A Study in Overlapping Factor Decomposition for Cooperative Co-Evolution

Elliott Pryor
Gianforte School of Computing
Montana State University
Bozeman, MT 59717
elliott.pryor@student.montana.edu

Amy Peerlinck
Gianforte School of Computing
Montana State University
Bozeman, MT 59717
amy.peerlinck@student.montana.edu

John Sheppard
Gianforte School of Computing
Montana State University
Bozeman, MT 59717
john.sheppard@montana.edu

***Abstract*—Large scale global optimization is where we seek to optimize a function with a high number of decision variables. Cooperative co-evolutionary algorithms (CCEA) improve optimization performance on these large scale problems through a divide and conquer approach. How the problem is divided can have a large impact on optimization performance. We provide two new decomposition methods that are capable of generating overlapping groups of variables. We apply a generalized CCEA called factored evolutionary algorithm (FEA) that is capable of optimizing and combining overlapping sub-problems. We compare results to existing methods to analyze the effect of introducing overlap in the sub-problems. We use five functions from the CEC'2010 benchmark suite as a base of comparison for all algorithms. We show that overlap can be beneficial for optimizing problems that are not fully separable.**

***Index Terms*—cooperative co-evolution, particle swarm optimization, problem decomposition, factored evolutionary algorithms**

I. Introduction

Function optimization is a problem that pervades many real-world tasks. Finding optimal solutions through exact approaches is often intractable; therefore, meta-heuristic techniques, such as stochastic hillclimbing [1] and population-based algorithms, are often used. Among the population-based approaches, Genetic Algorithms (GA) [2] and Particle Swarm Optimization (PSO) [3] have proven widely applicable and effective in fields such as economics [4], aerospace engineering [5], and precision agriculture [6]. However, for problems with a large number of dimensions, it may be difficult for these algorithms to find good solutions. To address this problem, Potter and De Jong proposed Cooperative Co-Evolutionary Algorithms (CCEA) [7]. CCEA decomposes the problem into 1-dimensional disjoint sub-problems. These sub-problems are then solved individually and combined into a global solution. CCEA can use any evolutionary algorithm (EA) as the optimizer for the sub-problems, e.g., particle swarm optimization [8], genetic algorithm [7], or differential evolution [9].

CCEA has been extended beyond the original 1-dimensional sub-problems to allow for different disjoint problem decompositions, where the set of optimization variables is divided into different groups [8], [10], [11]. The problem decomposition has been shown to influence the efficacy of the associated CCEA methods [12], [13]. We also use the term "factor" throughout this paper to refer to one of the decomposed sub-problems, and "factor architecture" to refer to the set of factors.

The Factored Evolutionary Algorithm (FEA) introduces the inclusion of overlapping sub-populations corresponding to the factor decomposition and proposes a general framework for associated decomposition-based methods such as CCEA [14]. In overlapping FEA, instead of the variables being decomposed into disjoint factors, variables are allowed to belong to multiple sub-populations. The resulting overlap requires that factors compete to be a part of the global solution. After competition, a new global solution is shared with all factors by being reinserted into the associated subpopulations. In previous research, overlapping subpopulations have been shown to improve upon co-evolutionary results, especially in the context of combinatorial optimization [14]. The effect of overlap between factors on the performance of real-valued, large scale optimization is the focus of this paper.

One popular method for problem decomposition in CCEA is Differential Grouping (DG) [10], [11]. DG uses variable interactions to determine subdivision, grouping together highly interacting variables and minimizing interactions between groups. However, DG does not allow for overlapping groups, which motivates developing new decomposition methods that can create overlapping factors for FEA.

Several studies have found that decomposition strategies for CCEA and FEA methods can impact overall model performance [12], [14], [15]. As a result, robust decomposition strategies have been proposed, which attempt to capture interactions within groups to enable efficient optimization [10], [11], [16], [17], [18]. In support of this prior work, this paper has two main contributions: an empirical examination of the effect of overlap in FEA architectures on large scale, continuous optimization problems and the introduction of two novel overlapping grouping methods to further assess the importance of these decomposition methods.

The rest of the paper is organized as follows: in Section II we provide background information about the problem and approaches. We then provide related work in Section III, and we present our novel decomposition methods in Section IV. We describe our experimental setup in Section

V. Results are presented (Section VI) and discussed (Section VII) before providing concluding remarks and potential future work (Sections VIII, IX).

II. Background

A. Optimization

Real valued functions are often optimized by using approaches such as hill-climbing [1] and gradient descent [19]. Alternatives utilizing population-based meta-heuristics are becoming increasingly popular, especially for high-dimensional problems. Here, we focus on the cooperative co-evolutionary methods since they have been demonstrated to be efficient and effective at solving such large scale problems.

1) Cooperative Co-Evolutionary Algorithms: In the context of high-dimensional problems, co-evolutionary algorithms have become increasingly popular [20]. Co-evolutionary approaches divide the population into subpopulations corresponding to subsets of the solution. The subpopulations are then optimized separately, using the population-based algorithm of choice, before combining them to find the final solution. There are two broad types of co-evolutionary algorithms, competitive and cooperative co-evolution, where the former is based on a predator-prey model [21], and the latter is based on symbiotic relationships in biology [7]. In this paper we will focus on cooperative methods.

