Large-Scale Global Optimization using Cooperative Coevolution with Variable Interaction Learning*

Wenxiang Chen, Thomas Weise, Zhenyu Yang, and Ke Tang**

Nature Inspired Computation and Applications Laboratory,
School of Computer Science and Technology,
University of Science and Technology of China.

Abstract. In recent years, Cooperative Coevolution (CC) was proposed as a promising framework for tackling high-dimensional optimization problems. The main idea of CC-based algorithms is to discover which decision variables, i.e, dimensions, of the search space interact. Non-interacting variables can be optimized as separate problems of lower dimensionality. Interacting variables must be grouped together and optimized jointly. Early research in this area started with simple attempts such as one-dimension based and splitting-in-half methods. Later, more efficient algorithms with new grouping strategies, such as DECC-G and MLCC, were proposed. However, those grouping strategies still cannot sufficiently adapt to different group sizes. In this paper, we propose a new CC framework named Cooperative Coevolution with Variable Interaction Learning (CCVIL), which initially considers all variables as independent and puts each of them into a separate group. Iteratively, it discovers their relations and merges the groups accordingly. The efficiency of the newly proposed framework is evaluated on the set of large-scale optimization benchmarks.

Key words: Variable Interaction Learning, Large-Scale Optimization, Numerical Optimization, Incremental Group Strategy, Cooperative Coevolution

1 Introduction

Evolutionary Algorithms (EAs) have been widely applied for solving both numerically and combinational optimization tasks [1]. Finding solutions for a problem usually becomes harder when the number of decision variables increases because of the *curse of dimensionality*. As a consequence, [2] reports that there is a rapid decline in performance of conventional EAs when dealing with large-scale problems. In order to improve the ability to solve high-dimensional optimization tasks, [3] proposes a coevolution approach for combinational optimization problems and [4] further generalizes the approach to a universal framework: Cooperative Coevolution (CC). CC algorithms tackle the curse of dimensionality with a divide-and-conquer method which separates the search space into subspaces of lower dimensionality. They therefore *decompose* the

* R. Schaefer et al. (Eds.): PPSN XI, Part II, LNCS 6239, pp. 300-309, 2010.

** This work is partially supported by two National Natural Science Foundation of China grants (No. 60802036 and No. U0835002).

Algorithm 1: $(subpop, pop, \boldsymbol{best}) \longleftarrow \mathbf{initializeCC}(NP)$

```
1 for i ⟵ 1 to N do
2     subpop_i ⟵ NP_i random one-dimensional samples from a given interval
3 pop ⟵ (subpop_1, ..., subpop_N)
4 best ← arg min f(pop)
5 return (subpop, pop, best)
```

decision vector into groups of variables which can be optimized cooperatively in cycles. After each cycle, the information gained in the separate optimization steps is joined for the next iteration. This approach yields good performance both in benchmark problems and real-world applications [4,5]. Despite this success, for the most important part of CC, the problem decomposition strategy, no satisfying solution has yet been developed. In this paper, we introduce a general, scalable, and highly efficient method for this purpose, called Cooperative Coevolution with Variable Interaction Learning (CCVIL).

The rest of this paper is organized as follows. In the next section, we give a short outline of the cooperative cooperationary idea in general and list related work in the area of problem decomposition in CC. CCVIL is then discussed in detail in Section 3 and experimentally studied by using a set of twenty large-scale benchmark problems in Section 4. It achieves excellent results in these experiments and frequently outperforms two related, state-of-the-art CC techniques as well as the optimizer which it uses internally. We finally conclude our paper in Section 5 where we also give pointers to future work.

2 Cooperative Coevolution

Many cooperative coevolutionary numerical optimization algorithms consist of three basic ingredients: [6]: *1)* A *decomposition* method used to divide the N-dimensional decision vector into groups $G_1 \ldots G_m$ of variables. Each such group is optimized with a separate subpopulation of the corresponding dimension $|G_i| < N$. *2)* In order to evaluate the fitness of the individuals from a certain subpopulation, a representative element from each of the other subpopulations is selected. In this *cooperation* step, a population of complete N-dimensional candidate solutions is constructed by concatenating the representatives to each element of the current subpopulation. *3)* An optimizer is applied to the population for (only) *optimizing* the decision variables in the current group.

