Combining Online and Offline Learning in MultiEA: An Improved Algorithm Portfolio Approach

RUAN Yuyan

***Abstract*—Evolutionary algorithm is a heated topic in the field of intelligence computing. This paper proposes a combination of online learning and offline learning on the basis of algorithm portfolio approach Multiple Evolutionary Algorithm (MultiEA) proposed by Yuen et al. to answer the question of how to select algorithms effectively, and proposes a new definition of the recent public future points. Without loss of generality, artificial bee colony (ABC), covariance matrix adaptive evolution strategy (CMA-ES), and a successful history-based adaptive differential evolution variable (LSHADE) were used for algorithm combination selection experiment.The experimental results show that by incorporating the online and offline learning, the performance of MuliEA is significantly improved.**

***Index Terms*—Evolutionary algorithms, Algorithm portfolio, Online and offline learning.**

I. Introduction

Many Evolutionary Algorithms (EAs) method, inspired by the natural phenomena, have shown their powerful abilities on solving numerical optimization problems. Commonly used algorithms including Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [1], Artificial Bee Colony (ABC) [2] and Differential Evolution(DE) are effective on both artificial and practical numerical black box optimization problems. However, solving numerical black box optimization problem is sometime still very challenging for individual algorithms with limited computational cost. Some individual algorithms are lack of computational resources, taking too much time to converge. While some are easily trapped in the local optimization, taking too less time to converge.

As No Free Lunch(NFL) theorems [3] reveals that the single individual algorithms are hardly to dominate in every problem because each algorithm has its strengths and weakness. Accordingly, constituting algorithms in a portfolio to solve numerical optimization problems is a good approach to utilize the abilities of the algorithms. However, algorithms in the portfolio share the computational cost to solve the problem. Compared with single algorithms, computational resources for each member algorithm in the portfolio is decreased. Therefore, a good portfolio method should dynamically allocate computational budget according to different problems. For example, more computational resources should be allocated to promising algorithms for efficient exploration of the optimization while the badly-performed algorithms are likely to stop searching in the early stage.

In this paper, we extended the previous algorithm portfolio approach Multiple Evolutionary Algorithm (MultiEA) [4] with the improvement of algorithm selection process in each generation and tested with a portfolio with three algorithms. In [4], choosing the algorithm for upcoming generation is based the one randomly sampled predicted result from bootstrap probability of algorithms. Because using a predicted value to represent the performance of an algorithm is too random, it may bring errors in algorithm selection. In this paper, we redefine and calculate online and offline probability to select algorithms in each generation. In stead of selecting algorithms only based on the sampled predicted results from algorithms, we flexibly apply online and offline probability to guide the algorithm for following generations. All tested methods keep the easy-access characters of MultiEA and apply new method to accurately calculate the probability. Some of the tested groups improve the predictive performance compared with original MultiEA.

The rest of this paper is organized as follows: Section 3 reviews the previous works of MultiEA and other portfolio methods. Section 4 lists a group of modified MultiEA method. Section 5 states the experimental results and section 6 is the analysis of the results.

II. Methodology

This set of experiments focuses on the selection of algorithms among an algorithm portfolio(AP). Without loss of generality, AP contains 3 algorithms: Artificial Bee Colony (ABC), Covariance Matrix Adaptation Evolution Strategy (CMA-ES), successful history-based adaptive differential evolution variants with linear population size reduction (L-SHADE) [5]. Some modifications are made based on the original version of MultiEA.

A. New definition of the nearest common future point(nfp)

Suppose there is an algorithm portfolio AP composed of q algorithms. Let α_i be the number of generations that Algorithm i A_i has run and the nearest common future point $nfp = max\{m_1t_1, ..., m_qt_q\}$, where $t_i = \alpha_i + 1$ and m_i is the average number of evaluations ($nfes$) used by A_i in the original version. Now, an extra parameter β is introduced to compute the nfp. The nfp is the minimum integer *such that* $nfp \geq max\{nfes_1, ..., nfes_q\}$ and $nfp \ mod \ \beta = 0$, where $nfes_i$ is the nfes used by A_i so far.

