dimension

```
01  S_F = Ø, S_CR = Ø, S_Δf = Ø
02  scale = scale_min + (scale_max − scale_min) × (nfe / MAX_NFE)
03  newpop = {}
04  for i = 1 to NP do
05    r = a random integral value in [1, H]
06    F_i = randc(M_F,r , scale), CR_i = randn(M_CR,r , 0.1)
07    Repair F_i and CR_i
08    x_i = pop[i]
09    Generate a mutant vector v_i by Eq. (1)
10    Generate a trial vector u_i by Eq. (3)
11    if f(u_i) ≤ f(x_i) then
12      newpop = newpop ∪ {u_i}
13      S_F = S_F ∪ {F_i}, S_CR = S_CR ∪ {CR_i}, S_Δf = S_Δf ∪ {Δf_i}
14      A = A ∪ {x_i}
15    else
16      newpop = newpop ∪ {x_i}
17    end if
18    nfe = nfe + 1
19  end for
20  while two individuals x_i and x_j have the same fitness value do
21    if randu(0.0, 1.0) ≤ nfe/MAX_NFE then
22      x_j = PolynomialMutation(x_j,1.0/D, 20)
23    else
24      x_j = PolynomialMutation(x_j,1.0/D, 5)
25    nfe = nfe + 1
26  end while
27  if S_CR ≠ Ø and S_F ≠ Ø then                    // Update Memory
28    Update the k^th memory in M_CR and M_F by Eq. (4)-(6);
29    k = k mod H +1
30    nfe^ns = 0
31  else
32    nfe^ns += NP
33    if randu(0.0, 1.0) ≤ nfe^ns/ nfe) then        // Memory Perturbation
34      M_CR,k = randu(0.0, 1.0)
35      M_F,k = randu(0.0, 1.0)
36      nfe^ns = 0
37      k = k mod H +1
38    end if
39  end if
40  return newpop
```

D. Crossover

The trial vector u_i is generated by binomial crossover, as shown in (3). The function *rand*[0,1) returns a random real value between 0 and 1 (excluding 1), and j_{rand} is a random integral value in [1, *D*].

$$u_{j,i} = \begin{cases} v_{j,i} & if\ rand[0,1) \le CR\ \ or\ \ j = j_{rand} \\ x_{j,i} & otherwise \end{cases} \tag{3}$$

E. Selection

The target vector x_i is replaced by the trial vector u_i when x_i is not better than u_i. Meanwhile, the successful *F*, *CR* values, and the improved fitness are stored. The target vector is added to the archive. Lines 11–17 in Algorithm III explain the details.

F. Dynamic Control of Hyper-Parameters

In line 2 of Algorithm III, we adjust the *scale* parameter of the Cauchy distribution for generating *F* by a linear increment mechanism. The values of $scale_{min}$ and $scale_{max}$ are predefined. *MAX_NFE* is a constant, representing the maximum number of fitness function evaluations allowed. Their values will be listed in the next section. The variable *nfe* denotes the number of fitness function evaluations that has been consumed. Simply speaking, the *scale* value used in line 6 increases from $scale_{min}$ linearly to $scale_{max}$ as more and more fitness function evaluations are consumed. The effect is that we gradually increase the probability of generating large *F* values during the evolution.

In lines 21–24, the ratio of *nfe* to *MAX_NFE* is taken to control the probability of selecting two intensity values of polynomial mutation. When *nfe* is small, it is more likely to do polynomial mutation with η value equal to 5, which is more exploratory. As *nfe* gets larger, it becomes more likely to do polynomial mutation with η value equal to 20, which tends to search in a more focused region.

Lines 32–38 present the memory perturbation mechanism. The variable *nfe*^{ns} accumulates the number of fitness function evaluations after last improvement. When its value is larger, it means the sub-population gets stuck for a longer time, and thus we give a higher probability to perturb the memory to try other mean *CR* and mean *F* values.

G. Success History Memory and Population Size Reduction

The history memory M_F and M_{CR} store potential mean values of *F* and *CR*. In Algorithm III, lines 5–7 present how to use M_F and M_{CR}. Since we remove the *CR* terminal value, we still generate the actual *CR* values by Gaussian distribution when M_{CR} is equal to zero. However, the zero-mean Gaussian distribution generates negative *CR* values in probability 0.5. The original repair mechanism of L-SHADE repairs negative *CR* values to zero, and thus it still leads to a very high chance to use zero *CR* value. Thus, we propose to use another repair mechanism: when *CR* is below zero, we set it to the absolute value; when *CR* is greater than one, we set it to one. The repair mechanism of *F* is the same as that of L-SHADE. When the value of *F* is negative, we take a new value by Cauchy distribution again.