Potter and De Jong based CCEA on the biological phenomenon of different species cooperating to improve each other's growth and quality of life [7]. In their initial version of CCEA, an n-dimensional problem is decomposed into n 1-dimensional subproblems, each with its own subpopulation that is optimized individually. The subpopulations are then evaluated based on the subproblem, as well as the overall objective function. Finally, the best individuals from each subpopulation are combined to create the final solution. This approach was extended to create k m-dimensional subproblems, where $m \ll n$ [8]. Yang *et al.* further introduced random grouping, where decision variables are reassigned randomly to different subpopulations in each cycle [9]. The idea of this approach was to capture variable interaction without explicitly determining which variables interact. This random grouping approach was then applied to CCPSO by Li *et al.* with CCPSO-rg-aw along with adaptive weighting [9], [22] and CCPSO2 which additionally included a new PSO update rule to improve performance [23]. Zhang and Chiang also proposed CPSOATT which uses consensus based PSO and fine-tuning to escape local optima to obtain high quality optimization results [24].

2) Factored Evolutionary Algorithms: In previous research, a framework for cooperative co-evolutionary methods was created termed Factored Evolutionary Algorithms (FEA), which included the novel concept of using overlapping subpopulations [14]. This is not to be confused with the Multi-factorial Evolutionary Algorithm (MFEA) [25], which presents a novel optimization paradigm to help solve multiple, potentially unrelated tasks. For example, MFEA has been used to solve different problems in optimizing resources in cloud computing, which is in contrast to FEA, which refers to subdividing a single problem into subproblems.

In FEA, the idea of subpopulations is extended to that of factors that overlap, which means a single decision variable can belong simultaneously to more than one factor. If a variable belongs to several factors, a decision has to be made regarding which subpopulation holds the best representation of that variable. In order to determine this, competition is introduced between the overlapping subpopulations, before sharing the the optimal value for that variable across all subpopulations.

More formally, FEA divides the decision variables in an optimization problem into s factors $\mathcal{F}_i \subset \mathcal{X}$, $i = 1, \ldots, s$ and assigns a subpopulation S_i to each factor $\mathcal{F}_i$. This representation can be adapted to fit both single population EA's and CCEA's by setting $s = 1$ or setting $s = |\mathcal{X}|$ and $\bigcup_{i=1}^{s} S_i = \mathcal{X}$, $S_i \cap S_j = \emptyset,\ \forall i, j\ i \neq j$. The proposed framework denotes a factor's solution as a partial solution, whereas the solution to the entire problem is known as the global solution G. G is derived by assembling the best members of the subpopulations through the previously mentioned competition process and is then used to share values for variables that are not part of the factor to create a full context for evaluation. When optimizing a factor to find a partial solution, only the variables that are part of the subpopulation are changed, while the other variables are set to be constant based on the context derived from G.

More specifically, competition occurs between overlapping factors to set the shared variable(s). The partial solution is injected into the global solution G for evaluation, where each subpopulation containing the shared variable substitutes its values for that variable into G for evaluation. Whichever factor results in the best global fitness for that specific variable gets to inject the variable value in the global solution. The resulting G is then shared with all factors.

B. Decomposition

An important aspect of cooperative co-evolutionary algorithms is the way the problem is decomposed into factors. Chen *et al.* [26] explored a Variable Interaction Learning (VIL) approach to problem decomposition [12] in an attempt to determine if correctly identifying variable interaction improves performance. VIL uses a bottom-up approach to finding variable interaction, merging interacting variables into groups based on random permutations of the variables. They concluded that when the problem decomposition identified at least 10% of the total number of interactions, CCEA benefits from the decomposition strategy.

Because of the bottom-up approach, certain combinations of variables are not examined, and the method is not capable of determining when variables are partially or fully separable, where a function is said to be fully separable if all variables are independent of one another and partially separable if groups of variables are independent of one another [27]. It is because of these limitations we chose not to include this method in our study.

C. Differential Grouping

Differential Grouping (DG) is based on a process of identifying partial separability [10]. It was introduced to automatically decompose a problem into subgroups to be used in CCEA. When using CCEA for large-scale optimization problems, DG has become a popular method to determine which variables should belong to which factors. The decomposition strategy allows for variables that are interacting to belong to the same group, while minimizing interdependence between factors. Interaction is determined by measuring how much the function changes between pairs of points. We measure Δ_1 by changing the value of x_i and evaluating the function at both points. We then change x_j to a different value and then compute Δ_2 by re-evaluating the function at the points from Δ_1. Geometrically speaking, in $\mathcal{R}^2$ this is evaluating Δ_1 and Δ_2 along parallel edges of a rectangle. The authors prove that x_i and x_j are not independent if the difference between Δ_1 and Δ_2 is large enough.

DG uses the aforementioned approach to create groups of variables. It starts by holding a variable constant and performing a pair-wise comparison to all other variables, where the other variables' values are changed. Two values Δ_1 and Δ_2 are calculated by varying the values for the variables that are being evaluated. If $|\Delta_1 - \Delta_2| > \epsilon$, the variables are said to interact, and the corresponding variable is removed from the set of remaining decision variables. The ϵ parameter plays an important role in determining interactions: a smaller ϵ will detect weaker interactions. Omidvar *et al.* [28] show that finding an underlying structure of interaction to create the factors improves CCEA's performance compared to the random grouping approach by Yang *et al.* [9].