B. Online Learning

Instead of getting a single sample from the bootstrap distribution of the predicted fitness value, the online learning draws α samples to calculate the online probability. After getting these samples for every algorithm, we proposed 2 methods to calculate the online probability vector from them,

and they are *straight-forward sampling* method and *sampling-and-combinations* method.

Computation procedure

Supposed $bpd_i(nft)$ is the bootstrap distributions for the predicted fitness of A_i at nfp respectively.

1) Straight-forward sampling
   The procedure of straight-forward sampling is demonstrated in *Algorithm 1*.

Algorithm 1 Calculate the Online Probability vector by straight-forward sampling method

Require: A portfolio of q EAs $AP = \{A_1, ..., A_q\}$;number of samples drawn from the bootstrap distribution β; nearest common future point nfp at the moment; bootstrap probability distribution $bpd_i(nft)$ for A_i at nft.

Ensure: Online probability calculated from straight-forward-sampling method $P_s(A_i, nft)$

for $i = 1 \rightarrow q$ **do**
  $Times_{win}(i) \leftarrow 0$
end for
for $j = 1 \rightarrow \alpha$ **do**
  for $i = 1 \rightarrow q$ **do**
    Get the sample fitness $sf(i, j)$ for A_i at iteration j based on distribution $bpd_i(nft)$
  end for
  Construct result combinations $S(j) = \{sf(1, j), sf(2, j), ..., sf(1, j)\}$ where $sf(i, j)$ is minimal among $S(j)$
  Update wining times for algorithm A_i $Times_{win}(i) \leftarrow Times_{win}(i) + 1$
end for
Online probability for algorithm A_i at $t = nft$: $P_s(A_i, nft) = Times_{win}(i)/\alpha$

2) Sampling-and-combinations
   The sampling procedure in sampling-and-combinations method is the same as the one in straight-forward sampling method. However, instead of accumulating winning times right after getting one sample, a permutation is obtained first and treated the same as the drawn samples. Concretely, suppose n samples are drawn for the q algorithms, then n^q *scenarios* will be obtained since there are q positions in each scenarios and for each position, there are n choices.

An illustrative example

Here is a concrete example to demonstrate how the online probability vector is obtained using different methods.

1) Straight-forward sampling
   Suppose $\alpha = 5$ and the samples drawn from the bootstrap distributions are as shown in Table. I
   Before sampling, i.e. iteration 0 in the table, $Times_{win}$ are initialized to 0. $Times_{win}(i)$ is incremented by 1 if the sample drawn at that iteration for A_i is the minimal among all the samples since it is a minimization problem. For example, $sf(1, 1)$, the sample fitness for A_1 at iteration 1, is the cell located at row iteration 1 and column $A1$, which is colored in red; $sf(2, 1)$ is the cell located at row iteration 1 and column $A2$, which is colored in blue. The online probability in this scenario is $(1/6, 3/6, 2/6)$, i.e. $(0.17, 0.5, 0.33)$.

TABLE I
ILLUSTRATION OF STRAIGHT-FORWARD SAMPLING

iteration	A_1	A_2	A_3	$Times_{win}(1)$	$Times_{win}(2)$	$Times_{win}(3)$
0	NA	NA	NA	0	0	0
1	3	4	2	0	0	1
2	5	2	4	0	1	1
3	4	1	2	0	2	1
4	1	5	6	1	2	1
5	4	2	3	1	3	1

TABLE II
ILLUSTRATION OF SAMPLING-AND-COMBINATIONS

iteration	A_1	A_2	A_3	$Times_{win}(1)$	$Times_{win}(2)$	$Times_{win}(3)$
0	NA	NA	NA	0	0	0
1	3	4	1	0	0	1
2	3	4	6	1	0	1
3	3	2	1	1	0	2
4	3	2	6	1	1	2
5	5	4	1	1	1	3
6	5	4	6	1	2	3
7	5	2	1	1	2	4
8	5	2	6	1	3	4