We update the history memory in the same way as L-SHADE does, shown in (4)-(6). In (5) and (6), S may denote S_{CR} or S_F. S_k denotes the k^{th} value in S.

$$\Delta f_k = |f(u_i) - f(x_i)| \tag{4}$$

$$w_k = \frac{\Delta f_k}{\sum_{l=1}^{|S|} \Delta f_l} \tag{5}$$

$$mean_{wL}(S) = \frac{\sum_{k=1}^{|S|} w_k \cdot S_k^2}{\sum_{k=1}^{|S|} w_k \cdot S_k} \tag{6}$$

The linear population size reduction mechanism (LPSR) in L-SHADE is implemented, as shown in (7).

$$NP = round\left(\left(\frac{N^{min} - N^{init}}{MAX\_NFE}\right) \times nfe + N^{init}\right) \tag{7}$$

H. Terminating Criterion

Our algorithm stops when the maximum number of fitness function evaluations (*MAX_NFE*) is reached. In the CEC2020 competition, functions of different dimensions are given different computational budget [1], which are given in Table III.

IV. Experiments and Results

A. Benchmark Functions

There are ten test functions f_1–f_{10} with four different dimensions: D = 5, 10, 15, and 20 in the CEC2020 Single Objective Bound Constrained Optimization Competition [1]. Because of the function characteristics, functions f_6 and f_7 are excluded when D is 5. The search ranges of decision variables are limited in [-100, 100] for all functions.

B. Parameter Setting

The proposed mpmL-SHADE is based on L-SHADE, and thus we need to set values for all parameters of L-SHADE. The first six rows of Table I give the meaning and values of them. Our algorithm requires some extra parameters: the number of sub-populations C, the minimum and maximum values of scale $scale_{min}$ and $scale_{max}$ for linear increment of scale in Cauchy distribution, and p_m and η for polynomial mutation. Values of N^{min}, H, r^{arc}, p, and initial success memory were set by the same values in L-SHADE [5]. Values of other parameters were set based on the experience on pilot runs.

TABLE I. Value of Parmeters in the Proposed Algorithm

Parameter	Meaning	Value
N^{init}	size of the initial population	$18D \cdot C$
N^{min}	minimal sub-population size	4
H	size of the success history memory	6
r^{arc}	archive size $\lvert A\rvert$ = round($r^{arc} \cdot N^{subpop}$)	2.6
p	required in cur-to-$p$best/1 mutation	0.11
$[M_c^F]^0$, $[M_c^{CR}]^0$	initial values of c^{th} sub-population's F/CR memory	0.5
C	the number of sub-populations	$0.2 \cdot D$
$scale_{min}$	required in LSPI	0.1
$scale_{max}$	required in LSPI	0.2
p_m, η	parameters of polynomial mutation	1/D, 5 or 20

C. Algorithm Complexity

Algorithm complexity is estimated by calculating T_0, T_1 and T_2 values, as defined in the CEC2020 competition [1]. It aims to show the required computational effort of an algorithm with respect to different problem dimensions. T_0 denotes the time required to run the following computation for 1,000,000 times.

```
x=x+x; x=x/2; x=x*x; x=sqrt(x);
x=log(x); x=exp(x); x=x/(x+2)
```

T_1 denotes the time required to evaluate function f_1 for 200,000 times, and T_2 denotes the average computation time (over 5 runs) to run a whole evolutionary algorithm with *MAX_NFE* = 200,000. The competition rules ask for the measured values for D = 5, 10, and 15, shown in Table II.

The experiments were performed in a Windows 10 environment with Intel i9-9900KF (5.0GHz) CPU and 64GB DDR4-3466 RAM. The mpmL-SHADE was implemented by C++.

TABLE II. Computational Complexity

	T_0 (ms)	T_1 (ms)	T_2 (ms)	$(T_2 - T_1) / T_0$
D = 5		51.6105	219.91852	210.2536165
D = 10	0.8005	70.9715	262.65952	239.4603623
D = 15		101.8337	337.03866	293.8225609

D. Numerical Results

The competition defines the maximum number of fitness function evaluations (*MAX_NFE*) for the four problem dimensions, listed in Table III. Our algorithm solved each test function for 30 times. According to the competition rules, statistics including the best, the worst, median, mean, and standard deviation of the error values over 30 runs are reported in Tables IV to VII. The error value is the deviation of the obtained solution from the optimum. If the error value is less than 10^{-8}, it is regarded as zero.

