Supplementary Material of "Contribution-based Cooperative Co-evolution for Non-separable Large-scale Problems with Overlapping Subcomponents"

Ya-Hui Jia, *Member, IEEE,* Yi Mei, *Senior Member, IEEE,* and Mengjie Zhang, *Fellow, IEEE*

I. Cooperation Frequency Analysis

In this section, we give the analysis of the cooperation frequency of each optimization model. First, a measurement of cooperation frequency is designed based on the round-robin (RR) structure. Then, the experimental results are given to show the difference among the CC optimization models.

Since conceptually RR has the largest cooperation frequency, we use RR as the reference to calculate other models' cooperation frequencies. Given M optimizers corresponding to M subcomponents of a n-dimensional problem, if an optimizer needs to use N times of fitness evaluations (FEs) in one generation, RR will totally use $M \cdot N$ FEs in one generation to optimize all n variables. Defining each $M \cdot N$ FEs as a period, for a specific CC model, we will check how many variables are optimized in every period. Suppose within a period, n^* variables are optimized, then, in this period, the cooperation frequency of this model is calculated as n^*/n. Since the problem is decomposed into subcomponents in CC methods, more variables being optimized in a period means more optimizers having cooperated in this period.

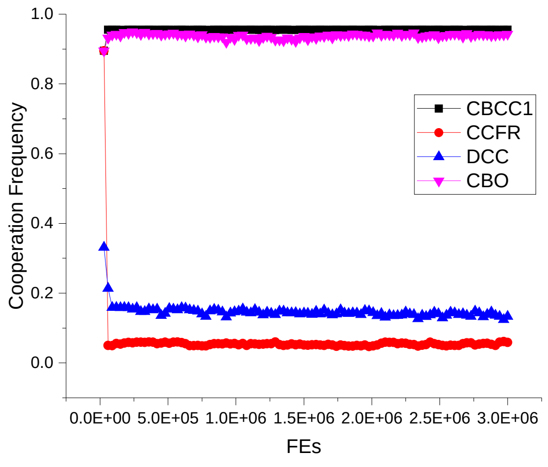
Since most evolutionary algorithms are iterative algorithms, the period would shift along with the optimization process which means that the cooperation frequency defined here is a dynamic value instead of a static value. Assuming that each subcomponent has the same number of variables, i.e. n/M, the lower bound and the upper bound of each CC model's cooperation frequency can be easily calculated as shown in Table A, including RR, CBCC1, CCFR, DCC, and CBO.

TABLE A
Lower bound and upper bound of cooperation frequency

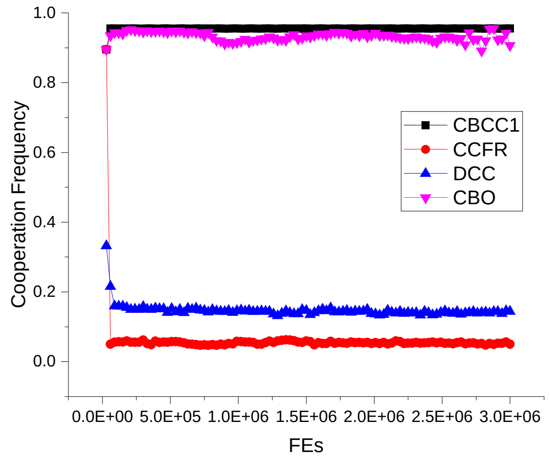
Model	Lower	Upper
RR	1	1
CBCC1	$(M-1)/M$	1
CCFR	$1/M$	$2/M$
DCC	$1/M$	1
CBO	$1/2$	1

Taking two functions with uniform subcomponent sizes f_3 and f_4 as representatives, the cooperation frequency is calculated for each CC model that is shown in Fig. A.

Basically, the results shown in these two figures are in accordance with our analysis. The cooperation frequency of CBO is slightly lower than CBCC1 since it rewards more



(a)



(b)

Fig. A. Cooperation frequency of CBCC1, CCFR, DCC, and CBO. (a) f_3, (b) f_4.

subcomponents than CBCC1. The cooperation frequencies of CBO and CBCC1 are both much higher than DCC and CCFR. CCFR has the lowest cooperation frequency that is the main reason why its performance is not good.