2) Sampling-and-combinations
   Suppose q is 3, n is 2, and the fitness samples that we get for A_1 is {3, 5}, A_2 is {4, 2} and A_3 is {2, 4}. In the resultant scenarios, there will be 2 possible number appear at 1^{st} position, i.e. 3 and 5. Similarly, for the 2^{nd} position can be 4 and 2, and for the 3^{rd} position, it can be 2 and 4. Then 8 combinations will be obtained and treated as if they were the directly drawn samples in the step of computing the $Times_{win}$. The online probability in this scenario is $(1/8, 3/8, 4/8)$, i.e. $(0.125, 0.375, 0.5)$.

C. Offline learning

Apart from learning from the information about the current problem, i.e. the online learning, some historical knowledge of the performance of some known algorithms on some specific problem type is also valuable sometimes. For example, for convex problem, it is well-known that L1-Magic Algorithm is the best algorithm. Therefore, offline probability vector $offline$ for a given algorithm portfolio for a specific problem is introduced. $offline[i]$ is defined as the probability that A_i outperforms others for the given problem. In real-life, $offline[i]$ may be some value between 0 and 1 because it is hard to find some algorithm that is absolutely the best. However, in order to show the effect of this offline probability vector, without loss of generality, in our experiment, some extra experiments are done to find out the best algorithm for every benchmark function in the CEC13 testbed, and $offline[i]$ is set to one if A_i outperforms others based on the pre-experiment results. For example, if A_1 is the known

best algorithm for solving the given problem, for example, function 10 in CEC13 testbed, $offline[1]$ is set to 1 while for all other i. Offline probability for the given algorithm portfolio will *differ* on different problems.

D. Combining online and offline learning

After getting the online probability vector and offline probability vector, we proposed some ways to incorporate both online knowledge and offline knowledge into the MultiEA and they are *Multiply Method*, *Average Method* and *Validation Method*.

To better illustrate the computation procedure, the 3 methods are explained based on one example. Suppose there are 3 algorithm in the portfolio, and the offline probability vector is $(0, 1, 0)$ and online probability vector is $(0.02, 0.05, 0.93)$.

1) Multiply Method
   The final probability vector P_{final} is computed by element-wise multiplying the online probability vector and offline probability vector and normalize it afterward. In the sample given above, $P_{final} = (0, 0.05, 0)$ and after normalization, $P_{final} = (0, 1, 0)$. Note that when *one* appears in both online probability vector and offline probability vector and they select different algorithm, the algorithm that *offline* probability vector selects should be chosen. For example, suppose $offline = (1, 0, 0)$, suggesting that A_1 is the best algorithm, while $online = (0, 1, 0)$, suggesting that A_2 is the best algorithm, P_{final} will be set to $offline$ directly. Obviously, P_{final} is the same as $P_{offline}$, which is simply use one single algorithm to solve the problem instead an algorithm portfolio.
2) Average Method
   The final probability vector P_{final} is computed by element-wise averaging the online probability vector and offline probability vector. In the sample given above, $P_{final} = (0.01, 0.5, 0.465)$ and after normalization, $P_{final} = (0.01, 0.518, 0.481)$.
3) Validation Method
   This method is to determine which probability vector that should be chosen given two probability vector first, i.e. online probability vector or offline probability vector, by making a quick stimulation, and trust it entirely. Concretely, two fitness value sequences are obtained by only using the online probability vector and offline probability vector, and their performance is evaluated at a common future point based on these two sequences. The decision of using which probability vector is made based on this predicted performance. The detailed computation steps are shown in *Algorithm 2*

III. Experiments

In this research, we test the original MultiEA and 5 variants of it on CEC13 testbed with dimension 30 and budget 10000*D, i.e. 300000. In the variants, the nearest common future point nfp is computed in the newly proposed way with β set to 150. If online learning is incorporated, α is set to 100.