In the conventional CC framework, optimizing a group with the corresponding subpopulation is called a *phase*. After finishing a *phase*, CC will turn to optimize the next group and start a new *phase*. One iteration over all groups constitutes a *cycle*. A CC algorithm performs several *cycles*. In Algorithms 1 and 2, we sketch the flow of a simple CC algorithm that treats the problem as completely separable [4].

2.1 Discovering Decision Variable Interactions

The decomposition strategy that identifies interacting decision variables and divides the search space into subspaces of lower dimensionality is the most important component

of CC algorithms. A function f is separable according to [7] if Equation 1 holds, i.e., if its global optimum can be reached by successive line search along the axes. Therefore, if a certain function is not separable, there must be interactions between at least two variables in the decision vector. Motivated by this, we provide definition of *interaction*: two decision variables i and j are *interacting* if there is a decision vector $\boldsymbol{x}$ whose i^{th} and j^{th} variable can be substituted with values x_i' and x_j' so that Equation 2 holds.

$$\arg\min_{(x_1,\ldots,x_N)} f(x_1,\ldots,x_N) = \left(\arg\min_{(x_1)} f(x_1,\cdots),\ldots,\arg\min_{(x_N)} f(\cdots,x_N)\right) \quad (1)$$

$$\begin{aligned}\exists \boldsymbol{x}, x_i', x_j' : \; & (f(x_1,\ldots,x_i,\ldots,x_j,\ldots,x_N) < f(x_1,\ldots,x_i',\ldots,x_j,\ldots,x_N)) \wedge \\ & (f(x_1,\ldots,x_i,\ldots,x_j',\ldots,x_N) > f(x_1,\ldots,x_i',\ldots,x_j',\ldots,x_N))\end{aligned} \quad (2)$$

The idea behind the decomposition of the decision variables in CC into groups $G_1, G_2, \ldots, G_m$ is that the fitness function f can be approximated as a linear combination of *component functions* $f_1, f_2, \ldots, f_m$. The domains of the functions f_i have the lower dimensionality $|G_i| < N$ since their results only depend on the variables in the corresponding group G_i. Although it is usually only possible to compute f but not its components f_i, the knowledge that the groups G_i can be optimized separately can speed up the optimization process significantly.

For discovering interactions between variables in the *decomposition* step, a simple method is suggested in [8]: Assume that $\boldsymbol{best}$ would be the vector of the best values for each decision variable discovered so far. After coevolutionary optimizing dimension i, the best individual $\boldsymbol{new}$ in the population $popcc$ of the optimizer only focusing on dimension i is extracted as well as a random candidate $\boldsymbol{rand}$ from the global population pop (with $\boldsymbol{new} \neq \boldsymbol{rand} \neq \boldsymbol{best}$). Based on these three vectors, two new candidate

$$x_j = \begin{cases} new_i & \text{if } j = i \\ best_j & \text{otherwise} \end{cases} \quad (3) \qquad x_j' = \begin{cases} new_i & \text{if } j = i \\ rand_k & \text{if } j = k \\ best_j & \text{otherwise} \end{cases} \quad (4)$$

vectors $\boldsymbol{x}$ and $\boldsymbol{x}'$ are created according to Equations 3 and 4 for testing whether dimensions i and k interact. The value at index k of $\boldsymbol{x}$ is better than the k^{th} value of $\boldsymbol{x}'$,

