Improved Constructive Cooperative Coevolutionary Differential Evolution for Large-Scale Optimisation

Emile Glorieux, Bo Svensson, Fredrik Danielsson
Department of Engineering Science
University West
S-461 86 Trollhättan, Sweden
Email: emile.glorieux@hv.se

Bengt Lennartson
Department of Signals and Systems
Chalmers University of Technology
S-412 96 Gothenburg, Sweden

***Abstract*—The Differential Evolution (DE) algorithm is widely used for real-world global optimisation problems in many different domains. To improve DE's performance on large-scale optimisation problems, it has been combined with the Cooperative Coevolution (CCDE) algorithm. CCDE adopts a divide-and-conquer strategy to optimise smaller subcomponents separately instead of tackling the large-scale problem at once. DE then evolves a separate subpopulation for each subcomponent but there is cooperation between the subpopulations to co-adapt the individuals of the subpopulations with each other. The Constructive Cooperative Coevolution (C^3DE) algorithm, previously proposed by the authors, is an extended version of CCDE that has a better performance on large-scale problems, interestingly also on non-separable problems. This paper proposes a new version, called the Improved Constructive Cooperative Coevolutionary Differential Evolution ($C^{3i}DE$), which removes several limitations with the previous version. A novel element of $C^{3i}DE$ is the advanced initialisation of the subpopulations. $C^{3i}DE$ initially optimises the subpopulations in a partially co-adaptive fashion. During the initial optimisation of a subpopulation, only a subset of the other subcomponents is considered for the co-adaptation. This subset increases stepwise until all subcomponents are considered. The experimental evaluation of $C^{3i}DE$ on 36 high-dimensional benchmark functions (up to 1000 dimensions) shows an improved solution quality on large-scale global optimisation problems compared to CCDE and DE. The greediness of the co-adaptation with $C^{3i}DE$ is also investigated in this paper.**

I. Introduction

Population-based algorithms are well-suited and popular for solving real-world global optimisation problems. More and more optimisation problems arise in engineering and other fields. Many real-world problems are large-scale in terms of number of parameters to optimise. The requirements for algorithms to solve these problems are increasing, in terms of solution quality, complexity, optimisation time reduction, usability, etc. There is a continuous demand in the field of computational intelligence for more advanced optimisation algorithms to solve large-scale real-world problems within a practical time-frame.

Differential Evolution (DE) is an efficient, simple, robust and powerful population-based evolutionary algorithm, especially for continuous optimisation problems [1]. DE became popular among computer scientists and practitioners soon after its original definition by Storn and Price [2]. A main difference with DE compared to other evolutionary algorithms is that it perturbs the individuals of the population with scaled differences of other randomly selected and distinct individuals. DE is thereby *self-organising*, which reduces the number of control parameters that must be set by the user [2].

DE and many DE-variants typically have an excellent performance for problems that have between 30 to 100 dimensions, but the performance rapidly decreases when it increases beyond 500 dimensions [3], i.e. large-scale problems. The number of dimensions is a major factor that contributes to the complexity of an optimisation problem and the difficulty to solve it. For such large scale problems, one of the approaches is to extend DE with the Cooperative Coevolution (CC) algorithm [4]–[13]. With CCDE, the optimisation problem is divided into smaller subcomponents that are easier to solve. Each subcomponent is assigned a separate subpopulation. These subpopulations are then evolved separately, by DE, while cooperating with each other to co-adapt the individuals. This improves the performance of the optimisation of large-scale problems. Though, there are difficulties with CCDE, specifically to decompose and optimise non-separable problems. For separable problems, the optimum can be found by performing a line search in each search direction (i.e. for each parameter). Whereas this is impossible for non-separable problems because there are interdependencies/interactions between parameters.

An extended version of CCDE was previously proposed by the authors, called Constructive Cooperative Coevolutionary Differential Evolution (C^3DE) [14], [15]. The extension of C^3DE includes deploying a constructive heuristic to create good initial feasible solution for the complete problem. This initial solution is then used for the cooperation between the subpopulations at the start of the optimisation with CCDE. When the search with C^3DE stagnates, it restarts to create a new different initial feasible solution. The results show that C^3DE outperforms CCDE on real-world problems concerning interacting production stations [14]. Though, it was found that there are several drawbacks and limitations with this version of C^3DE which make it relatively complex to implement and use.

The objective of this paper is to propose and investigate an improved version of the previously proposed C^3DE for large-scale optimisation. The improvements are concerned with a more effective mechanism to co-adapt the subpopulations. A second objective of this paper is to investigate how to control the greediness of the co-adaptation.

978-1-4799-7560-0/15 $31.00 © 2015 IEEE
DOI 10.1109/SSCI.2015.239

The main contribution of this paper is the proposed Improved Constructive Cooperative Coevolution Differential Evolution ($C^{3i}DE$) algorithm for large-scale optimisation. A novel element with $C^{3i}DE$ is that a subpopulation is co-adapted with an elite fraction of the other subpopulations, already during its initialisation. Furthermore, the size of the elite fraction affects the greediness of $C^{3i}DE$ and can be controlled. During the initialisation, the subpopulation is immediately co-adapted with the elite fraction of the previously initialised subpopulations, and are first initially optimised in this way to obtain an optimised elite fraction for the initialisation of the next subpopulations. Compared to the previous version, the improvements of $C^{3i}DE$ are a controllable greediness, fully scalable in number of subcomponents, fewer user-settings to tune, simpler to implement, and applicable to a wider range of problems. These improvements increase the performance and make $C^{3i}DE$ easier to use. A second contribution is the investigation of several method to control the greediness of $C^{3i}DE$ that indicates that 100% greedy or a dynamically controlled greediness are preferable.