Considering both the experiment here and the experiments in the main paper, one of the basic assertions of this paper that both high cooperation frequency and efficient resource allocation are important to optimizing overlapping problems is verified. RR and CBCC1 has higher cooperation frequency, however, the resource allocation methods in these two models are not efficient. CCFR has a good measurement of contribution of subcomponents to allocate computing resources, however, it does not have high cooperation frequency. Due to the contribution accumulation, DCC does not have high cooperation frequency either. CBO has found a good balance between these two factors, thus, it is more effective than other models on the large-scale overlapping problems.

From Fig. A, we can see the curve of the cooperation frequency of CBO has some fluctuations in the final stage of the optimization. The reason can be explained from Fig. D(d). From Fig. D(d), we can know that CBO has already stopped optimizing the problem from 10^6 FEs. Then, the contribution of each subcomponent basically halves in every generation. In the final stage, the contributions of many subcomponents become zero or near zero. (It may become zero due to the precision of the double floating number.) Then, in this stage, the effect of historical information of the contribution can be ignored. If an optimizer can make a slightly difference, it will be awarded. Since the contribution made in this stage is basically due to the randomness of the algorithm, the awarding process is very unstable. That is the reason why there are some fluctuations.

II. Full Figure Sets

Mapping between the figures in the supplementary material and the figures in the paper is shown as follows:

1) Fig. B is the full set of Fig. 6 (Convergence curves of CSO, LLSO, SHADEILS, RDG3, DCC, FEA, CCAS and CBCCO) of the paper. Basically, Fig. B verifies the analyses we made in Section V.A of the paper (page 10). When CMA-ES can effectively handle the subcomponents, i.e. on $f_1 - f_8$, CBCCO has the fastest convergence speed. On conforming functions, it will maintain this speed from the beginning to the end. On conflicting functions, CBCCO and RDG3 usually can get similar final results but the convergence speed of CBCCO is still faster. When CMA-ES cannot handle the subcomponents, i.e. on $f_9 - f_{12}$, all algorithms converge very early including CBCCO.
2) Fig. C is the full set of Fig. 7 (Convergence curves of CBD-R, Greedy, and CBD) of the paper. The observations can be gotten from Fig. C are similar to the analyses that we have made in Section V.B (page 11). On all of these figures, we can find that CBD is better than or at least equivalent to the greedy decomposition. Both the measurements of convergence speed and final objective value show that CBD is more effective than the other two decomposition methods. Even on $f_9 - f_{12}$ where all algorithms are premature, CBD is still better.
3) Fig. D corresponds to the experiment in Section V.C (Analysis of CBO) of the paper. Based on Fig. D, we can find that CBO has the fastest convergence speed compared with RR, CBCC1, and CCFR. The convergence speed of CCFR is the slowest since it has abandoned the RR structure which leads to a low cooperation frequency. The fast convergence speed of CBO is due to its effective resource allocation and the high cooperation frequency.