Algorithm 2: $\boldsymbol{best} \longleftarrow \mathbf{basicCC}(NP)$ (as introduced in [4])

```
(subpop, pop, best) ⟵ initializeCC(NP)
// Decomposition: implicitly performed based on separability assumption
while stoping criterion not met do // Optimization: Start a new cycle
    for i ⟵ 1 to N do // Start a new phase
        for j ⟵ 1 to NP_i do // Collaboration
            popcc_j ⟵ (best_1, ..., best_{i-1}, subpop_{i,j}, best_{i+1}, ..., best_N)
        (popcc, new) ⟵ optimizer(popcc, i)// Optimize the i^th subcomponent
        subpop_i ⟵ popcc_i
        best_i ← new_i
return best
```

since it comes from the vector of best known values $\boldsymbol{best}$ whereas x'_k stems from the random population member $\boldsymbol{rand}$. Optimizing dimension i should not influence this relation and $f(\boldsymbol{x}) \leq f(\boldsymbol{x}')$ would hold if the variables were separable. Therefore, if $f(\boldsymbol{x}') < f(\boldsymbol{x})$, i.e., $\boldsymbol{x}'$ is better than $\boldsymbol{x}$, there likely is an interaction between dimension i and k [8].

2.2 Related Work

In the early stage of the development of CC, researchers mainly adopted two simple decomposition methods: one-dimension based and splitting-in-half methods [4,9]. These two methods do not take the interaction between components into consideration. Thus, they cannot solve problems consisting of non-trivial variable interactions.

In order to mitigate this problem, a multi-level pyramidal genetic algorithm is utilized in [10] to better deal with multiple-choice scheduling. In the area of numerical optimization, Yang et al. proposed two effective CC-based algorithms, Differential Evolution using Cooperative Coevolution with adaptive grouping strategy (DECC-G) [6] and Multilevel Cooperative Coevolution (MLCC) [11]. DECC-G uses a constant group size, for instance 100, and randomly decomposes the high-dimensional variable vector into several such groups. These are then optimized with a certain EA. DECC-G with a small group size works properly for separable problems while highly nonseparable problems can better be solved with large group size. The problem of DECC-G is that the best group size is not known in advance. In order to overcome it, MLCC adopts a multilevel strategy for decomposition. It maintains a decomposer pool from which decomposers with different group sizes are selected depending on the problem under investigation and the stage of the evolution. For nonseparable problems, MLCC tends to select the decomposers with large group sizes. In the opposite case, MLCC prefers to choose the decomposers with smaller group sizes. However, determining a good pool of decomposers is hard in practice since the interaction between variables is usually not known beforehand. This, in turn, would lead to a waste of function evaluations.

By using the technique of learning variable interactions used in [8] and outlined here in Section 2.1 CC can become more adaptable. Nevertheless, the way it is used in [8] suffers severe shortcomings. For example, the maximum group size is limited to two, which rarely is the case real-world problems. Moreover, the *choice* of the dimensions i and k for interaction investigation (see Equations 3 and 4), frequently leads to the detection of non-existing interactions [8]. In [8], it is possible that dimension i and k are not optimized in successive phases. Thus, changes in other dimensions of $\boldsymbol{best}$ may take place in the mean time which violates the condition of Equation 2.

CCVIL overcomes the problem of the DECC-G and the MLCC algorithm, i.e., the group sizing, by using the interaction learning method of [8]. The choice of the dimensions i and k for interaction detection, however, is conducted in a way which prevents the discovery of non-existing interactions.

3 Cooperative Coevolution with Variable Interaction Learning

We propose a novel CC-framework called *Cooperative Coevolution with Variable Interaction Learning* (CCVIL) with incremental group sizes for solving large-scale op-