II. Cooperative Coevolutionary Differential Evolution

Cooperative Coevolution (CCDE) is a divide-and-conquer strategy to optimise large-scale problems. The pseudo-code of the CCDE algorithm is shown in Algorithm 1. The problem is decomposed into smaller, easier-to-solve, subcomponents that can be optimised separately, as shown on line 1 in Algorithm 1. This is done by grouping the parameters into n sets $\{G_1, \ldots, G_n\}$. After randomly initialising the subpopulations of each subcomponent (line 2 in Algorithm 1), an evolutionary algorithm is typically adopted to evolve the subpopulations of the subcomponents. This is done in a round-robin fashion for the subcomponents as shown on lines 4-10 in Algorithm 1. To calculate the fitness of an individual, the subpopulations cooperate by exchanging individuals. These individuals are called *collaborators* [16]. This exchange is necessary because an individual of a subpopulation represents only a subvector of the solution vector for the problem. To evaluate the fitness of an individual, it is combined with the collaborators, one from each other subcomponent, to form a solution vector for the complete problem (lines 7-8 in Algorithm 1). The fitness is assigned to the individual under evaluation and also to the collaborators (line 9 in Algorithm 1). The fitness assignment to the collaborators can be done in different ways, as proposed by Wiegand et al. [17], i.e. *optimistic*, *pessimistic* or *hedge*. In the *optimistic* case, an individual is assigned/keeps the best fitness of its own evaluation and all collaborations in which it takes part. The *pessimistic* case is similar, an individual is assigned/keeps the worst fitness of its own evaluation and all collaborations in which it takes part. In the *hedge* case, an individual is assigned the average fitness of its own evaluation and all collaborations in which it takes part. Next, the best individual, either a parent or its corresponding child is selected for the new subpopulation (line 10 in Algorithm 2).

Several researchers adopt and investigate improvements for CCDE to further improve the performance on large-scale problems [4]–[13]. More advanced version of CCDE focus either on the self-adaptation of the DE's control parameters or on automatic and adaptive grouping strategies to decompose the problem [18].

```
1:  G = {G_1, ..., G_n} ← grouping(n)
2:  pop ← initialise(pop_1, ..., pop_n)
3:  while termination criteria false do
4:      for each subcomponent i do
5:          select parents from pop_i
6:          produce children from selected parents
7:          select collaborators from (pop − pop_i)
8:          evaluate children interacting with collaborators
9:          assign fitness to children and collaborators
10:         select survivors for new pop_i
11:     end for
12: end while
13: return best solution
```

Algorithm 1: Pseudo code of the (CCDE) algorithm

Zamuda et al. [4] and Weber et al. [5] proposed self-adaptation for CCDE. Each individual of the subpopulations directly encodes two control parameters of DE, i.e. the mutation scale factor (F) and the crossover parameter (CR). A "log-normal" self-adaptation mechanism is adopted to tune these control parameters during the optimisation. The self-adaptation allows the control parameters to self-adapt for each specific subcomponent without any user interaction.

For non-separable problems, it was found that it is crucial to group interacting parameters together in the same subcomponent, so they are optimised together. Depending on the problem, this might not be straightforward or feasible, especially as often these interactions are not known explicitly. Different ways for grouping the parameters to decompose the problem have been proposed to handle such non-separable problems, for example random grouping [7], [8], correlation-based grouping [9], variance-based CCDE [10], multi-level CCDE [11], [12] and differential grouping [13]. A more detailed overview of grouping strategy is given by Trunfio [18]. These strategies are particularly interesting when nothing or very little is known about the interactions between the parameters. For some problems, an appropriate decomposition might be explicitly known and can be hand-decomposed in advance. In this work, the grouping strategy is not studied but considered as future work. Thereby, in the tests performed in this work, the problems are decomposed randomly in the equal-sized subcomponents.

Furthermore, the selection of the collaborators from the subpopulations is an important aspect of CCDE that can influence the performance [17]. For some problems, it is sufficient to chose the best individual in the subpopulation as the collaborator, but for other problems this is too greedy [19]. As investigated by Wiegand et al. [17], there are three main aspects to alternatives, i.e. collaborator selection pressure, number of collaborators, credit assignment when using multiple collaborators. It was found that the greedy alternative is preferable for straight-forward fully separable problems. For complex non-separable problems, the higher number of collaborators, the better the performance. Although, this comes with an additional computational cost. Hence, that is not always applicable. The influence of the selection pressure seemed to be of lesser importance. In this study of the collaborators selection by Wiegand et al. [17], only three different benchmark functions with low number of parameters

were considered. Therefore, in the work presented in this paper, the collaborators selection is further investigated for $C^{3i}DE$ and with wider range of larger problems.

III. Original Constructive Cooperative Coevolutionary Differential Evolution

The Constructive Cooperative Coevolutionary (C^3DE) algorithm is an extended version of CCDE and was proposed by the authors for optimising the control of interacting production stations using simulation-based optimisation [20], [21]. It was tested on real-world computationally-expensive large-scale problems and compared with other algorithms such as CCDE, DE, CCPSO and Particle Swarm Optimiser. It was found that it outperforms the other algorithms on these problems.