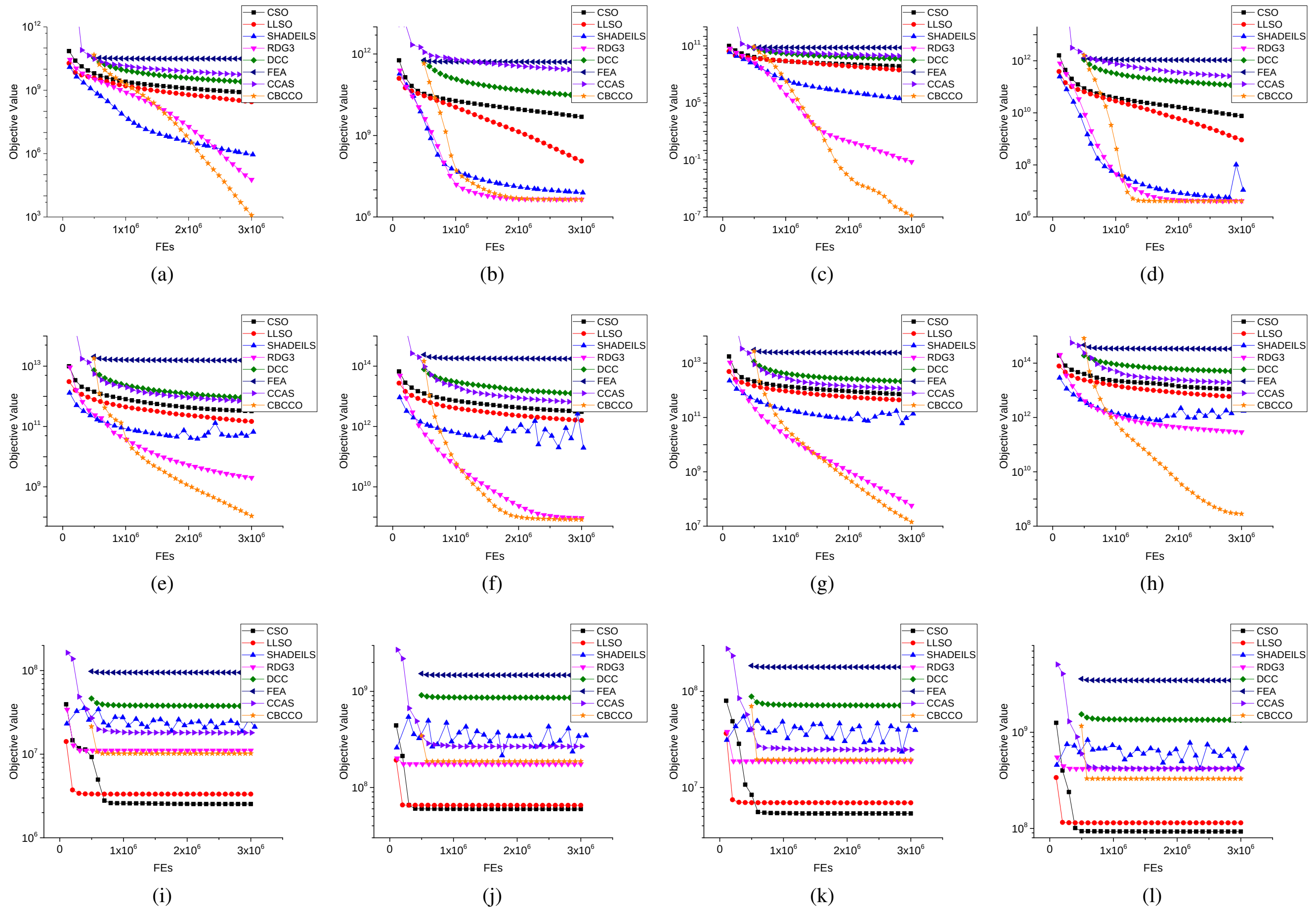
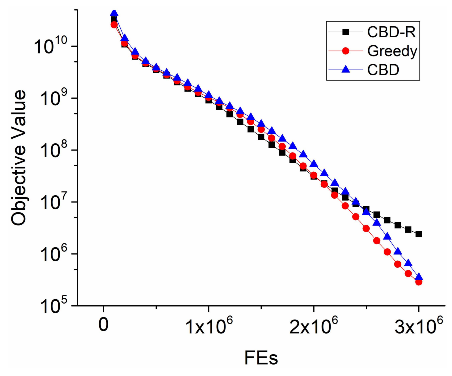
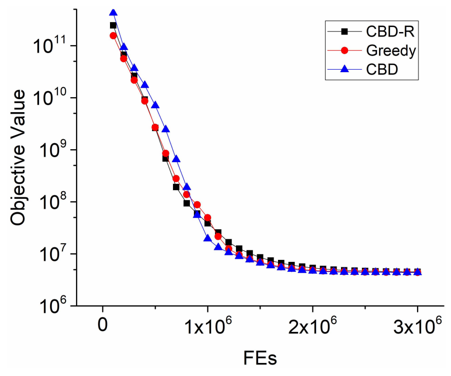


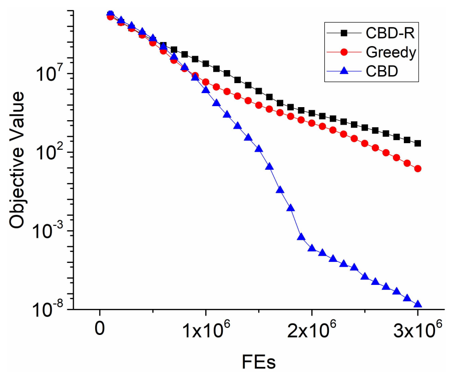
Fig. B. Convergence curves of CSO, LLSO, SHADEILS, RDG3, DCC, FEA, CCAS and CBCCO. (a) f_1, (b) f_2, (c) f_3, (d) f_4, (e) f_5, (f) f_6, (g) f_7, (h) f_8, (i) f_9, (j) f_{10}, (k) f_{11}, (l) f_{12}.