III. Related Work

Similar to FEA, the idea of combining competitive and cooperative methods was used by Goh and Tan [21] who combined the power of the two types of co-evolutionary algorithms to create a new competitive-cooperative co-evolutionary algorithm (COEA). In their algorithm, each subpopulation competes to represent a specific factor. The resulting factors then cooperate to find a better overall solution. This particular process enables the algorithm to find interdependencies among the different subpopulations, where similar subpopulations represent similar factors. The competitive pressure also enables a structure to emerge, which obviates the need for an algorithm such as DG. COEA was used for multi-objective optimization (MOO), and more specifically dynamic MOO. Their results showed that while their approach did improve optimization for dynamic environments, there were no significant improvements with static MOO. Since we are not focusing on MOO in this paper, COEA was not compared in our study.

Liu *et al.* [29] also use a soft grouping method for CCEA, where variable membership in a group is decided probabilistically. In this approach all variables belong to each group, i.e., groups are fully overlapping, but how much influence each of these variables has within a group is decided by a randomly assigned probability. Therefore, this approach is not a decomposition approach. That said, their approach was found to perform well on the CEC 2010 benchmark functions [27], especially with respect to non-separable functions.

Ma *et al.* [30] provide a comprehensive overview of different decomposition strategies for CCEA. They note that dynamic group assignments based on variable interaction learning improves CCEA results compared to static and random groupings. Five different groups of interaction learning based decomposition are defined: perturbation (DG), distribution model (FEA), statistical model, approximate model, and linkage adaptation. The latter three are not considered in this work since approximate modeling is used for problems where evaluating the objective function is too expensive and needs to be approximated [31], linkage adaptation methods influence operators of EA's directly, and due to the high computational expense of computing the statistical model Maximum Entropic Epistasis [13]. For example, Schaffer and Morishima [32] add a punctuation flag to the chromosome to indicate the crossover point, thus effectively grouping each chromosome.

Since its inception, there have been adjustments to the DG algorithm. In 2017, Omidvar, *et al.* [11] introduced DG2, a version of DG that sets the ϵ threshold automatically and reduces the number of necessary fitness evaluations to find the groups of interacting variables. Sun *et al.* [18] also adjust the ϵ threshold parameter in their recursive DG. Recursive DG looks at a pair of a set of variables and divides each subset recursively to increase efficiency. This approach was adjusted further by Yang *et al.* [33] to create efficient recursive DG, which further exploits gathered information about the interaction between variable groups. Both of these adjustments focus on improving the efficiency of DG but have no effect on the grouping itself.

IV. Methods

We present our new decomposition approaches for FEA architectures. We present Overlapping Differential Grouping (ODG) in Section IV-1 and a tree based decomposition in Section IV-2. Both methods can produce overlapping variable decompositions. ODG is an extension of DG [28] that continues to consider variables that have been marked as interacting instead of removing them (as in DG) allowing for overlapping groups. Tree based decomposition considers variables as nodes in a tree, and creates factors based on adjacent nodes in the tree.

1) Overlapping Differential Grouping: All of the versions of DG discussed in Section II-C are intended to identify disjoint factors in large-scale optimization. Here we present an alternative approach that adjusts DG to allow factors to overlap. Algorithm 1 shows our Overlapping Differential Grouping (ODG) algorithm. To find overlapping factors, instead of removing a variable once it has been found to interact, that variable remains in the set of decision variables. This allows a variable to be marked as interacting with multiple other variables: which leads to overlapping factors. This means that the interaction step of DG will be performed on all decision variables; however, for each such variable, only subsequent

Algorithm 1: Overlapping Differential Grouping

input : function f, lower bounds lb, upper bounds ub, number of dimensions n, threshold ϵ
initialize: $dims \leftarrow \{1, 2, \ldots, n\}$, $sepvars \leftarrow \{\}$, $groups_o \leftarrow \{\}$

1 **for** $i \in dims$ **do**
2 $grp \leftarrow \{i\}$
3 **for** $j \in dims \wedge j > i$ **do**
4 $p_1 \leftarrow lb \times ones(1, n)$
5 $p_2 \leftarrow p_1$
6 $p_2(i) \leftarrow ub$
7 $\Delta_1 \leftarrow f(p_1) - f(p_2)$
8 $p_1(j) \leftarrow 0$
9 $p_2(j) \leftarrow 0$
10 $\Delta_2 \leftarrow f(p_1) - f(p_2)$
11 **if** $|\Delta_1 - \Delta_2| > \epsilon$ **then** $grp \leftarrow grp \cup j$;
12 **if** $|grp| = 1$ **then** $sepvars \leftarrow sepvars \cup grp$;
13 **else** $groups_o \leftarrow groups_o \cup \{grp\}$;

output : $groups_o \cup \{sepvars\}$

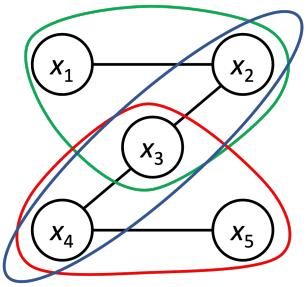

Fig. 1: Sample tree decomposition for function F20 with five variables.