The C^3DE algorithm works as following, it starts with step-wise constructing a feasible solution, by initialising and optimising one subcomponent's subpopulation at the time. The best individual of the previously initialised subpopulations is used as collaborator during the initial optimisations. For the fitness calculations, the problem's dimensions is thus reduced to only the dimensions of the individual and the collaborators. For example, when initially optimising the second subpopulation, the fitness is calculated on a problem reduced in dimensions corresponding to the dimensions of the first and second subcomponent.

After all subpopulations have been initialised, C^3DE continues optimising in the same way as CCDE. Then, C^3DE uses a fixed set of collaborators (one for each subcomponent) for all function evaluations. At certain intervals, this set of collaborators is updated by trying to combine the new single best individuals in each subpopulation. This is done by an exhaustive search of all possible enumerations of the old collaborators and new best individuals in the subpopulations.

When the search stagnates, C^3DE restarts the constructive method by stepwise reinitialising subpopulations and optimising the subcomponents. The construction restarts from the best individual from one of the subpopulations after initialisation that was not used as collaborator during the previous construction. To be able to select the best individual, it must be possible to compare their fitness value although the different individuals consider a different number of subcomponents. For this to be possible, it is required that all subcomponents are similar.

There are several drawbacks with this version of C^3DE. First, it is very greedy because the single best individual from each subpopulation is always used as collaborator. It also was found that it is not scalable in number of subcomponents due to this exhaustive search for updating the set of collaborators. Furthermore, there are several setting parameters that need to be tuned, such as the interval for updating the set of collaborators, how to decide that the search stagnated and how to restart. The updating of the set of collaborators and the restarting when the search stagnates make that it is not necessary straightforward to implement C^3DE. C^3DE also requires that all subcomponents are similar in nature, as that they represent similar substructures of the problem.

IV. Improved Constructive Cooperative Coevolutionary Differential Evolution

The Improved Constructive Cooperative Coevolutionary Differential Evolution ($C^{3i}DE$) algorithm is proposed in this paper. $C^{3i}DE$ avoids having the drawbacks and limitations of the earlier version (C^3DE) which were discussed in the previous section. The pseudo-code of $C^{3i}DE$ is shown in Algorithm 2. First, the optimisation problem is decomposed into n subcomponents based on the partitioning of the D parameters (or dimensions) $G = \{1, 2, \ldots, D\}$ into n sets $G_1, \ldots, G_n$ (line 1 in Algorithm 2). Each set of parameters defines the search space of the corresponding subcomponent. The number of dimensions for a subcomponent's search space is thus significantly less than total number of dimensions D. When a subcomponent is optimised, DE is applied on the subcomponent's search space. Each subcomponent is assigned a separate subpopulation $S^{(i)}$. By construction, an individual in $S^{(i)}$ only contains the parameter set of the corresponding subcomponent.

One requirement remains with $C^{3i}DE$ which is that a subpopulation's individual's fitness can be calculated combined with no collaborators or with fewer collaborators (less than one for each subcomponent). The fitness of an individual is then calculated by applying it to only the corresponding subcomponent, and without considering all other subcomponents. The individual can also be combined with collaborators from the other subpopulations but not necessarily all, then only the subcomponents of these collaborators are considered for calculating the fitness. This is thus equivalent to having the same problem with lower number of dimensions.

```
1:  G = {G_1, ..., G_n} ← grouping(n)
2:  pop ← {∅}
3:  for each subcomponent i do
4:      pop ←initialise(pop_i)
5:      while termination criteria false do
6:          select parents from pop_i
7:          produce children from selected parents
8:          select collaborators from (pop_1, ..., pop_{i-1})
9:          evaluate children interacting with collaborators
10:         assign fitness to children and collaborators
11:         select survivors for new pop_i
12:     end while
13: end for
14: while termination criteria false do
15:     for each subcomponent i do
16:         select parents from pop_i
17:         produce children from selected parents
18:         select collaborators from (pop − pop_i)
19:         evaluate children interacting with collaborators
20:         assign fitness to children and collaborators
21:         select survivors for new pop_i
22:     end for
23: end while
24: return best solution
```

Algorithm 2: Pseudo code of the $C^{3i}DE$ algorithm

A. Description $C^{3i}DE$ algorithm

$C^{3i}DE$ starts with stepwise initialising and optimising the subpopulations in a constructive fashion (line 3-13 in Algorithm 2). First, a subcomponent's subpopulation is initialised randomly (line 4 in Algorithm 2). This subpopulation is then optimised by DE as following. First, DE selects the parent individuals from the subpopulation and children are produced from these (lines 6-7 in Algorithm 1). Collaborators are selected from the previously initialised subcomponents (line 8 in Algorithm 2). Only these subcomponents will be considered for the fitness calculation and the others are neglected. The children are evaluated while cooperating with the selected collaborators (line 9 in Algorithm 2). The calculated fitness is assigned to the individual under evaluation and to the collaborators. The fitness assignment to the collaborators is done in an *optimistic* way [17], i.e. an individual is assigned the best fitness of its own evaluation and the collaborations in which it takes part. A collaborator's fitness value that was calculated with less subcomponents is always replaced by the new fitness calculated with more subcomponents. Next, the best individual, either a parent or its corresponding child is selected for the new subpopulation (line 10 in Algorithm 2). A single subpopulation is optimised until the predefined termination criterion is reached (line 5 in Algorithm 2) and then the subpopulation of the next subcomponent is initialised and optimised in the same way.