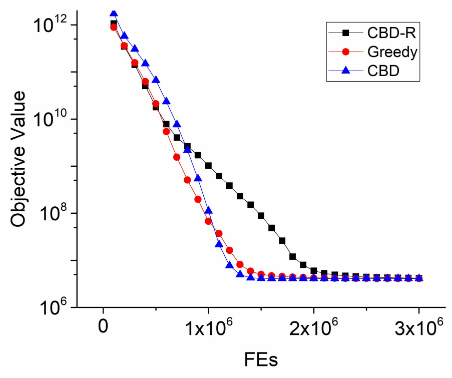
(a)



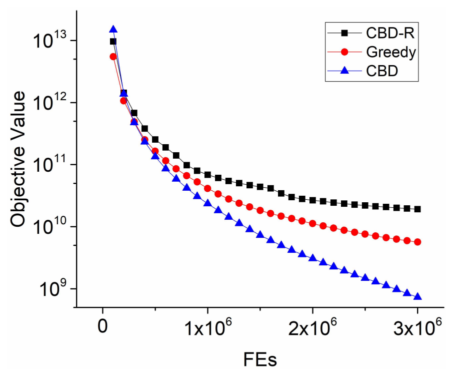
(b)



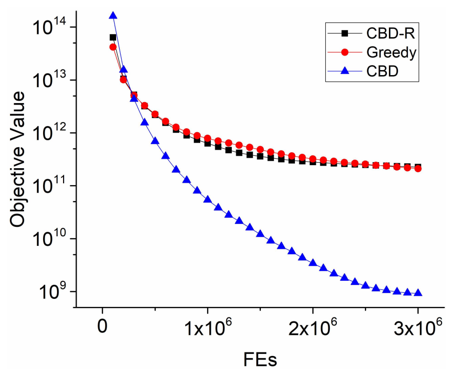
(c)



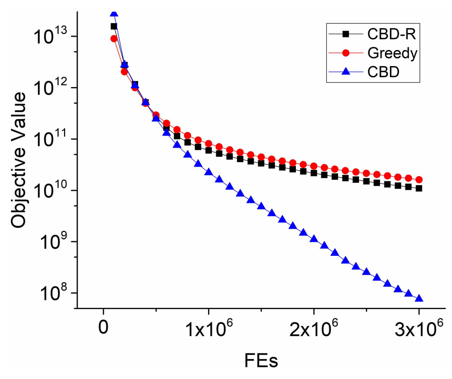
(d)



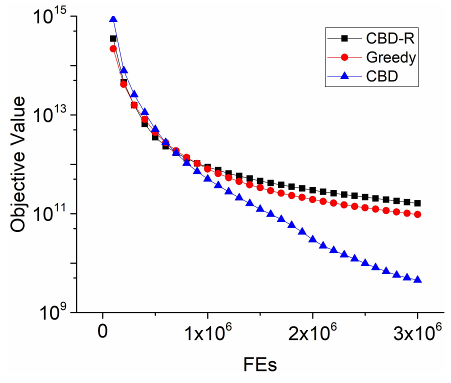
(e)



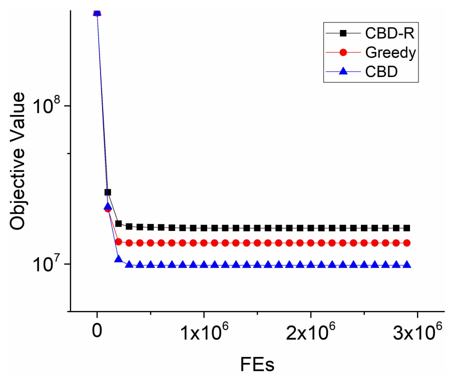
(f)