Algorithm 2: Tree Based Grouping

input : tree T, number of dimensions n
initialize: $dims \leftarrow \{1, 2, \ldots, n\}$, $groups_o \leftarrow \{\}$

1 **for** $i \in dims$ **do**
2 $grp(i) \leftarrow \{i\} \cup \{T.neighbors(i)\}$
3 $groups_o \leftarrow groups_o \cup \{grp(i)\}$

output : $groups_o$

variables are compared (line 3). This is because interactions with all prior variables have already been considered, and don't need to be considered again. Note that the primary difference between ODG and DG is that DG inserts an additional line just before our line 12. Specifically, their line corresponds to

$$dims \leftarrow dims \setminus grp$$

which removes members of that group from future consideration.

2) Tree Based Grouping: Similar to ODG, tree based differential grouping (Tree) is designed to create overlapping factors. We consider each variable as a vertex in a graph, and connect them in a tree T. Factors are then constructed to be adjacent nodes in the tree. For purposes of our experiments, we use a random tree, but we note that any tree could be used. In early experiments we tested using maximal spanning trees where edge weight is a variable interaction measure, but we did not see significant benefit given the added computational expense. One of our hypotheses is that, as long as the factor architecture is connected, the origin of the underlying tree (whether random or interaction-based) is not important. The goal is to enable communication between the factors so that the interacting effects propagate through the tree structure during optimization.

The algorithm is described more formally in Algorithm 2. This algorithm iterates over each node in the tree and creates a factor consisting of the variable i and the variables connected to it in the tree $T.neighbors(i)$. This method constructs a simple tree-based architecture that has a connected factor decomposition. In a connected factor decomposition, any factor must be able to connect to any other factor through a sequence of overlapping factors. More specifically, we can envision a factor decomposition (i.e., factor architecture) to consist of a graph where each vertex in the graph corresponds to a factor, and an edge is created between two factors whenever the intersection of the variable sets in these factors is non-empty. The resulting factor decomposition is said to be connected if the resulting graph is connected. An illustration of such a tree is shown in Figure 1 We note that tree-based grouping results in a connected factor decomposition where this property does not necessarily hold for ODG.

V. Experimental Approach

The purpose of this paper is to compare the influence of the factor decomposition on large-scale optimization. Based on this, we hypothesize that the overlapping factor decomposition will provide significant benefit to optimization quality specifically on problems with a non-separable component. This is because the overlap inherently takes into account and manages variable interactions during the compete and share steps. Therefore, we hypothesize further that these interactions can be handled through sufficient iterations of the FEA algorithm to produce good results. This secondary hypothesis is motivated by prior work in a distributed setting relating the optimization process to a distributed consensus problem and showing that relaxed consensus objectives can still lead to effective performance [34].

For our experimental set-up, we follow the guidelines in [27], performing 25 trials on 1000 dimensions. We hold the number of function evaluations for most of experiments to 3×10^6.

Several different algorithms are compared: CCEA, FEA, and PSO; furthermore, we use PSO as the base algorithm for both FEA and CCEA. For CCEA decomposition, we use DG [28] and the original n 1-dimensional subproblem decomposition for PSO – CPSO-S [8]. To decompose FEA, we compare the proposed DG-extension (ODG) (section IV-1) as well as the tree-based decomposition (Tree) using a random

tree (section IV-2). In addition, we use another tree-based method, Tree2, where we merge the smallest pairs of factors from the Tree method iteratively until the total number of factors is 500. This helps determine the effect of factor decomposition and number of factors. Since we use PSO as the base algorithm for both FEA and CCEA methods, we also compare our results with single-population PSO.

For consistency, hyperparameters are held constant across functions and methods. We set 15 PSO iterations, and in earlier experiments, we did not see an influence of performance on PSO population size, so we set it to the smallest value: 10 particles per subswarm for all FEA and CCEA trials. We also set ϵ to 10^{-3} for consistency across the different methods and functions. We did not tune them individually because we sought to test the influence of the overlap.

For the canonical PSO experiments we used a population size of 1000 and used 3000 iterations in order to generate the same number of function evaluations. For PSO hyperparameters we set $c_1 = c_2 = 1.49445$ and the inertia weight $\omega = 0.729$, following work in [35]. We also decided to use the *gBest* topology.

Five CEC' 2010 benchmark functions for the session on large-scale global optimization [27] were used to test our methods, since these were used in early experiments with DG [10][28]. There are 20 functions split into five different categories: 1) separable, 2) single-group shifted and m-rotated, 3) $D/2m$ group shifted and m-rotated, 4) D/m group shifted and m-rotated, and 5) non-separable functions. D corresponds to the number of dimensions, and m determines how many variables are in a single group. These benchmark functions are all scalable, meaning the number of dimensions can be chosen. Fully separable and non-separable functions are defined as having 1-dimensional sub-vectors and having every pair of decision variables interact respectively. Li, *et al.* [36] define partially additively separable functions as having the following general form, where $\mathbf{x}_i$ are mutually exclusive decision vectors of f_i and m is the number of independent factors:

$$f(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x}_i) \ .$$

For our experiments, we ran the algorithms on one benchmark function from each of the five categories for 1000 dimensions and set $m = 50$; each function has a global minimum value of 0.