TABLE III. The Maximum Number of Fitness Function Evaluations

Problem Dimension	*MAX_NFE*
D = 5	50,000
D = 10	1,000,000
D = 15	3,000,000
D = 20	10,000,000

TABLE IV. Error Values of mpmL-SHADE (D=5)

Func.	Best	Worst	Median	Mean	Std
f_1	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
f_2	0.0000E+00	3.7470E–01	6.2947E–02	1.1661E–01	1.3654E–01
f_3	0.0000E+00	5.4146E+00	5.1482E+00	4.7000E+00	1.4661E+00
f_4	2.3494E–02	1.4360E–01	1.0851E–01	9.6084E–02	3.0934E–02
f_5	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
f_8	0.0000E+00	1.0000E+02	0.0000E+00	3.3333E+00	1.7951E+01
f_9	0.0000E+00	3.0000E+02	1.0000E+02	1.0333E+02	4.0689E+01
f_{10}	3.0000E+02	3.4737E+02	3.4737E+02	3.4105E+02	1.6102E+01

TABLE V. ERROR VALUES OF MPML-SHADE (D=10)

Func.	Best	Worst	Median	Mean	Std
f_1	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
f_2	6.3009E–02	3.6023E+00	2.5191E–01	6.8095E–01	1.1259E+00
f_3	1.0367E+01	1.1365E+01	1.0461E+01	1.0565E+01	2.3384E–01
f_4	1.6905E–01	4.0592E–01	2.7884E–01	2.8404E–01	5.8284E–02
f_5	0.0000E+00	4.1629E–01	2.0814E–01	1.2489E–01	1.1526E–01
f_6	1.5651E–04	1.3554E–01	3.8008E–02	5.0894E–02	4.0899E–02
f_7	7.0608E–06	4.3363E–01	2.7867E–02	1.3732E–01	1.5459E–01
f_8	0.0000E+00	1.0000E+02	1.0000E+02	9.6667E+01	1.7951E+01
f_9	0.0000E+00	3.3186E+02	3.2635E+02	2.7606E+02	9.8182E+01
f_{10}	3.9774E+02	4.4578E+02	3.9801E+02	4.0260E+02	1.3855E+01

TABLE VI. ERROR VALUES OF MPML-SHADE (D=15)

Func.	Best	Worst	Median	Mean	Std
f_1	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
f_2	0.0000E+00	2.3601E+00	1.2491E–01	1.8306E–01	4.0725E–01
f_3	1.5567E+01	1.5567E+01	1.5567E+01	1.5567E+01	2.0963E–13
f_4	2.4114E–01	4.5490E–01	3.9389E–01	3.8242E–01	4.4318E–02
f_5	1.5612E–01	2.1460E+00	3.1224E–01	5.4042E–01	5.0447E–01
f_6	4.6535E–01	9.6453E–01	7.2995E–01	7.2517E–01	1.2459E–01
f_7	1.7299E–01	7.0814E–01	5.0000E–01	5.4320E–01	1.1498E–01
f_8	1.0000E+02	1.0000E+02	1.0000E+02	1.0000E+02	4.9118E–13
f_9	3.8708E+02	3.8968E+02	3.8968E+02	3.8959E+02	4.6649E–01
f_{10}	4.0000E+02	4.0000E+02	4.0000E+02	4.0000E+02	4.9815E–13

TABLE VII. ERROR VALUES OF MPML-SHADE (D=20)

Func.	Best	Worst	Median	Mean	Std
f_1	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
f_2	0.0000E+00	9.3685E–02	3.1228E–02	3.9673E–02	2.1195E–02
f_3	2.0387E+01	2.0387E+01	2.0387E+01	2.0387E+01	4.6736E–13
f_4	4.1611E–01	5.6583E–01	5.0069E–01	4.9748E–01	4.2334E–02
f_5	2.0817E–01	6.3861E+00	1.2552E+00	1.3801E+00	1.4521E+00
f_6	7.9020E–02	2.9642E–01	2.1061E–01	2.0522E–01	4.7101E–02
f_7	2.5517E–01	7.5000E–01	4.8157E–01	5.0994E–01	1.1874E–01
f_8	1.0000E+02	1.0000E+02	1.0000E+02	1.0000E+02	7.4723E–13
f_9	4.0031E+02	4.0278E+02	4.0133E+02	4.0139E+02	6.6822E–01
f_{10}	4.1366E+02	4.1366E+02	4.1366E+02	4.1366E+02	2.7413E–04