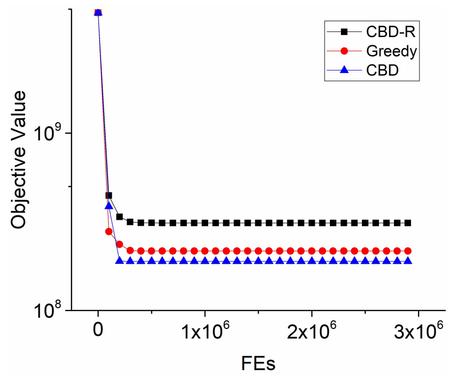
(g)



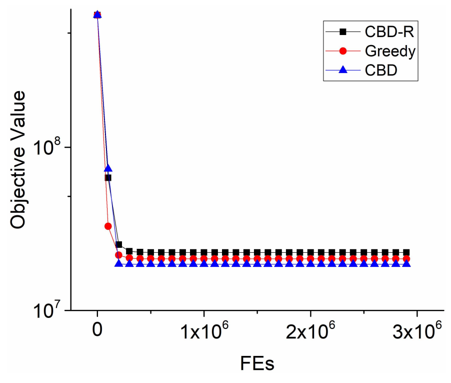
(h)



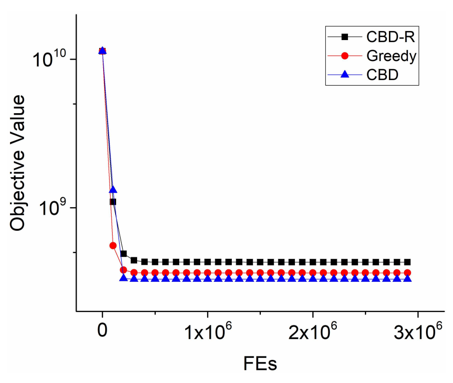
(i)



(j)



(k)



(l)

Fig. C. Convergence curves of CBD-R, Greedy, and CBD. (a) f_1, (b) f_2, (c) f_3, (d) f_4, (e) f_5, (f) f_6, (g) f_7, (h) f_8, (i) f_9, (j) f_{10}, (k) f_{11}, (l) f_{12}.