1) F3: Separable—Shifted Ackley
2) F5: Single-group m-nonseparable—Shifted m-rotated non-separable Rastrigin
3) F11: $D/2m$-group m-nonseparable—Shifted m-rotated Ackley
4) F17: D/m-group m-nonseparable—Shifted m-dimensional
5) F20: Fully nonseparable—Shifted Rosenbrock

VI. Results

Because of the generality of the FEA framework, we apply that framework to all of the architectures studied for this paper. We present metrics characterizing the different factor architectures generated for our experiments in Table I. It is interesting to note that as the non-separability of the function increases, the factor size of ODG also increases. This is because ODG detects the variable interactions and groups them together; the more variable interactions, the larger the interaction groups.

TABLE I: Summary of groupings made by each algorithm.

Function		DG	ODG	Tree	Tree2
F3	Number of Factors	1000	1000	1000	500
	Average Factor Size	1	1	2.998	5.99
F5	Number of Factors	951	1000	1000	500
	Average Factor Size	1.051525	2.225	2.998	5.992
F11	Number of Factors	513	1000	1000	500
	Average Factor Size	1.949318	13.208	2.998	5.988
F17	Number of Factors	40	1000	1000	500
	Average Factor Size	25	24.252	2.998	5.994
F20	Number of Factors	266	1000	1000	500
	Average Factor Size	3.759398	43.421	2.998	5.99

We ran experiments as outlined in Section IV. Table II shows the mean value of the objective function found during optimization, averaged over 25 trials, and the standard deviation of the returned values for each of the functions. Entries highlighted in bold are statistically significantly better than the other algorithms, tested using Wilcoxon's Rank-Sum Test at a confidence interval of 95%. Note that when multiple entries are bolded for a function, the corresponding methods were found to be statistically the same as one another. For example, for the fully separable function F3, the three methods—DG, CPSO-S, ODG—were found to be statistically equivalent but significantly better than Tree, Tree2, and PSO.

Observing the results in Table II, we see that the overlapping methods typically outperform DG on the non-separable functions. This indicates that the overlapping decomposition provides significant benefit as hypothesized. They also typically have lower standard deviations, indicating more consistent and stable performance when compared to DG. On every function with a non-separable component, FEA methods perform competitively.

We also plot the convergence curves for the first trial of each method. These results can be seen in Figure 2. Based on these results, we find a number of interesting results. First, we see rapid convergence for the three best algorithms on F3, which makes sense given the fact F3 is fully separable. Thus we would not expect overlap to provide any benefit. However, as soon as variable interaction is introduced (as shown in the remaining four plots), DG suffers. We also find that the type of factor architecture does, in fact, have an effect on performance, given the fact the "best" architecture varies across the functions studied.

Finally, we see that, on functions F17 or F20, none of the methods converge well within the limited number of function evaluations. Based on this, we ran a second set of experiments and allowed the functions to run for double the number of function evaluations (6×10^6). We ran 10 trials of each method for double the number of fitness evaluations (i.e., 6×10^6), and

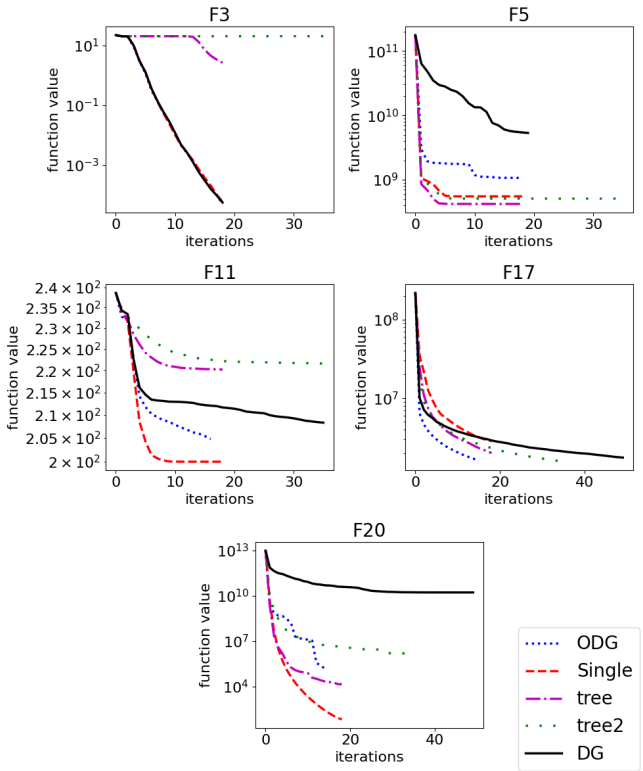

Fig. 2: Convergence plots for first trial of each method. Learning terminates when max function evaluations (3×10^6) are reached

present the results in Table III. If we also consider Figure 3, we see that most of the algorithms seem to have converged on F20, but it appears that they have not yet converged on F17, so there could be more benefit to continuing evaluation. The results improve significantly for all except DG on F17, and Single and Tree improve significantly on F20; however, the relative performance of each method did not change on F20. Specifically, if we rank the performance after 6 million fitness evaluations, we see that the rank is the same as when we stop at 3 million fitness evaluations.

We also singled out F20 to examine the effect of the factor architecture. This is because F20 comes from the fully non-separable group of functions. We note from the problem definition that each variable i interacts with $i-1$ and $i+1$. So we manually craft a factor architecture covering these interactions in order to examine the benefit of a factor architecture that matches the interactions within the problem. For trials limited to the standard 3×10^6 function evaluations, we find an average over 25 trials of 4.597E+03. This improves upon the next best score produced by FEA by over a factor of 2.