Once this is done for all subcomponents, the optimisation continues in the same fashion as CCDE. Each subpopulation pop_i is optimised separately (lines 14-24 in Algorithm 2). This is done in a *round-robin* fashion for the subpopulation of the subcomponents, performing one generation for each subpopulation. Now, in contrast with before, collaborators from all other subpopulations are used for the individuals' fitness calculations. This is continued until the termination criteria for the optimisation are met. The best individual that includes all subcomponents is then presented as the found optimal solution for the problem.

The generation of the initial population of DE is an important issue. Although random initialisation is typically used for DE, for certain optimisation problems and/or hybrid optimisation algorithms that use DE, it is likely that there exist other population initialisation strategies that lead to much better results [22]. $C^{3i}DE$ inherits the stepwise initialisation and optimisation of the subpopulations from the previous version. Instead of initialising all subpopulations randomly at once, each subpopulation is initially optimised taking into account the subcomponents that were previously initialised. The individuals of the subpopulations are thereby already co-adapted to the other subcomponents during the initialisation.

Compared to the previous version, the improvements with $C^{3i}DE$ relate to using different randomly selected collaborators from an elite subset of each subpopulation, both during the initial optimisations and when all subpopulations are initialised. This allows to control the greediness, whereas in the previous version, the search was always 100% greedy because only the best individual of a subpopulation is used as collaborator. This makes the initialised individuals of the subpopulations more robust in the sense that they are co-adapted to different collaborators instead of just one. Furthermore, it avoids having to perform the exhaustive search of all possible enumerations of the old and new collaborators to update the set of collaborators used once all subpopulations are initialised. This exhaustive search puts a limit on the number of subcomponents the previous version could handle. With the improved $C^{3i}DE$, there is no exhaustive search to update the collaborators and thus no limitations on the number of subcomponents.

In total, there are four user-parameters with $C^{3i}DE$, next to the usual user-parameter(s) for the termination-criteria of an evolutionary algorithms. A first user-parameter of $C^{3i}DE$ is the number of subcomponents (n). It is assumed that equal-sized $(\frac{D}{n})$ subcomponents are used as proposed in this work, otherwise the size of each subcomponent must also be defined. A second user-parameter of $C^{3i}DE$ is the size of the subpopulations (NP), or in other words the number of individuals in the subpopulations. Another user-parameter is the termination criteria for the initial optimisation of a subpopulation. This will set the ratio between how much of the optimisation effort is spent on the initial optimisations, and how much when all subpopulations are initialised. In this work, a ratio of 1/1 was adopted so half of the total optimisation effort is spent on the initial optimisations. A final user-parameter is the greediness of the collaborator selection (k). This decides which individuals of a subpopulation are chosen as collaborators. Different ways to set k will be proposed and discussed in the next subsection.

B. Greediness collaborator selection

To control the greediness of the collaborator selection, collaborators are selected from an elite set of individuals of each subpopulation, i.e. a fraction of a subpopulation with the k best individuals. As with CCDE, the greediness influences the performance as discussed by Wiegand et al. [17]. When $k = 1$, the selection has maximum greediness and only the best individual in a subpopulation is chosen as collaborator. When k is equal to the size of the subpopulations $k = NP$, then there is no greediness and a random individual is chosen from a subpopulation as collaborator. Another possibility is to use a specific value for k (i.e. $1 < k < NP$). Then, a collaborator is randomly selected from the k best individuals in a subpopulation. Different values can be used during the initialisation optimisations of the subpopulations (line 8 in Algorithm 2) and then another k value is used when all subpopulations are initialised, (line 18 in Algorithm 2).

It might be valuable to consider dynamically adjusting k, to increase $C^{3i}DE$'s greediness as the search progresses. When dynamically adjusting k during the search, it starts from the subpopulation size $k = NP$. This means that a random individual from each subpopulation is chosen. During the search, k is gradually decreased so that by then end of the optimisation it is equal to one $k = 1$. The collaborator selection is initially fully random and as the optimisation progresses, the greediness increases until it finally becomes 100% greedy at the end of the optimisation. The search space is widely explored at the beginning of the optimisation and by the end the optimisation searches around the best solutions found so far.

Another possibility is to let DE self-adaptively tune the value for k for each subpopulation. In this case, there is a specific k value for each subcomponent. The self-adaptive

TABLE I. THE BENCHMARK FUNCTIONS' SPECIFICATIONS [24], [25]

	Separable	Modality	Domain
f_{Ackley}	Yes	Multi	$[-35; 35]$
f_{Eliptic}	Yes	Uni	$[-100; 100]$
$f_{\text{Rastrigin}}$	Yes	Multi	$[-5.12; 5.12]$
f_{Sphere}	Yes	Uni	$[-10; 10]$
$f_{\text{SumOfSquares}}$	Yes	Uni	$[-10; 10]$
$f_{\text{W/Wavy}}$	Yes	Multi	$[-\pi; \pi]$
$f_{\text{Dixon\&Price}}$	No	Uni	$[-10; 10]$
$f_{\text{Rot.Ackley}}$	No	Uni	$[-35; 35]$
$f_{\text{Rot.Rastrigin}}$	No	Uni	$[-5.12; 5.12]$
$f_{\text{Rosenbrock}}$	No	Uni	$[-10; 10]$
f_{Schwefel}	No	Uni	$[-10; 10]$
f_{Griewank}	No	Multi	$[-5; 5]$

mechanism used is entirely similar as for the control parameter in the self-adaptive DE (jDErpo) algorithm proposed by Brest et al. [23]. Each individual of a subpopulation has its own k value. The better values for k lead to better individuals, which propagate their k values into the new populations. The individuals with a bad k value are replaced by other individuals.