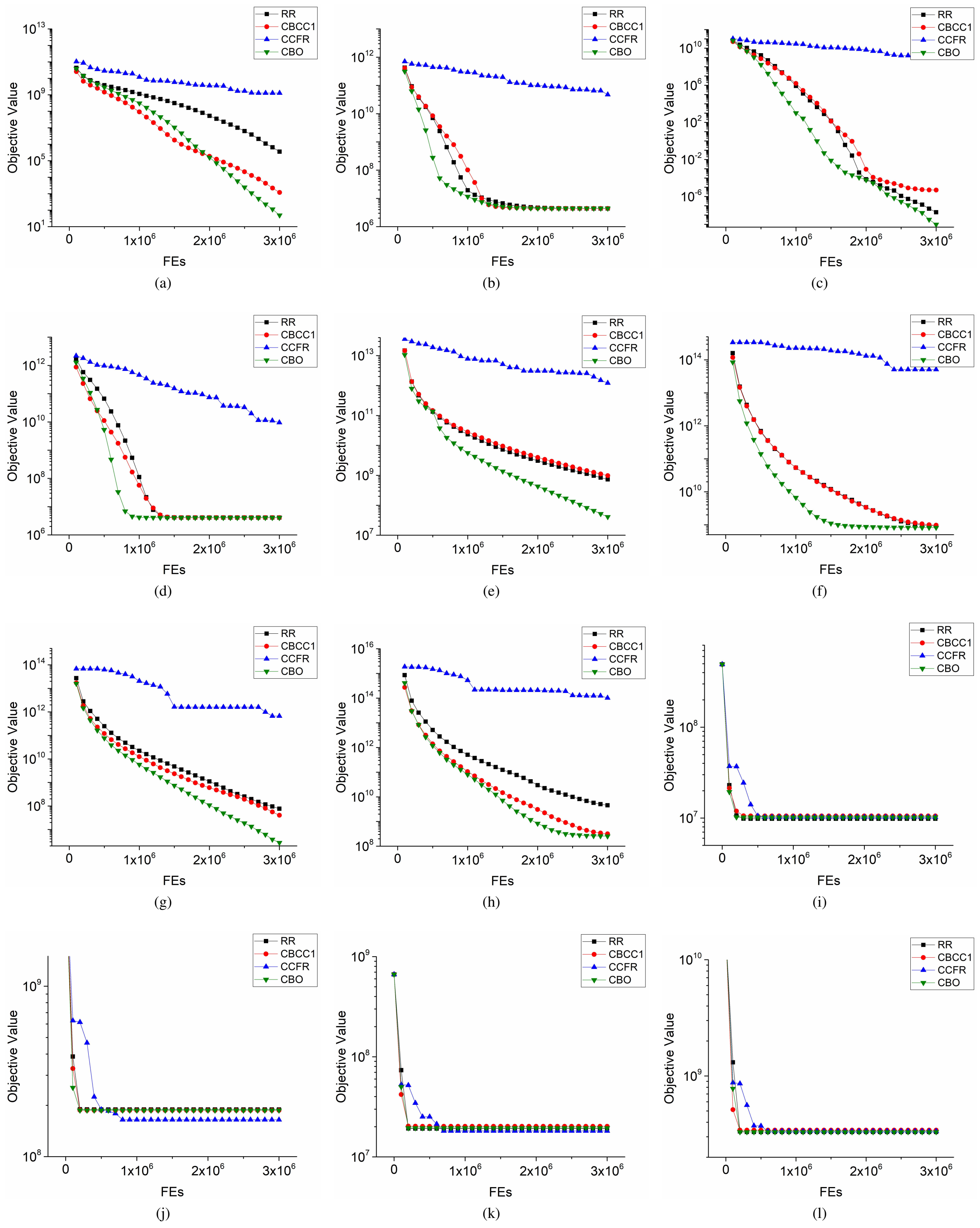


Fig. D. Convergence curves of RR, CBCC1, CCFR, and CBO. (a) f_1, (b) f_2, (c) f_3, (d) f_4, (e) f_5, (f) f_6, (g) f_7, (h) f_8, (i) f_9, (j) f_{10}, (k) f_{11}, (l) f_{12}.

III. Algorithm Complexity

Since most evolutionary algorithms (EAs) are stochastic algorithms, they usually requires a certain number of fitness evaluations to converge. Thus, in general, the time complexity of a conventional EA can be stated as $O(N\cdot(X+Y))$ where N represents the number of fitness evaluations, $O(X)$ represents the time complexity to generate a solution, $O(Y)$ represents the time complexity of evaluating a solution. In the research field of evolutionary computation, a commonly adopted assumption is that the complexity of evaluating a solution is much higher than the operations to generate a solution such as crossover and mutation, which means $O(Y) \gg O(X)$, and the evaluation process is also independent of the used algorithm. Thus, under the two assumptions, given the same number of fitness evaluations as the stopping criterion, theoretically time complexity of an evolutionary algorithm is $O(NY)$ where N is the total number of fitness evaluations. This is the reason that most competitions on continuous optimization problems set the number of fitness evaluations as the stopping criterion.

However, if we ignore the assumptions, different algorithms must have different time complexities to generate a solution. For an EA that is not CC-based algorithm, the time complexity comes from its own operations to generate a solution. For a CCEA, it depends on the applied optimizer. In CBCCO, we adopted CMA-ES as the optimizer. For a n-dimensional problem, the time complexity to generate a solution of CMA-ES is roughly $O(n^2)$, but in CBCCO, since the problem is divided, the time complexity to generate a solution is lower. Assume the n-dimensional problem is decomposed into M subcomponents, then roughly the size of each subcomponent $n_s = n/M$. Then, the time complexity to generate one solution in CBCCO is $O(n_s^2)$. Thus, the complete time complexity of CBCCO is $O(N\cdot(n_s^2+Y))$. Following the same way, we have summarized the time complexities of all compared algorithms in Table B. It should be notice that n_s may change in DCC and CCAS since the subcomponent size in these two algorithms is controlled either by some parameters or the random grouping method. For FEA, when it generate a sub-solution, the time complexity is $O(n_s^2)$. However, when the complete context vector is generated, each time only one value of one variable is changed, thus the complexity is $O(1)$. Since it requires a large number of FEs to in the competition process, its time complexity is in the between of $O(N\cdot Y)$ and $O(N\cdot(n_s^2+Y))$. Clearly, among the compared algorithms, DCC and CCAS have the lowest time complexity. However, whether the time complexities of RDG3 and CBCCO are higher than CSO, LLSO and SHADEILS is uncertain. It also depends on the number of subcomponents M, which is problem-dependent. If n_s^2 is smaller than n, i.e. $\frac{n}{M^2} < 1$, the time complexities of CBCCO and RDG3 are lower than CSO, LLSO, and SHADEILS. Otherwise, the time complexities of CBCCO and RDG3 are the highest.