It is worth noting that CPSO-S performed by far the best, specifically on F20. In particular, CPSO-S performs over 2 orders of magnitude better than the next best algorithm (Tree). Potential reasons are discussed in Section VII.

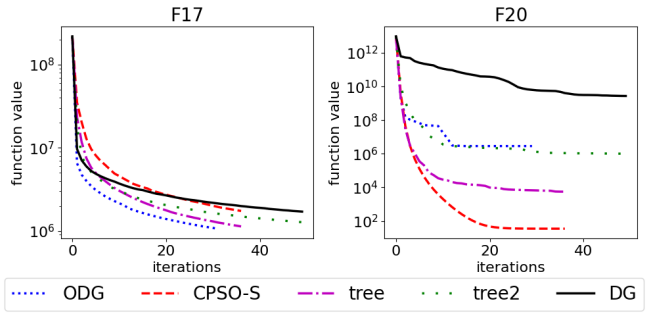

Fig. 3: Convergence plots for F17 and F20 with 6×10^6 function evaluations

VII. Discussion

Our results reaffirm that proper problem decomposition is important for large scale optimization. Canonical PSO performed the worst on every function, typically by several orders of magnitude. So any type of problem decomposition will improve optimization performance on these large-scale optimization problems. Even so, the basic decomposition performed by DG was not shown to be particularly effective on several of the functions studied.

In general, the introduction of overlapping factors in the problem decomposition helps with optimization on non-separable problems. The poor performance of the Tree algorithms on F3, which was completely separable, was expected since these methods always produce overlapping factors. Since F3 is fully separable, the overlap does not provide any benefit, and instead increases the difficultly of solving the sub-problems by increasing their dimensionality. We also want to note that ODG performed very well on this function in that it was able to recognize the fully separable nature, thus reducing to DG.

On the other hand, the benefit of the overlap is very prominent in F20. The FEA methods show improvement by many orders of magnitude over that of CCEA using DG. The performance of CPSO-S is discussed later. F20 is a fully non-separable problem indicating that overlap may be beneficial. If we look at F17, we also see a similar result where the overlapping methods perform significantly better than the non-overlapping methods, further supporting the hypothesis that overlap is beneficial to non-separable problems.

It is worth noting that, as expected, the factor architecture seems to be important at determining optimization success. This observations is very prominent in the improvement on F20. By addressing the underlying variable interactions present in the function with a manually defined factor architecture, the results were improved by several orders of magnitude.

Despite the improvements shown by choosing the correct factor architecture, it is interesting that Tree and Tree2 both produce architectures that lead to high quality results despite not addressing any underlying properties of the function. That implies that having overlapping factors is a great benefit to optimization, which can be further extended by carefully

TABLE II: Comparison of different optimization methods on CEC 2010 benchmark functions. Bold values indicate best results that were significantly better (Wilcoxon Rank-Sum p-value < 0.05)

Function		DG	CPSO-S	ODG	Tree	Tree2	PSO	Manual
F3	Mean	**5.79E-05**	**5.93E-05**	**5.92E-05**	1.37E+00	1.79E+01	2.16E+01	
	std	8.23E-06	9.08E-06	9.52E-06	4.24E-01	4.91E+00	8.43E-03	
F5	Mean	8.05E+09	**5.95E+08**	1.22E+09	**5.53E+08**	**5.56E+08**	3.63E+10	
	std	2.58E+09	1.30E+08	4.15E+08	1.12E+08	1.27E+08	2.31E+09	
F11	Mean	2.08E+02	**2.00E+02**	2.04E+02	2.20E+02	2.22E+02	2.37E+02	
	std	4.52E-01	1.07E-02	3.67E-01	1.46E+00	2.48E-01	5.86E-02	
F17	Mean	1.77E+06	2.58E+06	1.57E+06	1.75E+06	**1.37E+06**	3.29E+07	
	std	1.07E+05	1.74E+05	7.79E+04	1.42E+05	1.04E+05	1.91E+06	
F20	Mean	6.10E+09	**7.36E+01**	4.58E+06	1.18E+04	1.41E+06	6.82E+12	4.60E+03
	std	4.74E+09	2.31E+01	7.81E+06	2.62E+03	6.91E+05	1.71E+11	2.17E+03

TABLE III: Comparison of different optimization methods on F17 and F20 with double the number of function evaluations. Bold values indicate best results that were significantly better (Wilcoxon Rank-Sum p-value < 0.05)

Function		DG	CPSO-S	ODG	Tree	Tree2	Manual
F17	Mean	1.68E+06	1.56E+06	**9.82E+05**	**1.02E+06**	**1.07E+06**	
	std	4.28E+04	1.37E+05	6.42E+04	9.30E+04	1.12E+05	
F20	Mean	4.25E+09	**6.24E+01**	4.26E+06	6.21E+03	9.94E+05	4.43E+03
	std	2.83E+09	7.45E+01	8.29E+06	8.12E+02	4.35E+05	1.38E+03

choosing a factor architecture. This seems to provide a level of support for our secondary hypothesis.

We also note that using variable interaction as the basis for problem decomposition may not be beneficial in creating an appropriate factor architecture. ODG considers these interactions and is capable of determining separability, but it does not always outperform the random architectures Tree and Tree2. It is possible that ODG does not capture these interactions fully, or there may be other properties of the problem decomposition, such as connectivity, that play a larger role.