Algorithm 2 Calculate the final probability vector by validation method

Require: Online probability vector $online$; offline probability vector $offline$; the fitness value sequence that A_i has obtained so far $fitvals(i)$; the average number of evaluations that A_i cost for one generation $nfes(i)$; β the extra parameter to compute the nearest common future point.

Ensure: Final probability vector calculated by validation method $P_s(A_i, nft)$

$maxlen \leftarrow max\{len(fitvals(1)), ..., len(fitvals(q))\}$
for $i = 1 \rightarrow q$ **do**
  if $len(fitvals(i)) < maxlen$ **then**
    Use the last value in $fitvals(i)$ to fill the vector to $maxlen$ elements
  end if
end for
for $i = 1 \rightarrow q$ **do**
  Initialize the index of which value among $fitvals$ to be selected next time $index(i) \leftarrow 1$
end for
The sequence of fitness values obtained by the online probability vector $y_{on} \leftarrow \emptyset$, the number of evaluation cost $x_{on} \leftarrow \emptyset$
The sequence of fitness values obtained by the offline probability vector $y_{off} \leftarrow \emptyset$, the number of evaluation cost $x_{off} \leftarrow \emptyset$
// Get the fitness value sequence using only $online$
while $max\{index\} \leq maxlen$ **do**
  Roulette select algorithm A_i based on $online$
  $y_{on} \leftarrow y_{on} \cup fitvals(i, index(i))$
  $x_{on} \leftarrow x_{on} \cup nfes(i, index(i)) + x_{on}(end)$
  $index(i) \leftarrow index(i) + 1$
end while
// Get the fitness value sequence using only $offline$
for $i = 1 \rightarrow q$ **do**
  $index(i) \leftarrow 1$ ▷ Reset $index$
end for
while $max\{index\} \leq maxlen$ **do**
  Roulette select algorithm A_i based on $offline$
  $y_{off} \leftarrow y_{off} \cup fitvals(i, index(i))$
  $x_{off} \leftarrow x_{off} \cup nfes(i, index(i)) + x_{off}(end)$
  $index(i) \leftarrow index(i) + 1$
end while
The nearest common future point nfp_{valid} is computed based on $x_{on}(end)$ and $x_{off}(end)$ with the β in the main algorithm
Obtain the predicted fitness value at nfp_{valid} for both sequences pf_{on} and pf_{off} using linear regression
if $pf_{on} < pf_{off}$ **then**
  $P_{final} \leftarrow online$
else
  $P_{final} \leftarrow offline$
end if

A. Group Definitions

1) Group 1: Using the original implementation proposed by YUEN, et.al [4].
2) Group 2: Incorporating the online learning using the sampling-and-combinations, which means selecting the algorithm for the next generation based on the online probability using the sampling-and-combinations method.
3) Group 3: Incorporating the online learning using the straight-forward sampling, which means selecting the algorithm based on the online probability using the straight-forward-sampling method.
4) Group 4: Incorporating both online and offline learning using the multiply method, which means selecting the algorithm for the next generation based on the final probability computed from both online and offline probability using the multiply method.
5) Group 5: Incorporating both online and offline learning using the multiply method, which means selecting the algorithm for the next generation based on the final probability computed from both online and offline probability using the average method.
6) Group 6: Incorporating both online and offline learning using the validation method, which means selecting the algorithm for the next generation based on the final probability computed from both online and offline probability using the validation method.

B. Results

1) Means on CEC13 testbed, 30D
   The means of the 6 groups are shown in the Table III.
2) Comparisons with the original version
   U test is performed between the 5 variants and the original version, and the results are extracted and displayed Table IV.
3) Comparisons of the 2 strategies for computing the online probability
   Since two different strategies are deployed to compute the online probability, a separate U-test is performed between these two groups, i.e. *Group 2* and *Group 3*. The result is shown in Table V.