Algorithm 3: $groupInfo \longleftarrow$ Learning Stage of CCVIL

$K \longleftarrow 0$
$groupInfo \longleftarrow \{\{1\},\{2\},\ldots,\{N\}\}$// initially assume full separability
repeat // start a new learning cycle
  $\Pi \longleftarrow$ random permutation of dimension indices $\{1,2,\ldots,N\}$
  $K \longleftarrow K+1$
  $lastIndex \longleftarrow 0$
  $(subpop, pop, \boldsymbol{best}) \longleftarrow \mathbf{initializeCC}(3,3,\ldots,3)$// use $NP_i = 3\ \forall i \in 1..N$
  for $i = 1$ **to** N **do** // start a new learning phase, i.e., tackle next dimension
    if $lastIndex \neq 0$ **then**
      $G_1 \longleftarrow \mathbf{find}(groupInfo, \Pi_i)$ // find the group containing Π_i
      $G_2 \longleftarrow \mathbf{find}(groupInfo, lastIndex)$ // find group of last optimized variable
    if $i = 1 \vee (G_1 \neq G_2)$ **then**
      for $j = 1$ **to** NP **do**
        $popcc_j \longleftarrow (best_1,\ldots,best_{\Pi_i-1}, subpop_{\Pi_i,j}, best_{\Pi_i+1},\ldots)$
      $(popcc, \boldsymbol{new}) \longleftarrow \mathbf{optimizer}(popcc, \Pi_i)$ // any optimizer, we used JADE
      $subpop_{\Pi_i} \longleftarrow popcc_{\Pi_i}$
      $best_{\Pi_i} \leftarrow new_{\Pi_i}$
      if $lastIndex \neq 0$ **then**
        Compose $\boldsymbol{x}$ and $\boldsymbol{x'}$ according to Equations 3 and 4
        if $f(\boldsymbol{x}) < f(\boldsymbol{x'})$ **then** // interaction between dim. i and $lastIndex$?
          $groupInfo \longleftarrow ((groupInfo \setminus \{G_1\}) \setminus \{G_2\}) \cup (\{G_1 \cup G_2\})$
      $lastIndex \longleftarrow \Pi_i$ // *only* test successively optimized dimensions
until $(|groupInfo| = 1) \vee [(K > \check{K}) \wedge (|groupInfo| = N)] \vee (K > \hat{K})$
return $groupInfo$ // return the set of mutually separable groups of interacting variables

timization problems of separable, partially-nonseparable, and nonseparable character. Instead of setting the group sizes as a constant or defining a set of values from which to choose them, we allow the optimization process to adapt them by learning the interaction between the decision variables. The whole procedure of CCVIL is divided into two stages, the *learning stage* and *optimization stage* executed once in exactly this sequence. Both stages are divided into cycles and phases, similar to the conventional CC framework (see Section 2).

3.1 Learning Stage

The *learning stage* of CCVIL optimizes one dimension after the other, similar to the simple CC approach given in Algorithm 2. While doing this, it only tests the currently and the previously optimized dimension for interaction. Since only these two dimensions changed between the application of the interaction detection mechanism of [8], Equation 2 can never be violated and only becomes true for real interactions. Therefore, the flaw of detecting non-existent interactions is avoided. Before each learning cycle, the order of visiting the dimensions is randomly permutated so that each two dimension

have the same chance to be processed in a row. The details of the learning stage are presented in Algorithm 3.

The efficiency of CCVIL becomes clear from a stochastic point of view. The probability of placing any two (possibly interacting) dimensions i and j in an N-dimensional problem adjacently in one random permutation Π is $2/N$. The probability $p_{capt}(K)$ that this happens in at least once in K learning cycles then is given in Equation 5.

$$p_{capt}(K) = 1 - (1 - 2/N)^K \tag{5}$$

In a 1000-dimensional problem, the probability of putting any two (possibly interacting) variables adjacently in a random permutation and examining interaction between them during $K = 500$ cycles is already $p_{capt}(K) = 0.6325$ and raises to 0.7984 for 800 cycles. Given a limited number of fitness function evaluations, as many learning cycles as possible should thus be performed. Therefore, the population size and generation limit of the internal optimizer should be as small as possible during the learning stage (3 and 1, respectively, in this work). CCVIL issues an independent restart for each cycle to prevent possible loss of population diversity during the learning stage.