When the problem dimension is small (D = 5), our algorithm solves functions f_1 and f_5 to the optimality all the time. The optima of functions f_2, f_3, f_8 and f_9 can be found in the best case. When the dimension increases to 10 or higher, function f_1 still can be solved optimally over all 30 runs; functions f_5, f_8, and f_9 are solvable when D = 10, and f_2 is solvable when D = 15 and 20. Functions f_3, f_8, f_9, and f_{10} are difficult to our algorithm. It almost gets trapped in the same local optima all the time.

E. Performance Comparison

To know how well our mpmL-SHADE performs, we also ran L-SHADE and mL-SHADE to solve each of the ten test functions for 60 times with the computational budget in Table III. In Tables VIII to XI, we list mean and standard deviation of error values for the three algorithms. The standard deviation values are listed in the parentheses. We did t-test at the significance level of 0.05. The symbols +, −, and ≈ mean that the proposed mpmL-SHADE is better than (+), worse than (−) and not statistically different from (≈) the compared algorithm.

TABLE VIII. MEAN AND STD OF ERROR VALUES (D=5)

Func.	mpmL-SHADE	mL-SHADE		L-SHADE	
f_1	0 (0)	0 (0)	≈	0 (0)	≈
f_2	0.1124 (0.1222)	0.2349 (0.8698)	≈	0.3109 (0.8756)	≈
f_3	4.3700 (1.8132)	4.9887 (1.1087)	+	4.9091 (1.4054)	≈
f_4	0.0925 (0.0434)	0.0872 (0.0405)	≈	0.0976 (0.0380)	≈
f_5	0.0208 (0.1120)	0 (0)	≈	0.0104 (0.0799)	≈
f_8	3.49987 (17.9652)	0 (0)	≈	1.6667 (12.8019)	≈
f_9	100 (18.2574)	101.6744 (12.8610)	≈	101.6667 (12.8019)	≈
f_{10}	341.8408 (15.2059)	344.2091 (11.8154)	≈	341.8408 (15.2059)	≈

TABLE IX. MEAN AND STD OF ERROR VALUES (D=10)

Func.	mpmL-SHADE	mL-SHADE		L-SHADE	
f_1	0 (0)	0 (0)	≈	0 (0)	≈
f_2	0.8191 (1.2330)	1.9702 (2.4104)	+	5.5230 (3.9118)	+
f_3	10.6067 (0.2236)	10.7776 (0.5729)	+	11.8947 (0.5659)	+
f_4	0.2874 (0.0501)	0.2909 (0.0497)	≈	0.3000 (0.0567)	≈
f_5	0.1735 (0.1575)	0.3763 (0.3927)	+	0.2492 (0.1835)	+
f_6	0.0780 (0.0759)	0.1047 (0.1541)	≈	0.2395 (0.1640)	+
f_7	0.0940 (0.1401)	0.0465 (0.1207)	≈	0.2531 (0.2346)	+
f_8	100 (0)	98.7399 (9.6793)	≈	98.3391 (12.8027)	≈
f_9	218.5695 (111.2147)	284.9399 (87.9440)	+	296.4128 (82.5192)	+
f_{10}	401.7480 (12.5378)	412.5086 (21.3484)	+	413.2784 (21.6126)	+

TABLE X. MEAN AND STD OF ERROR VALUES (D=15)

Func.	mpmL-SHADE	mL-SHADE		L-SHADE	
f_1	0 (0)	0 (0)	≈	0 (0)	≈
f_2	0.1151 (0.0519)	0.3952 (0.7628)	+	8.6542 (16.1597)	+
f_3	15.5670 (0)	15.7586 (0.3642)	+	16.7030 (0.4934)	+
f_4	0.3823 (0.0501)	0.3965 (0.0438)	≈	0.3734 (0.0419)	≈
f_5	0.6163 (0.5200)	1.1141 (1.0080)	+	7.4813 (25.8315)	+
f_6	0.7188 (0.1363)	0.8408 (0.1842)	+	1.2232 (1.1271)	+
f_7	0.5321 (0.1551)	0.6679 (0.1887)	+	0.7104 (0.2051)	+
f_8	100 (0)	100 (0)	≈	100 (0)	≈
f_9	389.6343 (0.3327)	382.6355 (23.0764)	−	390.0513 (0.2892)	+
f_{10}	400 (0)	400 (0)	≈	400 (0)	≈