TABLE III
MEANS OF THE 6 GROUPS

	Group1	Group2	Group3	Group4	Group5	Group6
f_1	0.00E+00	3.78E-08	0.00E+00	0.00E+00	0.00E+00	0.00E+00
f_2	1.53E+02	8.72E+03	2.29E+02	4.86E+03	3.16E+03	7.03E+03
f_3	6.27E+03	9.56E+05	1.83E+02	4.97E+02	4.64E+02	2.03E+05
f_4	3.15E+03	3.76E+03	1.08E-06	1.84E-08	8.13E-07	4.54E-07
f_5	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
f_6	1.35E+00	8.80E-01	3.25E-04	8.80E-01	7.34E-01	8.80E-01
f_7	6.70E+00	1.59E+00	7.03E+00	1.88E+00	3.48E+00	4.48E+00
f_8	2.09E+01	2.09E+01	2.09E+01	2.09E+01	2.08E+01	2.09E+01
f_9	3.14E+01	3.34E+01	2.93E+01	2.83E+01	2.66E+01	2.82E+01
f_{10}	1.51E-02	1.35E-02	1.31E-02	1.92E-02	1.05E-02	2.48E-02
f_{11}	4.33E+01	3.50E+00	4.86E+01	0.00E+00	0.00E+00	0.00E+00
f_{12}	4.80E+01	2.40E+01	4.84E+01	1.59E+01	1.45E+01	1.64E+01
f_{13}	1.01E+02	4.62E+01	9.11E+01	2.75E+01	2.88E+01	2.98E+01
f_{14}	1.42E+03	4.84E+03	1.02E+03	9.02E-03	1.62E-02	3.83E+02
f_{15}	3.84E+03	5.35E+03	5.07E+03	3.13E+03	3.07E+03	3.32E+03
f_{16}	2.02E+00	9.21E-02	1.86E+00	8.27E-02	8.86E-01	1.65E+00
f_{17}	4.17E+01	3.14E+01	3.55E+01	3.04E+01	3.04E+01	3.04E+01
f_{18}	7.80E+01	7.13E+01	7.65E+01	6.68E+01	6.51E+01	6.49E+01
f_{19}	2.61E+00	1.64E+00	1.94E+00	1.24E+00	1.19E+00	1.14E+00
f_{20}	1.11E+01	1.21E+01	1.12E+01	1.04E+01	1.05E+01	1.06E+01
f_{21}	2.60E+02	2.86E+02	2.60E+02	2.82E+02	2.40E+02	2.87E+02
f_{22}	9.05E+02	2.49E+03	7.99E+02	8.72E+01	8.73E+01	8.47E+01
f_{23}	5.20E+03	6.56E+03	4.54E+03	3.34E+03	3.39E+03	3.41E+03
f_{24}	2.19E+02	2.06E+02	2.16E+02	2.07E+02	2.11E+02	2.10E+02
f_{25}	2.72E+02	2.89E+02	2.68E+02	2.52E+02	2.47E+02	2.68E+02
f_{26}	2.13E+02	2.13E+02	2.22E+02	2.17E+02	2.18E+02	2.04E+02
f_{27}	6.65E+02	5.33E+02	6.48E+02	3.49E+02	4.47E+02	6.11E+02
f_{28}	3.00E+02	2.93E+02	2.93E+02	3.00E+02	2.94E+02	3.00E+02

TABLE IV
U TEST RESULTS OF THE 5 VARIANTS WITH THE ORIGINAL DESIGN

	Group 2	Group 3	Group 4	Group 5	Group 6
f_1	=	=	=	=	=
f_2	-	=	-	-	-
f_3	=	=	=	=	=
f_4	+	=	+	=	=
f_5	=	=	=	=	=
f_6	=	=	=	=	=
f_7	+	=	+	=	=
f_8	-	=	=	=	=
f_9	=	=	+	+	=
f_{10}	=	=	=	+	-
f_{11}	+	=	+	+	+
f_{12}	+	=	+	+	+
f_{13}	+	=	+	+	+
f_{14}	-	=	+	+	+
f_{15}	-	-	=	=	=
f_{16}	+	=	+	+	=
f_{17}	-	=	-	+	+
f_{18}	=	=	+	+	+
f_{19}	=	+	+	+	+
f_{20}	=	=	+	+	=
f_{21}	=	=	-	=	-
f_{22}	-	=	+	+	+
f_{23}	-	=	+	+	+
f_{24}	+	=	+	=	+
f_{25}	-	=	+	+	=
f_{26}	=	=	=	=	=
f_{27}	+	=	+	+	=
f_{28}	-	=	=	=	=