We furthermore set a lower and an upper threshold for the number of learning cycles: $\check{K}$ and $\hat{K}$. If CCVIL cannot detect interactions between any two variables in the first $\check{K}$ cycles of learning stage, it assumes that the problem is fully separable or that the interactions are rather weak and immediately switches to the optimization stage. Additionally, a transition to the optimization stage is enforced after $\hat{K}$ cycles even if not all interactions have been discovered in order to limit the runtime used for learning. As default settings, we recommend 10 for $\check{K}$ and to set $\hat{K}$ to the number of cycles needed to achieve 80% for $p_{capt}(\hat{K})$ (see Equation 5). This number can be computed by $\hat{K} \geq \log(1 - 0.8)/\log(1 - 2/N)$. However, $\hat{K}$ should not lead to a consumption of more than 60% of the function evaluations in the learning stage.

3.2 Optimization Stage

The user of our CC framework is completely free in the choice of the optimizer to be used for the variables grouped together. During both, the interaction learning and the optimization stage of CCVIL, we apply JADE for this purpose. JADE is an enhanced variant of Differential Evolution (DE) [12,13] with improved speed and reliability compared with plain DE [14]. In each phase of the optimization stage of CCVIL, the optimizer is applied to the complete subspace defined by one group $G_i \in groupInfo$ (whereas the remaining variables of the candidate vectors are constant and correspond to the representatives from the other groups).

During the learning stage, CCVIL may divide the variables into groups of different sizes. In the optimization step, the population size NP_i and number of generations Gen_i granted to the optimizer for processing the group G_i should depend on its size. As a rule of thumb, we set $Gen_i = \min\{|G_i| + 5, 500\}$. Whereas the population size NP_i of JADE is set as small as possible during the interaction learning, in the optimization stage, we apply an adaptive strategy. The initial value here is $NP_i = |G_i| + 10$, which is sufficient for unimodal problems. After the population loses its diversity and ceases to improve, an independent restart with thrice the population size is performed

as suggested in [15]. This is done when the *relative* fitness improvement of the current cycle compared to the previous on is below 10^{-2}, i.e., $(f_{K-1} - f_K)/f_K < 10^{-2}$.

Furthermore, the difficulties in solving the component functions may vary a lot. Therefore, it is reasonable to stop optimizing converged groups, when no improvements can be achieved for a certain number of cycles (in the context of this work, we set this threshold to five).

4 Experimental Studies

4.1 Experimental Setup

For benchmarking CCVIL, we choose the set of twenty 1000-dimensional functions provided by in [7]. These functions represent high-dimensional problems with different degrees of variable interactions. This makes them especially suitable to test the ability of our algorithm which was designed for tackling this kind of tasks. We compared CCVIL to DECC-G [6], MLCC [11], and JADE [14]. In the experiments, we grant three million fitness function evaluations to each algorithm run. We fixed the population size in JADE to 1000 and used default parameter settings for DECC-G and MLCC are obtained from the related publications [6,11]. For each algorithm and benchmark function, 25 independent runs were performed.

4.2 Benchmark Functions and Learned Groups

In Table 1, we list the characteristics of the benchmark functions applied in our experiments. The column *Sep* denotes functions which are separable according to Equation 1. Most of the non-separable functions consist of mutually separable groups of interacting variables. f_{10} is the sum of ten rotated instances of Rastrigin's function applied to groups of 50 decision variables each and one non-rotated instance of the same function applied to the remaining 500 decision variables. Since the plain Rastrigin's function is separable (and the rotated version is not), an ideal interaction learning stage would thus discover 500 separable groups of size 1 and plus 10 groups of size 50, as listed in column *Groups (real)*. f_{18} is composed of 20 Rosenbrock's functions, each applied to 50 decision variables, leading to 20 "real" groups.

In the last column of Table 1, we noted the median of the number of groups discovered by CCVIL during the 25 runs. From these results, we can clearly see that CCVIL most often is able to represent the interactions between the variables correctly. Due to the limitation $\hat{K}$ imposed on the runtime of the learning stage, it may not discover all interactions, i.e., may not merge all interacting groups, and thus, yield a slightly higher number of groups.