Different values for k, dynamically adjusting k and self-adaptively tuning k have been investigated in this work and are discussed in the numerical section of this paper.

V. IMPLEMENTATION

The jDErpo algorithm proposed by Brest et al. [23] is used in combination with C^{3i}DE. This is a self-adaptive version that uses two different mutation strategies (rand/1 and *p*Best/1). It also uses a mechanism to adapt the control parameters, i.e. the mutation scale factor (F) and the crossover parameter (CR). During the optimisation, the control parameters are also gradually increased, by raising the lower boundaries for the adaptation. It is shown that these improvements result in a better performance of jDErpo compared to other version of DE [23].

The performance of C^{3i}jDErpo, with different k values, is evaluated and compared with CCjDErpo and standalone jDErpo for a set of large-scale benchmark functions. For clarity, in C^{3i}jDErpo and CCjDErpo, the subcomponents are optimised using jDErpo. The tests with different k values allows us to investigate the best k-value for different types of problems. In total, 36 tests are done for each algorithm in the comparison, i.e. for 12 different functions and each again for 3 different number of parameters ($D = 100, D = 500, D = 1000$). The 12 functions are listed in Table I and a detailed description is given by Jamil and Yang [24] and Omidvar et al. [25]. The functions are decomposed randomly into 25 equally-sized different subcomponents ($n = 25$).

For all tests with C^{3i}jDErpo and CCjDErpo, jDErpo is used to optimise the subcomponents. The size of the subpopulations was set to $NP = 50$. The population size for the stand-alone DE was set to $NP = 100$. The stand-alone DE has a larger population size to be able to handle all D parameters at once, where as the subpopulations is to optimise only the $\frac{D}{n}$ parameters of the corresponding subproblem. The termination criterion of the optimisation was $3 \cdot 10^6$ fitness calculations. For C^{3i}jDErpo, the termination criterion for the initialisation optimisation of a subcomponent was set to 60,000 fitness calculations. This results in that half of the optimisation effort (i.e. $1.5 \cdot 10^6$ fitness calculations) is spent on the initialisation optimisations of the subcomponents. All tests are repeated 25 times to obtain reliable mean results. All repetitions are performed independently, with random start values.

In total five different k-values are tested in this work. First, two different k-values were used, k_1 for during the initial optimisations, and another value k_2 when all subpopulations have been initialised. In Table II, this is referred to as $k = (k_1, k_2)$. In these tests, k_1 was set to 5 (best 10% of the subpopulation) and k_2 was set to 25 (best 50% of the subpopulation). This was based on the intuition that the search must be greedy during the initial optimisations to quickly find good solutions. When all subpopulations are initialised, the search must be less greedy, to explore the regions around these good solutions from the initial optimisations. For the second group of tests, the k-value was set to the size of the subpopulations. In Table II, this is referred to as $k = NP$. This means that the greediness is 0% and any individual from a subpopulation can be selected as collaborator. In the third group of tests, the k-value was set to 1. This is referred to as $k = 1$ in Table II. This means that the greediness is 100% and that the best individual in the subpopulation is always chosen as collaborator. In the fourth group of tests, k-value is dynamic, which means that in the beginning of the optimisation it is set to subpopulation size (i.e. zero greediness) and it is then decreased incrementally as the optimisation progresses, and by the end of the optimisation it has become 1 (i.e. 100% greedy). This is referred to as "$dynamic\, k$" in Table II. For the fifth group of tests, the k-value is self-adaptive. This is referred to as "$adaptive\, k$" in Table II.

VI. RESULTS AND DISCUSSION

The results of all tests done in this work are presented in Table II. The values shown in this table are the fitness of the best solution found during the optimisation (i.e. after $3 \cdot 10^6$ function evaluations) and are the mean of the 25 repetitions. The significantly best results, using the Kruskal-Wallis nonparametric one-way ANOVA test with Bonferroni correction and a significance level of 0.05, are highlighted in bold.

First, the comparison of the different k-values for C^{3i}jDErpo is discussed. The results show that for 22 of the 36 tests, there was no difference between the different options for k. The value $k = (k_1, k_2)$ performs worse than the other k-values in 6 tests, $k = NP$ in 13 tests and $adaptive\, k$ in 1 test. In general, it can be said that $k = 1$ and $dynamic\ k$ are the preferred options for k. In the authors' opinion, it is rather remarkable that $k = 1$ generates good results even though it is 100% greedy.

The k-value becomes more important with non-separable functions. In 3 of the 18 tests on separable functions there are differences between the different k-values. Whereas, this is the case in 11 of the 18 tests on the non-separable functions.