TABLE XI. MEAN AND STD OF ERROR VALUES (D=20)

Func.	mpmL-SHADE	mL-SHADE		L-SHADE	
f_1	0 (0)	0 (0)	≈	0 (0)	≈
f_2	0.0328 (0.0244)	0.0923 (0.3313)	≈	2.3906 (1.3826)	+
f_3	20.3872 (0)	20.4548 (0.1712)	+	20.8177 (0.5226)	+
f_4	0.4937 (0.0542)	0.4774 (0.0481)	≈	0.4696 (0.0466)	−
f_5	1.6733 (2.1976)	9.8369 (30.4214)	+	55.0543 (60.0608)	+
f_6	0.2043 (0.0434)	0.2221 (0.0564)	≈	0.3479 (0.0805)	+
f_7	0.5255 (0.1127)	0.3746 (0.1548)	−	0.8125 (0.1334)	+
f_8	100 (0)	100 (0)	≈	100 (0)	≈
f_9	401.5287 (0.6957)	400.1643 (1.7017)	−	402.7025 (1.0593)	+
f_{10}	413.6575 (0.0004)	413.6641 (0.0152)	+	413.6654 (0.0147)	+

The comparison results are summarized in Table XII. Overall speaking, mpmL-SHADE performs better than the two compared algorithms in solving 10-D and 15-D functions. The performance of mL-SHADE and mpmL-SHADE is close when solving 5-D and 20-D functions.

TABLE XII. SUMMARY OF COMPARISON RESULTS BETWEEN MPML-SHADE, ML-SHADE, AND L-SHADE

mpmL-SHADE vs.	mL-SHADE			L-SHADE		
	+	≈	−	+	≈	−
D = 5	1	7	0	0	8	0
D = 10	5	5	0	7	3	0
D = 15	5	4	1	6	4	0
D = 20	3	6	1	7	2	1
Total	14	22	2	20	17	1

F. Analysis of the Number of Sub-populations

In the last experiment, we tested six values, $0.2{\cdot}D$, $0.3{\cdot}D$, ..., $0.7{\cdot}D$, for the number of sub-populations (C) in mpmL-SHADE. The product is rounded up to get an integral value. We compared the six variants by the evaluation criteria in the CEC2020 competition. The measurement Score1 is based on the normalized errors, and Score2 is based on the ranks of compared variants. Each score contributes 50% of the total score. Table XIII shows the results. We found that the variant with $0.2{\cdot}D$ sub-populations performed the best. The performance declines as the number of sub-populations increases. After observing the convergence curves, we thought that a larger number of sub-populations results in fewer fitness function evaluations for each sub-population (since *MAX_NFE* is the same) and thus the algorithm stops before finding high-quality solutions.

TABLE XIII. PERFORMANCE COMPARISON WITH RESPECT TO THE NUMBER OF SUB-POPULATIONS

Number of Sub-populations	$0.2{\cdot}D$	$0.3{\cdot}D$	$0.4{\cdot}D$	$0.5{\cdot}D$	$0.6{\cdot}D$	$0.7{\cdot}D$
Score1	50	48.2995	44.1225	41.7733	45.9315	39.7564
Score2	49.0814	47.9487	50	47.1033	44.6301	43.7939
Total Score	99.0814	96.2482	94.1225	88.8766	90.5615	83.5503

V. CONCLUSIONS

In this paper we developed an adaptive DE for solving real-parameter single objective optimization problems. The proposed algorithm is based on our previous algorithm, and the new design is threefold: multi-population evolution, dynamic control of hyper-parameters, and mutation with dynamic intensity. Performance of the proposed algorithm was tested by ten test functions in a competition of CEC2020 and was compared with earlier versions. The results showed that the proposed design is beneficial for some functions, but certainly there is some space for improvement. We will continue our research to study how different algorithm components fit different functions and then propose a better integration. We will also investigate how to form sub-populations and how to do interaction.

REFERENCES

[1] C. T. Yue, K. V. Price, P. N. Suganthan, J. J. Liang, M. Z. Ali, B. Y. Qu, N. H. Awad, and Partha P Biswas , "Problem definitions and evaluation criteria for the CEC 2020 special session and competition on single objective bound constrained numerical optimization," Technical Report, Nanyang Technological University, Singapore, November 2019.