A consistent and unexpected result is the performance of CPSO-S on all functions. CPSO-S is the simplest version of CCEA with PSO, and is not expected to perform well on non-separable problems because it does not consider any form of variable interaction. However, CPSO-S is competitive across all functions. In particular, it achieved the lowest score on F11 and F20. We believe that this is due to the shape of the underlying function landscape, or a byproduct of the hyperparameter settings. The number of PSO iterations was set fairly small at 15 to balance the number of function evaluations. So CPSO-S might be better able to optimize since it only considers 1-dimensional populations, while the other algorithms potentially had more dimensions in the factors. Note that this result can also be a byproduct of the challenges associated with using fitness evaluations as the means for making results comparable, as pointed out by Engelbrecht [37]. In particular, across the various factor architectures, not all fitness evaluations are created equal.

VIII. Conclusion

In this paper we examined different decomposition methods for factored evolutionary particle swarm optimization, as compared to each other and the single-population equivalent. We extended Differential Grouping to create overlapping groups and created two new tree based decomposition approaches. The algorithms were tested using five representative functions from the set of CEC'2010 benchmark functions using the proposed guidelines [27]. Results show that overlap can be beneficial for optimization of non-separable problems, or problems with a non-separable component.

IX. Future Work

The impact of factor decomposition is still not well understood. More testing involving variable interaction within overlapping factor decompositions as well as the impact of connected factor decompositions on optimization performance is necessary. Another interesting question would be considering the impact of the amount of overlap and the size of individual factors have on optimization performance. Our results may indicate improved performance on smaller group sizes due to the success of CPSO-S.

Limited hyperparameter tuning was performed in order to maintain consistency across the experiments, so further tuning to improve performance of the individual methods would be beneficial. Due to the limited number of function evaluations, the number of parameter configurations was fairly limited. In addition, as highlighted by Engelbrecht [37], using numbers of fitness evaluations for determining terminations, and by extension for determining the basis for comparison, may be problematic and not as fair as most expect. The guidelines for the CEC 2010 studies were set before Engelbrecht's study appeared, so this process should be reconsidered. Removing the FE restriction and tuning parameters for each algorithm on each function could yield significant new insight into the relationships between factor architectures and optimization performance.

Furthermore, there are several other decomposition strategies that can be created. We would like to look at expanding some of the proposed methods as well as exploring other existing methods. Because of the increased flexibility introduced by the overlap, we can imagine many methods that better consider the function landscape and can lead to increased performance.

We propose a fairly simple extension from our Tree based decomposition by creating a maximal spanning tree from the interactions between variables. Another approach is to employ a hierarchical decomposition strategy based on local variable interactions across the function's domain.

Finally, we believe a framework can be created to formalize and unify the wide variety of problem decompositions that exist. Such a unification could result in an improved decision making process to discover the right factor architecture for a problem.

ACKNOWLEDGMENTS

We would like to thank members of the Numerical Intelligent Systems Laboratory at Montana State University for several fruitful discussions in the development of this work. We also thanks Dr. Stephyn Butcher and Dr. Shane Strasser for their insights into optimizing the FEA framework and setting up the various experiments.

REFERENCES

[1] B. Selman and C. P. Gomes, "Hill-climbing search," *Encyclopedia of Cognitive Science*, vol. 81, p. 82, 2006.

[2] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1975.

[3] R. Eberhart and J. Kennedy, "Particle swarm optimization," in *IEEE International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948.

[4] K. Meng, H. G. Wang, Z. Dong, and K. P. Wong, "Quantum-inspired particle swarm optimization for valve-point economic load dispatch," *IEEE Transactions on Power Systems*, vol. 25, no. 1, pp. 215–222, 2009.

[5] M. Pontani and B. A. Conway, "Particle swarm optimization applied to space trajectories," *Journal of Guidance, Control, and Dynamics*, vol. 33, no. 5, pp. 1429–1441, 2010.

[6] A. Peerlinck, J. Sheppard, J. Pastorino, and B. Maxwell, "Optimal design of experiments for precision agriculture using a genetic algorithm," in *IEEE Congress on Evolutionary Computation (CEC)*, 2019, pp. 1838–1845.

[7] M. A. Potter and K. A. D. Jong, "Cooperative coevolution: An architecture for evolving coadapted subcomponents," *Evolutionary Computation*, vol. 8, no. 1, pp. 1–29, 2000.

[8] F. Van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 225–239, 2004.

[9] Z. Yang, K. Tang, and X. Yao, "Large scale evolutionary optimization using cooperative coevolution," *Information Sciences*, vol. 178, no. 15, pp. 2985–2999, 2008.

[10] M. N. Omidvar, X. Li, Y. Mei, and X. Yao, "Cooperative co-evolution with differential grouping for large scale optimization," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 3, pp. 378–393, 2013.

[11] M. N. Omidvar, M. Yang, Y. Mei, X. Li, and X. Yao, "Dg2: A faster and more accurate differential grouping for large-scale black-box optimization," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 6, pp. 929–942, 2017.

[12] W. Chen, T. Weise, Z. Yang, and K. Tang, "Large-scale global optimization using cooperative coevolution with variable interaction learning," in *International Conference on Parallel Problem Solving from Nature*, 2010, pp. 300–309.