Next, the results of C^{3i}jDErpo (with $k = 1$) are compared the results of CCjDErpo. C^{3i}jDErpo has a better performance than CCjDErpo in 19 of the 36 tests. In all other tests, there is no difference between the performance of C^{3i}jDErpo and

TABLE II. RESULTS INVESTIGATION k VALUE FOR C^{3i}DE AND COMPARISON WITH CCDE AND DE (WHEN THERE IS A SIGNIFICANT DIFFERENCE, THE BEST RESULTS ARE HIGHLIGHTED IN BOLD)

		C^{3i}DE					CCDE	DE
	D	(k_1, k_2)	NP	1	*dynamic*	*adaptive*		
f_{Ackley}	100	**6.5E-19**	**6.5E-19**	**6.5E-19**	**6.5E-19**	**6.5E-19**	**1.4E-15**	1.7E-1
	500	**1.1E-15**	**1.1E-15**	**1.2E-15**	**1.1E-15**	**1.1E-15**	2.9E+0	5.4E+0
	1000	**3.0E-11**	**4.5E-3**	**7.8E-10**	**6.9E-10**	**7.7E-14**	6.6E+0	8.4E+0
f_{Elliptic}	100	**0.0E+0**	**0.0E+0**	**0.0E+0**	**0.0E+0**	**0.0E+0**	**0.0E+0**	1.8E-22
	500	**2.8E-23**	**9.3E-24**	**2.1E-23**	**1.1E-23**	**1.3E-23**	3.4E-23	4.5E-3
	1000	**3.4E-14**	4.9E-2	**3.6E-20**	**2.7E-20**	**2.4E-20**	**6.2E-19**	3.9E+7
$f_{\text{Rastrigin}}$	100	**0.0E+0**	**0.0E+0**	**0.0E+0**	**0.0E+0**	**0.0E+0**	**0.0E+0**	5.4E+1
	500	2.6E+1	2.5E+1	2.6E+1	2.5E+1	2.6E+1	8.6E+1	6.1E+2
	1000	9.8E+2	1.1E+3	**6.8E+2**	**6.8E+2**	**5.3E+2**	**5.2E+2**	1.4E+3
f_{Sphere}	100	**0.0E+0**	**0.0E+0**	**0.0E+0**	**0.0E+0**	**0.0E+0**	**0.0E+0**	4.1E-29
	500	5.2E-30	3.7E-30	4.7E-30	6.0E-30	4.0E-30	1.0E-29	7.2E-9
	1000	**1.0E-20**	**3.3E-16**	**5.9E-26**	**6.7E-26**	**6.4E-26**	**1.3E-25**	8.9E+1
$f_{\text{SumOfSquares}}$	100	**0.0E+0**	**0.0E+0**	**0.0E+0**	**0.0E+0**	**0.0E+0**	**0.0E+0**	2.1E-27
	500	2.3E-27	3.0E-27	1.2E-27	1.9E-27	1.6E-27	4.5E-27	6.6E-6
	1000	**8.5E-18**	7.2E-14	**4.0E-23**	**3.2E-23**	**4.4E-23**	**1.4E-22**	3.1E+4
$f_{\text{W/Wavy}}$	100	**0.0E+0**	**0.0E+0**	**0.0E+0**	**0.0E+0**	**0.0E+0**	**5.1E-17**	6.3E-2
	500	**3.9E-3**	**3.5E-3**	**3.6E-3**	**3.7E-3**	**3.6E-3**	2.0E-2	2.2E-1
	1000	**2.8E-2**	**2.8E-2**	**2.9E-2**	**2.8E-2**	**2.9E-2**	6.5E-2	2.8E-1
$f_{\text{DixonPrice}}$	100	3.6E+1	4.7E+1	**3.1E+1**	**3.1E+1**	**3.1E+1**	4.8E+3	**6.7E-1**
	500	2.6E+2	8.0E+2	**1.0E+2**	**1.1E+2**	**1.2E+2**	4.6E+2	6.9E+2
	1000	1.8E+3	4.3E+3	**1.4E+3**	**1.5E+3**	**1.3E+3**	2.3E+3	2.6E+6
$f_{\text{Rot.Ackley}}$	100	7.3E-19	**0.0E+0**	**0.0E+0**	**1.7E-20**	**0.0E+0**	3.3E+3	4.3E+3
	500	3.8E-17	8.4E-17	**1.0E-17**	**1.1E-17**	1.4E-17	1.6E+4	1.7E+4
	1000	**4.2E-14**	7.1E-12	**3.3E-16**	**3.6E-16**	**3.4E-16**	3.4E+4	3.7E+4
$f_{\text{Rot.Rastrigin}}$	100	**1.1E-14**	**1.1E-14**	**1.1E-14**	**1.1E-14**	**1.1E-14**	2.1E+1	2.0E+1
	500	**1.2E-14**	**1.2E-14**	**1.2E-14**	**1.2E-14**	**1.2E-14**	2.2E+1	2.1E+1
	1000	**2.7E-9**	1.3E-8	**1.2E-11**	**1.3E-11**	**1.1E-11**	2.2E+1	2.1E+1
$f_{\text{Rosenbrock}}$	100	**1.9E+4**	**2.8E-2**	**8.8E-9**	**1.9E+4**	**8.8E-9**	1.3E+2	2.1E+1
	500	**5.4E+2**	4.7E+3	**5.3E+2**	**5.3E+2**	**5.3E+2**	9.6E+2	1.2E+3
	1000	**1.7E+3**	5.5E+5	**1.6E+3**	**1.7E+3**	**1.6E+3**	**2.1E+3**	1.2E+5
f_{Schwefel}	100	**2.0E-31**	**1.5E-31**	**2.5E-30**	**3.2E-31**	**0.0E+0**	8.2E+3	**2.3E-7**
	500	**2.7E-3**	9.4E+0	**1.4E-4**	**1.5E-4**	**1.9E-4**	1.4E+5	**1.5E+2**
	1000	**3.0E+2**	2.7E+4	**5.7E+2**	**2.0E+2**	**7.0E+2**	2.8E+5	**7.9E+5**
f_{Griewank}	100	3.0E-4	1.5E-4	3.0E-4	4.4E-4	4.4E-4	4.0E-19	3.0E-4
	500	4.4E-4	1.5E-4	4.4E-4	4.4E-4	3.0E-4	5.7E-18	2.6E-10
	1000	**1.4E-18**	**1.5E-4**	**3.0E-4**	**3.0E-4**	**1.3E-18**	**4.8E-17**	5.5E-2