IV. RESULTS AND DISCUSSIONS

A. Different strategies to compute online probability

Since there are 3 types of problems in the CEC13 testbed, the number of the two strategies computing the final probability vector outperform the other on what kind of problem is counted and presented in Fig. 1 and Fig. 2. Concretely, the *straight-forward sampling* method outperforms the *sampling-and-combinations* on 1 unimodal function, 6 basic multimodal functions and 2 composition functions while the *sampling-and-combinations* method outperforms the *straight-forward sampling* method on 1 unimodal function, 3 multimodal functions and 5 composition functions.

It can be observed that both strategies have their strengths and drawbacks, where the *sampling-and-combinations* method

TABLE V
U-TEST OF GROUP 2 AND 3

	h	p	zval	ranksum	Result
f_1	0	0.333710696	0.966666667	930	=
f_{21}	1	2.60E-11	6.667888748	1343	+
f_3	0	0.47332197	0.717084483	964	=
f_4	1	0.002170798	-3.06581192	711	-
f_5	0	NaN	NaN	915	=
f_6	0	1	0	915.5	=
f_7	1	0.000140669	-3.806989545	657	-
f_8	1	0.045146208	2.003289644	1051	+
f_9	0	0.176127545	1.352774926	1007	=
f_{10}	0	0.782915912	0.275521107	934	=
f_{11}	1	1.53E-08	-5.658586834	554	-
f_{12}	1	2.28E-05	-4.235737883	628	-
f_{13}	1	6.77E-05	-3.98440265	645	-
f_{14}	1	2.29E-09	5.97584704	1317	+
f_{15}	0	0.129670225	1.515403605	1018	=
f_{16}	1	2.37E-10	-6.335126292	486	-
f_{17}	1	1.05E-06	4.881583629	1245	+
f_{18}	1	0.043583548	-2.01807407	778	-
f_{19}	0	0.118817344	1.559756882	1021	=
f_{20}	0	0.290472149	1.057086417	987	=
f_{21}	1	0.035617164	2.101271226	1045	+
f_{22}	1	0.02920541	2.180702749	1063	+
f_{23}	1	0.00039881	3.540869888	1155	+
f_{24}	1	0.012732115	-2.491175683	746	-
f_{25}	1	9.21E-05	3.910480523	1180	+
f_{26}	0	0.853381737	0.184805318	928	=
f_{27}	1	0.008684371	-2.624235512	737	-
f_{28}	1	0.015660356	2.41673633	1017	+

seems to have better performance on composition functions while *straight-forward sampling* method seems to have better performance on multimodal functions.

The reason behind may be that in the *sampling-and-combinations* method, the permutation can produce some scenario that may not be likely to happen, which can increase the variability of the algorithm, thereby having better performance on the more complicated problems, i.e. composition functions.