[13] Y. Sun, M. Kirley, and S. K. Halgamuge, "Quantifying variable interactions in continuous optimization problems," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 2, pp. 249–264, 2017.

[14] S. Strasser, J. Sheppard, N. Fortier, and R. Goodman, "Factored evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 2, pp. 281–293, 2017.

[15] Y. Sun, M. Kirley, and S. K. Halgamuge, "Extended differential grouping for large scale global optimization with direct and indirect variable interactions," in *ACM Genetic and Evolutionary Computation Conference (GECCO)*, 2015, pp. 313–320.

[16] R. Wang, F. Zhang, T. Zhang, and P. J. Fleming, "Cooperative coevolution with improved differential grouping method for large-scale global optimisation," *International Journal of Bio-Inspired Computation*, vol. 12, no. 4, pp. 214–225, 2018.

[17] Y. Sun, M. Kirley, and S. K. Halgamuge, "A recursive decomposition method for large scale continuous optimization," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 5, pp. 647–661, 2017.

[18] Y. Sun, M. N. Omidvar, M. Kirley, and X. Li, "Adaptive threshold parameter estimation with recursive differential grouping for problem decomposition," in *ACM Genetic and Evolutionary Computation Conference (GECCO)*, New York, NY, USA, 2018, p. 889–896.

[19] V. Gardeux, R. Chelouah, P. Siarry, and F. Glover, "Unidimensional search for solving continuous high-dimensional optimization problems," in *Ninth International Conference on Intelligent Systems Design and Applications*, 2009, pp. 1096–1101.

[20] L. Wang, J. Shen, S. Wang, and J. Deng, "Advances in co-evolutionary algorithms," *Control and Decision*, vol. 30, no. 2, pp. 193–202, 2015.

[21] C.-K. Goh and K. C. Tan, "A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 1, pp. 103–127, 2009.

[22] X. Li and X. Yao, "Tackling high dimensional nonseparable optimization problems by cooperatively coevolving particle swarms," in *IEEE Congress on Evolutionary Computation (CEC)*, 2009, pp. 1546–1553.

[23] ——, "Cooperatively coevolving particle swarms for large scale optimization," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 2, pp. 210–224, 2011.

[24] Y.-F. Zhang and H.-D. Chiang, "A novel consensus-based particle swarm optimization-assisted trust-tech methodology for large-scale global optimization," *IEEE transactions on cybernetics*, vol. 47, no. 9, pp. 2717–2729, 2016.

[25] A. Gupta, Y.-S. Ong, and L. Feng, "Multifactorial evolution: toward evolutionary multitasking," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 3, pp. 343–357, 2015.

[26] W. Chen and K. Tang, "Impact of problem decomposition on cooperative coevolution," in *IEEE Congress on Evolutionary Computation (CEC)*, 2013, pp. 733–740.

[27] K. Tang, X. Li, P. N. Suganthan, Z. Yang, and T. Weise, "Benchmark functions for the CEC'2010 special session and competition on large-scale global optimization," Nature Inspired Computation and Applications Laboratory, Tech. Rep., 2009.

[28] M. N. Omidvar, X. Li, and X. Yao, "Cooperative co-evolution with delta grouping for large scale non-separable function optimization," in *IEEE Congress on Evolutionary Computation (CEC)*, 2010, pp. 1762–1769.

[29] W. Liu, Y. Zhou, B. Li, and K. Tang, "Cooperative co-evolution with soft grouping for large scale global optimization," in *IEEE Congress on Evolutionary Computation (CEC)*, 2019, pp. 318–325.

[30] X. Ma, X. Li, Q. Zhang, K. Tang, Z. Liang, W. Xie, and Z. Zhu, "A survey on cooperative co-evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 3, pp. 421–441, 2018.

[31] E. Sayed, D. Essam, and R. Sarker, "Dependency identification technique for large scale optimization problems," in *IEEE Congress on Evolutionary Computation (CEC)*, 2012, pp. 1–8.

[32] J. D. Schaffer and A. Morishima, "An adaptive crossover distribution mechanism for genetic algorithms," in *Second International Conference on Genetic Algorithms*, 1987, pp. 36–40.

[33] M. Yang, A. Zhou, C. Li, and X. Yao, "An efficient recursive differential grouping for large-scale continuous problems," *IEEE Transactions on Evolutionary Computation*, 2020.

[34] S. G. W. Butcher, S. Strasser, J. Hoole, B. Demeo, and J. W. Sheppard, "Relaxing consensus in distributed factored evolutionary algorithms," in *ACM Genetic and Evolutionary Computation Conference (GECCO)*, July 2016, pp. 5–12.

[35] S.-Z. Zhao, J. J. Liang, P. N. Suganthan, and M. F. Tasgetiren, "Dynamic multi-swarm particle swarm optimizer with local search for large scale global optimization," in *IEEE Congress on Evolutionary Computation (CEC)*, 2008, pp. 3845–3852.

[36] X. Li, K. Tang, M. N. Omidvar, Z. Yang, K. Qin, and H. China, "Benchmark functions for the CEC 2013 special session and competition on large-scale global optimization," vol. 7, no. 33, p. 8, 2013.

[37] A. P. Engelbrecht, "Fitness function evaluations: A fair stopping condition?" in *IEEE Swarm Intelligence Symposium*, 2014.