CCjDErpo. In general, it can be said that C^{3i}jDErpo has a better or similar performance compared to CCjDErpo.

In the tests on non-separable functions, there is a significant difference in 14 of the 18 tests. This is a clear indication that C^{3i}jDErpo outperforms CCjDErpo on non-separable problems. This is interesting because it is known that cooperative coevolutionary optimisation algorithms struggle with non-separable problems.

C^{3i}jDErpo and CCjDErpo perform similar in most the tests on the separable functions (13 of 18). In the other 5 tests, there is a significant difference is favour of C^{3i}jDErpo. Concluding that C^{3i}jDErpo performs as good as CCjDErpo and sometimes even better on separable problems.

When comparing C^{3i}jDErpo with jDErpo, the results show that C^{3i}jDErpo has a better performance for the majority of the benchmark functions (i.e. 27 of 36) and in the 9 other, the performance is similar. This indicates that C^{3i}jDErpo performs superior on large-scale problems

It should also be mentioned that for 8 of the 18 tests on the non-separable functions, CCjDErpo has a worse performance than jDErpo. Specifically, this is the case for the tests on $f_{\text{DixonPrice}}$ ($D = 100$), $f_{\text{Rot.Rastrigin}}$ ($D = 100, 500, 1000$), $f_{\text{Rosenbrock}}$ ($D = 100$), and f_{Schwefel} ($D = 100, 500, 1000$). This confirms the conclusion from other researchers that CCDE has difficulties to solve non-separable problems [11], [13], [26]. Note that this cannot be seen in Table II for all those 8 tests because only the significantly best results are highlighted in bold.

The computational cost of the different algorithms, i.e. C^{3i}jDErpo, CCjDErpo and jDErpo was compared. There was no noticeable difference in computational cost of these different algorithms. Therefore, no further investigations into the computational cost were done in this work. It could does be concluded that C^{3i}jDErpo has the same computational cost as the CCjDErpo and jDErpo.

In general, it can be said that it is preferable to use C^{3i}jDErpo instead of CCjDErpo for large-scale benchmark functions, as either C^{3i}jDErpo performance is better or similar than CCjDErpo's, and there is no extra computational cost.

VII. Conclusions

In this paper, the Improved Constructive Cooperative Coevolutionary Differential Evolution ($C^{3i}DE$) algorithm for large-scale global optimisation is proposed and evaluated. $C^{3i}DE$ adopts a divide-and-conquer strategy to optimise subcomponents of the considered problem separately while cooperating with each other to co-adapt the individuals of the subpopulations during the optimisation. At the start of the optimisation with $C^{3i}DE$, the subpopulation of each subcomponent is initially optimised in a constructive fashion, by considering only the previously initialised subpopulations for the co-adaptation. The improvement of $C^{3i}DE$, compared to the previous version $C^{3}DE$, are that randomly selected collaborators from an elite fraction of the other subpopulations are used for the function evaluations. Resulting in that with $C^{3i}DE$, the greediness can be controlled, it becomes scalable in number of subcomponents, there are less user-parameters to tune, and no longer requires to have similar subcomponents. In this paper, the self-adaptive jDErpo algorithm proposed by Brest et al. [23] is used in the $C^{3i}DE$ to evolve the subpopulations.

The performance of $C^{3i}DE$ is evaluated on a set of high-dimensional benchmark functions (up to 1000 dimensions). The results are compared with CCDE and standalone DE. The results show that $C^{3i}DE$ has a superior performance on large-scale problems compared to DE. Furthermore, $C^{3i}DE$ outperforms CCDE on non-separable problems, whereas CCDE struggles with this type of problems. On separable problems, the performance of $C^{3i}DE$ is similar or better compared to CCDE. Also, there is no extra computational cost with $C^{3i}DE$ compared to CCDE. Hence, it can be concluded that in general, $C^{3i}DE$ is a competitive optimisation algorithm compared to CCDE.

Several mechanisms to control the greediness of $C^{3i}DE$ are proposed and tested in this paper. These included 100% greedy, 0% greedy, a different greediness during the initial optimisations, a dynamically adjusted greediness and a self-adaptive greediness per subcomponent. The results indicate that the 100% greedy option ($k = 1$) or a dynamically adjusted k-value are most preferable. It was also observed that the k-value becomes more important on non-separable problems then on separable problems.