Furthermore, we notice that, for most of the benchmark functions related to Ackley's function (f_3, f_6, f_{11}), CCVIL combines more variables than expected. The reason for this is that Ackley's function is separable according to Equation 1, but not *additively separable* [16], i.e., it cannot be divided into an exact arithmetic sum of component functions, as pointed out in [17]. Therefore, our algorithm correctly picks up interactions since it aims at representing the fitness function as linear combination of component functions.

Table 1. The main characteristics of the 20 benchmark functions [7] used in this paper.

	Function	Sep	Multi modal	Groups real	Groups found
f_1	Shifted Elliptic Function	Yes	No	1000	1000
f_2	Shifted Rastrigin's Function	Yes	Yes	1000	1000
f_3	Shifted Ackley's Function	Yes	Yes	1000	969
f_4	Single-group Shifted 50-rotated Elliptic Function	No	No	951	963
f_5	Single-group Shifted 50-rotated Rastrigin's Function	No	Yes	951	952
f_6	Single-group Shifted 50-rotated Ackley's Function	No	Yes	951	921
f_7	Single-group Shifted 50-dimensional Schwefel's	No	No	951	952
f_8	Single-group Shifted 50-dimensional Rosenbrock's	No	Yes	951	1000
f_9	10-group Shifted 50-rotated Elliptic Function	No	No	510	627
f_{10}	10-group Shifted 50-rotated Rastrigin Function	No	Yes	510	516
f_{11}	10-group Shifted 50-rotated Ackley Function	No	Yes	510	501
f_{12}	10-group Shifted 50-dimensional Schwefel's	No	No	510	522
f_{13}	10-group Shifted 50-dimensional Rosenbrock's	No	Yes	510	1000
f_{14}	20-group Shifted 50-rotated Elliptic Function	No	No	20	232
f_{15}	20-group Shifted 50-rotated Rastrigin's Function	No	Yes	20	37
f_{16}	20-group Shifted 50-rotated Ackley Function	No	Yes	20	39
f_{17}	20-group Shifted 50-dimensional Schwefel's Function	No	No	20	42
f_{18}	20-group Shifted 50-dimensional Rosenbrock	No	Yes	20	1000
f_{19}	Shifted Schwefel's Function 1.2	No	No	1	1
f_{20}	Shifted Rosenbrock's Function	No	Yes	1	1000

The decision variables of Rosenbrock's function, although entirely nonseparable, exhibit only a very weak interaction. Our algorithm discovers that functions f_8, f_{13}, f_{18} and f_{20} can best be treated as separable problems, i.e., problems with 1000 independent decision variables.

4.3 Comparison with other CC-based algorithms and JADE

In Table 2, we provide the results of the comparison of our algorithm with DECC-G, MLCC, and plain JADE. We list the mean results of the 25 runs for each benchmark as well as the standard deviations. The columns named R denote the outcomes of a two-tailed Mann-Whitney U test [18] with 5% significance level. A *W* in column R_1 stands for statistically significant win of CCVIL against both, DECC-G and MLCC, a *L* a loss, and "–" means that no significant difference was found. Column R_2 represents the same comparison between CCVIL and the native JADE (used as optimizer in CCVIL).

Table 2 shows that CCVIL is clearly superior to DECC-G and MLCC. It outperforms them in 15 benchmarks and only loses on f_3, the separable but not *additively* separable Ackley function. CCVIL also wins against its internal optimizer JADE alone in ten out of 20 benchmark functions and loses in six. Especially for separable and highly non-separable functions, CCVIL proofs to be advantageous. The performance of CCVIL is worse than its JADE in most single-group non-separable functions. The reason is the structure of the benchmark set [7] where a large factor (1 million) is put in

Table 2. Comparison with other CC-based algorithms and plain JADE.