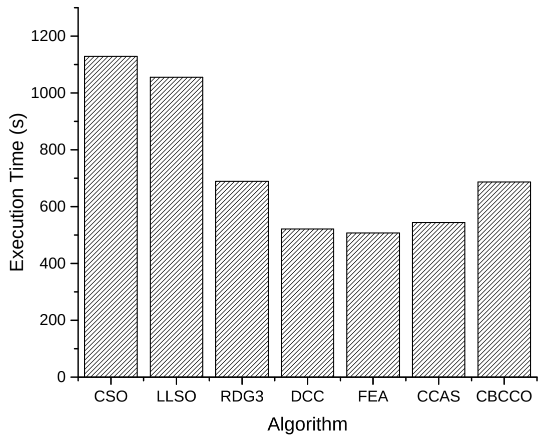
Besides the theoretical analysis about the time complexity, we have also recorded the execution time of each algorithm on f_1 and f_2. Since SHADEILS is implemented in Python, it is not compared in this experiment. [1] Other algorithms are implemented in C++. All algorithms run on the i7-9750H CPU with basic frequency 2.60GHz but overlocked to 3.98GHz. The results is shown in Fig. E in the form of histogram.

From Fig. E, we can find that there are three levels of the execution time. CSO and LLSO are in the highest level that they consumed more time than the other algorithms. RDG3 and CBCCO are in the middle level since they both applied CMA-ES as the optimizer and used the fixed grouping result. DCC, FEA, and CCAS are in the lowest level that they consumed less time than others. The reason that the time consumptions of DCC and CCAS are low is that both algorithms use differential evolution (DE) as their optimizers. The time complexity of DE to generate a solution is essentially lower than CMA-ES. The reason that FEA consumes little time is already explained previously. If we take the benchmark functions as example, actually the theoretical time complexity of CBCCO is larger

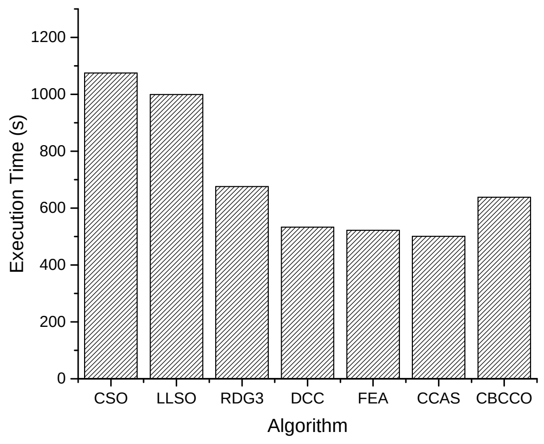
[1]The code of SHADEILS is provided by its author Molina.

TABLE B
Time Complexity of All The Compared Algorithms.

Methods	Complexity
CSO, LLSO, SHADEILS	$O(N\cdot(n+Y))$
RDG3, CBCCO	$O(N\cdot(n_s^2+Y))$
DCC, CCAS	$O(N\cdot(n_s+Y))$
FEA	$(O(N\cdot Y), O(N\cdot(n_s^2+Y)))$



(a)



(b)

Fig. E. Execution time of CSO, LLSO, RDG3, DCC, FEA, CCAS, and CBCCO. (a) f_1, (b) f_2.

TABLE C
COMPARISON OF THE BEST VALUE OF EACH ALGORITHM ON EACH FUNCTION AMONG THE 30 INDEPENDENT EXECUTIONS.