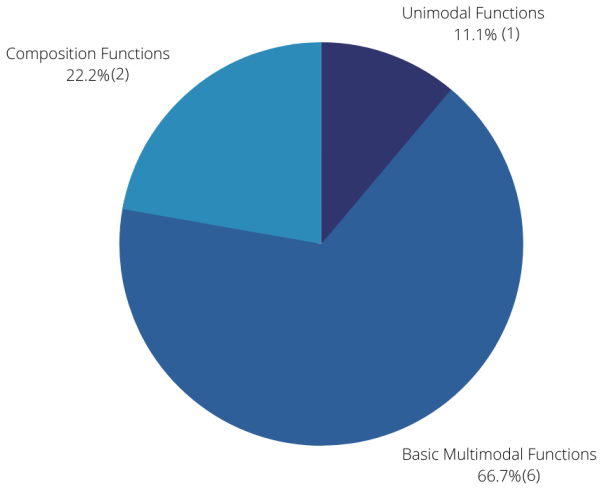

Fig. 1. Portion of function types that group 2 dominates

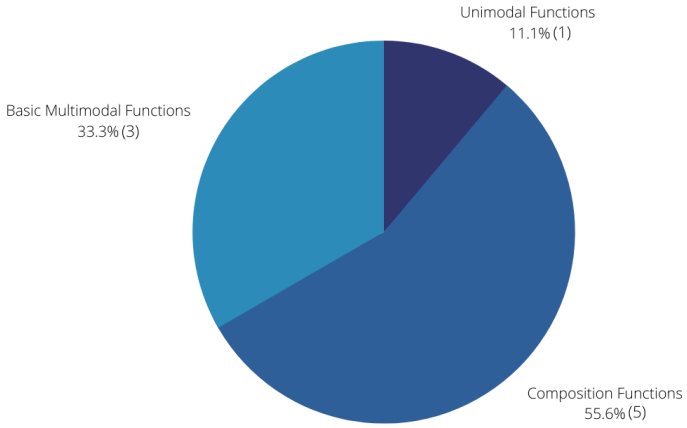

Fig. 2. Portion of function types that group 3 dominates

B. Performance of different groups on different types of problems

From Table VI, it can be observed that the performance for Group 4, 5, 6 is pretty good, and especially, for Group 5, it does not perform worse than the original version on any function other than one Unimodal function, which is a good sign for robustness. For another thing, the improvements mainly lie in the Multimodal functions with relatively slight impact on Unimodal functions and Composition functions. This may indicate that the MultiEA with online and offline probability may have better performance on problems with medium difficulty. Moveover, Group 4, 5 and 6 on Composition functions are improved as well, which means the ability to solve difficult problems is enhanced.

TABLE VI
PERFORMANCE OF DIFFERENT GROUPS ON DIFFERENT TYPES OF PROBLEMS

Better			
	Unimodal(5)	Multimodal(15)	Composition(8)
Group 2	1	5	2
Group 3	0	1	0
Group 4	1	10	5
Group 5	0	11	4
Group 6	0	7	3
Worse			
	Unimodal(5)	Multimodal(15)	Composition(8)
Group 2	1	4	4
Group 3	0	1	0
Group 4	1	1	1
Group 5	1	0	0
Group 6	1	1	1

V. CONCLUSION

In this paper, we introduce the concept of online and offline learning in Evolutionary Computation and several strategies to incorporate both into MultiEA. Compared to the original version of MultiEA, the performance is greatly improved when

combing online and offline learning, and the improvement mainly lies in dealing with multimodal and composition problems, which indicates that the ability to solve problems of medium and above difficulty is enhanced. In addition, performance of average and multiply method is better than validation method in general, which is a sign for incorporating both online and offline knowledge is better than using one only.

REFERENCES

[1] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es)," *Evolutionary computation*, vol. 11, no. 1, pp. 1–18, 2003.

[2] D. Karaboga and B. Akay, "A comparative study of artificial bee colony algorithm," *Applied mathematics and computation*, vol. 214, no. 1, pp. 108–132, 2009.

[3] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE transactions on evolutionary computation*, vol. 1, no. 1, pp. 67–82, 1997.

[4] S. Y. Yuen, C. K. Chow, X. Zhang, and Y. Lou, "Which algorithm should i choose: An evolutionary algorithm portfolio approach," *Applied Soft Computing*, vol. 40, pp. 654–673, 2016.

[5] R. Tanabe and A. S. Fukunaga, "Improving the search performance of shade using linear population size reduction," in *2014 IEEE congress on evolutionary computation (CEC)*. IEEE, 2014, pp. 1658–1665.