	CCVIL		DECC-G		MLCC		R_1	Naive JADE		R_2
	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev		Mean	Std Dev	
f_1	1.55e-17	7.75e-17	2.93e-07	8.62e-08	1.53e-27	7.66e-27	–	1.57e+04	1.38e+04	W
f_2	6.71e-09	2.31e-08	1.31e+03	3.24e+01	5.55e-01	2.20e+00	W	7.66e+03	9.67e+01	W
f_3	7.52e-11	6.58e-11	1.39e+00	9.59e-02	9.86e-13	3.69e-12	L	4.52e+00	2.41e-01	W
f_4	5.00e+12	3.38e+12	9.62e+12	3.43e+12	1.70e+13	5.38e+12	W	6.14e+09	3.81e+09	L
f_5	1.76e+08	6.47e+07	2.63e+08	8.44e+07	3.84e+08	6.93e+07	W	1.35e+08	1.21e+07	L
f_6	2.94e+05	6.09e+05	4.96e+06	8.02e+05	1.62e+07	4.97e+06	W	1.94e+01	1.79e-02	–
f_7	8.00e+08	2.48e+09	1.63e+08	1.38e+08	6.89e+05	7.36e+05	–	2.99e+01	3.30e+01	–
f_8	6.50e+07	3.07e+07	6.44e+07	2.89e+07	4.38e+07	3.45e+07	–	1.19e+04	4.92e+03	L
f_9	6.66e+07	1.60e+07	3.21e+08	3.39e+07	1.23e+08	1.33e+07	W	2.70e+07	2.08e+06	L
f_{10}	1.28e+03	7.95e+01	1.06e+04	2.93e+02	3.43e+03	8.72e+02	W	8.50e+03	2.30e+02	W
f_{11}	3.48e+00	1.91e+00	2.34e+01	1.79e+00	1.98e+02	6.45e-01	W	9.29e+01	9.66e+00	W
f_{12}	8.95e+03	5.39e+03	8.93e+04	6.90e+03	3.48e+04	4.91e+03	W	6.21e+03	1.34e+03	–
f_{13}	5.72e+02	2.55e+02	5.12e+03	3.95e+03	2.08e+03	7.26e+02	W	1.87e+03	1.11e+03	W
f_{14}	1.74e+08	2.68e+07	8.08e+08	6.06e+07	3.16e+08	2.78e+07	W	1.00e+08	8.84e+06	L
f_{15}	2.65e+03	9.34e+01	1.22e+04	9.10e+02	7.10e+03	1.34e+03	W	3.65e+03	1.09e+03	W
f_{16}	7.18e+00	2.23e+00	7.66e+01	8.14e+00	3.77e+02	4.71e+01	W	2.09e+02	2.01e+01	W
f_{17}	2.13e+04	9.16e+03	2.87e+05	1.97e+04	1.59e+05	1.43e+04	W	7.78e+04	5.87e+03	W
f_{18}	1.33e+04	1.00e+04	2.46e+04	1.05e+04	7.09e+03	4.77e+03	–	3.71e+03	9.58e+02	L
f_{19}	3.52e+05	2.04e+04	1.11e+06	5.00e+04	1.36e+06	7.31e+04	W	3.48e+05	1.67e+04	–
f_{20}	1.11e+03	3.04e+02	4.06e+03	3.66e+02	2.05e+03	1.79e+02	W	2.06e+03	2.01e+02	W

front of the nonseparable component function. If even a single interaction is not discovered by CCVIL, it will perform worse than JADE which treats the fitness function as completely nonseparable. This fact leads us to the conclusion that CCVIL can achieve much better results if the learning phase can proceed sufficiently long to discover all interactions. To find a good strategy to distribute runtime between the learning and the optimization stage of CCVIL is thus an interesting point for future research.

5 Conclusions and Future Work

In this paper, we introduced a novel CC framework called Cooperative Coevolution with Variable Interaction Learning, or CCVIL for short. In the related work study, we showed that currently no efficient method for finding groups of interacting variables in CC exists. CCVIL fills this gap with a two-stage approach: In a learning step, the interactions between the decision variables of a (potentially very high-dimensional) search space are detected. In the second stage, these groups are optimized according to the traditional CC model. We showed in an experimental study based on twenty 1000-dimensional benchmark functions with different degrees of variable interaction and group sizes that CCVIL can outperform two very efficient, state-of-the-art CC approaches as well as its internal optimizer JADE.