[2] R. Tanabe and A. S. Fukunaga, "Improving the search performance of SHADE using linear population size reduction," In: *Proceedings of IEEE Congress on Evolutionary Computation (CEC2014)*, Beijing, 2014, pp. 1658-1665. [L-SHADE]

[3] A. K. Qin and P. N. Suganthan, "Self-adaptive differential evolution algorithm for numerical optimization," In: *Proceedings of IEEE Congress on Evolutionary Computation (CEC2005)*, Edinburgh, Scotland, 2005, pp. 1785-1791. [SaDE]

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

[5] R. Tanabe and A. Fukunaga, "Success-history based parameter adaptation for differential evolution," In: *Proceedings of IEEE Congress on Evolutionary Computation (CEC2013)*, Cancun, 2013, pp. 71-78. [SHADE]

[6] J. Brest, M. S. Maučec and B. Bošković, "iL-SHADE: Improved L-SHADE algorithm for single objective real-parameter optimization," In: *Proceedings of IEEE Congress on Evolutionary Computation (CEC2016*), Vancouver, BC, 2016, pp. 1188-1195. [iL-SHADE]

[7] J. Brest, M. S. Maučec and B. Bošković, "Single objective real-parameter optimization: Algorithm jSO," In: *Proceedings of IEEE Congress on Evolutionary Computation (CEC2017)*, pp. 1311-1318, 2017. [jSO]

[8] N. H. Awad, M. Z. Ali and P. N. Suganthan, "Ensemble sinusoidal differential covariance matrix adaptation with Euclidean neighborhood for solving CEC2017 benchmark problems," In: *Proceedings of IEEE Congress on Evolutionary Computation (CEC2017)*, San Sebastian, 2017, pp. 372-379. [L-SHADE-cnEpSin]

[9] V. Stanovov, S. Akhmedova and E. Semenkin, "LSHADE algorithm with rank-based selective pressure strategy for solving CEC 2017 benchmark problems," In: *Proceedings of IEEE Congress on Evolutionary Computation (CEC2018)*, Rio de Janeiro, 2018, pp. 1-8. [L-SHADE-RSP]

[10] J. Yeh, T. Chen and T. Chiang, "Modified L-SHADE for single objective real-parameter optimization," In: *Proceedings of IEEE Congress on Evolutionary Computation (CEC2019)*, Wellington, New Zealand, 2019, pp. 381-386. [mL-SHADE]

[11] J. J. Liang and P. N. Suganthan, "Dynamic multi-swarm particle swarm optimizer," In: *Proceedings of IEEE Swarm Intelligence Symposium (SIS2005)*, 2005, pp. 124-129.

[12] P. Novoa-Hernandez, C. Cruz Corona, and D.A. Pelta, "Self-adaptive, multipopulation differential evolution in dynamic environments," *Soft Computing*, vol. 17, pp. 1861-1881, 2013.

[13] G. H. Wu, R. Mallipeddi, P. N. Suganthan, R. Wang, and H.K. Chen, "Differential evolution with multi-population based ensemble of mutation strategies," *Information Sciences*, vol. 329, pp. 329-345, 2016.

[14] D. Zaharie, "A multipopulation differential evolution algorithm for multimodal optimization," In: *Proceedings of 10th Mendel Int. Conference on Soft Computing*, Jun. 2004, pp. 17–22. [MMDE]

[15] W. Gao, G. G. Yen and S. Liu, "A cluster-based differential evolution with self-adaptive strategy for multimodal optimization," *IEEE Transactions on Cybernetics*, vol. 44, no. 8, pp. 1314-1327, Aug. 2014.

[16] B. Bošković and J. Brest, "Clustering and differential evolution for multimodal optimization," In: *Proceedings of IEEE Congress on Evolutionary Computation (CEC2017),* San Sebastian, 2017, pp. 698-705.

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

[18] P. Bujok and R. Polaková, "Migration model of jSO algorithm," In: *Proceedings of 25th International Conference on Systems, Signals and Image Processing (IWSSIP2018)* , 2018, pp. 1-5. [mSO]

[19] K. Deb and M. Goyal, "A combined genetic adaptive search (GeneAS) for engineering design," *Computer Science and Informatics*, 1996.

[20] M. Hamdan, "The distribution index in polynomial mutation for evolutionary multiobjective optimisation algorithms: An experimental study", In: *Proceedings of International Conference on Electronics Computer Technology*, 2012.