Func.	CSO	LLSO	SHADEILS	RDG3	DCC	FEA	CCAS	CBCCO
f_1	4.93E+08	5.38E+07	1.13E+05	9.62E+03	8.50E+08	1.93E+10	1.17E+09	**7.77E+01**
f_2	1.28E+09	4.21E+07	6.43E+06	4.35E+06	4.26E+09	3.16E+11	4.46E+10	**4.34E+06**
f_3	3.65E+08	8.66E+07	3.01E+04	6.14E-04	1.87E+09	3.27E+10	1.67E+09	**1.89E-14**
f_4	3.40E+09	8.93E+07	4.55E+06	3.99E+06	3.70E+10	5.60E+11	3.89E+10	**3.98E+06**
f_5	2.64E+11	1.03E+11	1.91E+10	8.72E+08	4.71E+11	6.15E+12	3.29E+11	**2.05E+07**
f_6	2.40E+12	1.14E+12	1.13E+11	8.45E+08	5.43E+12	6.96E+13	3.15E+12	**8.14E+08**
f_7	5.32E+11	3.08E+11	3.81E+10	4.00E+06	1.32E+12	1.26E+13	6.39E+11	**4.45E+05**
f_8	8.33E+12	4.26E+12	2.91E+11	1.16E+11	2.85E+13	2.13E+14	8.19E+12	**2.39E+08**
f_9	**1.60E+06**	2.49E+06	8.62E+06	7.70E+06	2.41E+07	6.75E+07	6.91E+06	7.16E+06
f_{10}	**4.33E+07**	5.31E+07	9.87E+07	1.33E+08	5.39E+08	1.05E+09	1.33E+08	1.18E+08
f_{11}	**4.17E+06**	5.47E+06	1.34E+07	1.35E+07	4.71E+07	1.25E+08	1.72E+07	1.49E+07
f_{12}	**6.90E+07**	8.60E+07	1.95E+08	2.93E+08	7.85E+08	2.66E+09	3.47E+08	2.25E+08

than CSO since $\frac{n}{M^2} > 1$. The reason that CBCCO takes less time than CSO is that generating a sub-solution with 50 variables by CMA-ES is faster than generating a solution with 1000 variables by CSO. The big 'O' notation only gives the approximation behaviour of the algorithm that has ignored the detailed operations of the algorithm. The comparison between DCC, CCAS, and CBCCO shows that the time complexity of a CCEA highly depends on the applied optimizer. In addition, the fact that CBCCO only takes slightly longer time than DCC and CCAS implies that although the time complexity of solution generation in CBCCO is higher than in CCAS or DCC, the solution evaluations really occupy a big proportion of the time consumption.

Generally, although CBCCO is not the fastest algorithm within all the tested algorithms, its execution time is only slightly longer than the best ones. Considering the advantage of CBCCO over other algorithms in the comparison of the objective value, we can make the conclusion that CBCCO is both effective and efficient in dealing with large-scale overlapping problems.

IV. COMPARISON OF BEST VALUE

The best objective values of the results of each compared algorithm among the 30 independent executions are shown in Table C corresponding to Table IV of the paper. Also, the best values of CBCCO2 on f_9 to f_{12} are shown in Table D corresponding to Table V of the paper.

From Table C and Table D,we can see that the best results show the same overall pattern as the mean and standard deviation results in Table IV and Table V of the main paper. CBCCO achieved better results than the other algorithms on f_1 to f_8 and CSO achieved better results on f_9 to f_{12}. When we adjusted the parameter of the applied optimizer CMA-ES, CBCCO2 achieved better results than CSO as shown in Table D. Thus, this comparison also demonstrates that CBCCO is very good even using the best value as the measurement of performance.

TABLE D
COMPARISON OF THE BEST VALUE OF CSO, LLSO, AND CBCCO2 ON f_9 TO f_{12}.

Func.	CSO	LLSO	CBCCO2
f_9	**1.60E+06**	2.49E+06	1.93E+06
f_{10}	4.33E+07	5.31E+07	**3.21E+07**
f_{11}	4.17E+06	5.47E+06	**3.34E+06**
f_{12}	6.90E+07	8.60E+07	**6.21E+07**