The experiments also showed the drawback of CCVIL: finding the optimal distribution of runtime between the learning and the optimization stage is an open question. Here, we could only provide some simple rules-of-thumb, but this problem surely

will be subject of our future work. In order to reduce the overall learning time, we will furthermore explore generating the permutations Π in our algorithm according to a deterministic scheme instead of creating them randomly. Additionally, we wish to experiment with possibly more efficient optimizers such as CMA-ES [19] as internal optimizers.

References

1. Sarker, R., Mohammadian, M., Yao, X.: Evolutionary Optimization. Kluwer Academic Publishers Norwell, MA, USA (2002)
2. van den Bergh, F., Engelbrech, A.P.: A cooperative approach to particle swarm optimization. IEEE Transactions on Evolutionary Computation **8**(3) (2004) 225–239
3. Husbands, P., Mill, F.: Simulated co-evolution as the mechanism for emergent planning and scheduling. In: 4th Intl. Conf. on Genetic Algorithms, Morgan Kaufmann (1991) 264–270
4. Potter, M.A., De Jong, K.A.: Cooperative coevolution: architecture for evolving coadapted subcomponents. Evolutionary Computation **8**(1) (2000) 1–29
5. Yong, C., Miikkulainen, R.: Cooperative coevolution of multi-agent systems. University of Texas at Austin, Austin, TX (2001)
6. Yang, Z., Tang, K., Yao, X.: Large scale evolutionary optimization using cooperative coevolution. Information Sciences **178**(15) (2008) 2985–2999
7. Tang, K., Li, X., Suganthan, P.N., Yang, Z., Weise, T.: Benchmark functions for the CEC'2010 special session and competition on large scale global optimization. TR, NICAL, USTC, Hefei, Anhui, China (2009) `http://nical.ustc.edu.cn/cec10ss.php`.
8. Weicker, K., Weicker, N.: On the improvement of coevolutionary optimizers by learning variable interdependencies. In: Proc. 1999 Congress on Evolutionary Computation (CEC'99). IEEE Press. (1999) 1627–1632
9. Potter, M.A., De Jong, K.A.: A cooperative coevolutionary approach to function optimization. In: 3rd Conf. on Parallel Problem Solving from Nature. Volume 2. (1994) 249–257
10. Aickelin, U.: A Pyramidal Evolutionary Algorithm with Different Inter-Agent Partnering Strategies for Scheduling Problems. In: GECCO Late-Breaking Papers. 1–8
11. Yang, Z., Tang, K., Yao, X.: Multilevel cooperative coevolution for large scale optimization. In: IEEE Congress on Evolutionary Computation, IEEE Press (2008) 1663–1670
12. Price, K., Storn, R., Lampinen, J.A.: Differential Evolution: A Practical Approach to Global Optimization. Springer-Verlag (2005)
13. Chakraborty, U.K., ed.: Advances in Differential Evolution. Springer, Berlin (2008)
14. Zhang, J., Sanderson, A.C.: JADE: adaptive differential evolution with optional external archive. IEEE Transactions on Evolutionary Computation **13**(5) (2009) 945–958
15. Auger, A., Hansen, N.: A restart CMA evolution strategy with increasing population size. In: IEEE Congress on Evolutionary Computation. Volume 2., IEEE Press (2005)
16. Streeter, M.: Upper bounds on the time and space complexity of optimizing additively separable functions. In: Genetic and Evolutionary Computation–GECCO 2004, Springer (2004) 186–197
17. Hansen, N., Kern, S.: Evaluating the CMA evolution strategy on multimodal test functions. In: Parallel Problem Solving from Nature – PPSN VIII, Springer (2004) 282–291
18. Mann, H.B., Whitney, D.R.: On a test of whether one of two random variables is stochastically larger than the other. The Annals of Mathematical Statistics **18**(1) (1947) 50–60
19. Hansen, N., Müller, S.D., Koumoutsakos, P.: Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evolutionary Computation **11**(1) (2003) 1–18