Future work should include investigating how automatic and/or dynamic/adaptive decomposition strategies can be combined with $C^{3i}DE$. The problem decomposition can have a very significant influence on the performance of the algorithm. With an automatic decomposition strategy, it is not necessary to have a predefined decomposition, thereby making $C^{3i}DE$ easier and more convenient to use on a wider range of optimisation problems. Such a strategy could then also automatically determine the termination criteria for the initial optimisation of the subpopulations. Thereby eliminating the need for the user to determine this for each problem. In future work, the DE algorithm could also automatically adapt its population size during the optimisation, as proposed by Brest et al. [27] or Teng et al. [28]. Also, all tests in this work were done for single-objective problems, therefore in future work $C^{3i}DE$ should be evaluated for multi-objective problems. This type of problems are typically more difficult to solve, especially when there are many parameters that must be optimised.

Acknowledgment

This work was conducted at University West's *Production Technology West* research centre in Trollhättan, Sweden and was supported by Västra Götalandsregionen, Sweden, under the grant PROSAM 612-0974-14.

References

[1] F. Neri and V. Tirronen, "Recent advances in differential evolution: a survey and experimental analysis," *Artificial Intelligence Review*, vol. 33, no. 1-2, pp. 61–106, Oct. 2009.

[2] R. Storn and K. Price, "Differential Evolution A Simple and Efficient Heuristic for global Optimization over Continuous Spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, Dec. 1997.

[3] S. Das and P. Suganthan, "Differential Evolution: A Survey of the State-of-the-Art," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, Feb. 2011.

[4] A. Zamuda, J. Brest, B. Boskovic, and V. Zumer, "Large Scale Global Optimization using Differential Evolution with self-adaptation and cooperative co-evolution," in *Proc. Congr. Evolutionary Computation (CEC 08)*, 2008, pp. 3718–3725.

[5] M. Weber, F. Neri, and V. Tirronen, "Two algorithmic enhancements for the parallel differential evolution," *International Journal of Innovative Computing and Applications*, vol. 3, no. 1, pp. 20–30, Jan. 2011.

[6] O. Olorunda and A. Engelbrecht, "Differential evolution in high-dimensional search spaces," in *IEEE Congress on Evolutionary Computation, 2007. CEC 2007*, Sep. 2007, pp. 1934–1941.

[7] Z. Yang, J. Zhang, K. Tang, X. Yao, and A. Sanderson, "An adaptive coevolutionary Differential Evolution algorithm for large-scale optimization," in *IEEE Congress on Evolutionary Computation, 2009. CEC '09*, May 2009, pp. 102–109.

[8] M. Omidvar, X. Li, Z. Yang, and X. Yao, "Cooperative Co-evolution for large scale optimization through more frequent random grouping," in *IEEE Congress on Evolutionary Computation, 2010. (CEC'10)*, 2010, pp. 1–8.

[9] T. Ray and X. Yao, "A cooperative coevolutionary algorithm with Correlation based Adaptive Variable Partitioning," in *IEEE Congress on Evolutionary Computation, 2009. CEC '09*, May 2009, pp. 983–989.

[10] Y. Wang, B. Li, and X. Lai, "Variance priority based cooperative co-evolution differential evolution for large scale global optimization," in *Proc. Congr. Evolutionary Computation (CEC 09)*, 2009, pp. 1232–1239.

[11] Y. Zhenyu, T. Ke, and Y. Xin, "Multilevel cooperative coevolution for large scale optimization," in *IEEE Congress Evolutionary Computation*, 2008, pp. 1663–1670.

[12] M. Omidvar, Y. Mei, and X. Li, "Effective decomposition of large-scale separable continuous functions for cooperative co-evolutionary algorithms," in *2014 IEEE Congress on Evolutionary Computation (CEC)*, Jul. 2014, pp. 1305–1312.

[13] M. Omidvar, X. Li, Y. Mei, and X. Yao, "Cooperative Co-evolution with Differential Grouping for Large Scale Optimization," *IEEE Transactions on Evolutionary Computation*, vol. Early Access Online, 2013.

[14] E. Glorieux, F. Danielsson, B. Svensson, and B. Lennartson, "Constructive cooperative coevolutionary optimisation for interacting production stations," *The International Journal of Advanced Manufacturing Technology*, vol. 80, no. 1-4, pp. 673–688, 2015.

[15] E. Glorieux, B. Svensson, F. Danielsson, and B. Lennartson, "A Constructive Cooperative Coevolutionary Algorithm Applied to Press Line Optimisation," in *Proceedings of the 24th International Conference on Flexible Automation and Intelligent Manufacturing*, May 2014, pp. 909–917.

[16] E. Popovici, A. Bucci, R. P. Wiegand, and E. D. De Jong, "Coevolutionary principles," in *Handbook of Natural Computing*. Springer, 2012, pp. 987–1033.

[17] R. P. Wiegand, W. C. Liles, and K. A. D. Jong, "An Empirical Analysis of Collaboration Methods in Cooperative Coevolutionary Algorithms," in *Proc. from the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, 2001, pp. 1235–1242.

[18] G. A. Trunfio, "Adaptation in Cooperative Coevolutionary Optimization," in *Adaptation and Hybridization in Computational Intelligence*, ser. Adaptation, Learning, and Optimization, I. Fister and I. F. Jr, Eds. Springer International Publishing, 2015, no. 18, pp. 91–109.

[19] M. A. Potter and K. A. De Jong, "Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents," *Evol. Comput.*, vol. 8, no. 1, pp. 1–29, Mar. 2000.

[20] E. Glorieux, "Constructive cooperative coevolution for optimising interacting production stations," Licentiate Thesis, University West, Trollhttan, Sweden, 2015.

[21] E. Glorieux, F. Danielsson, B. Svensson, and B. Lennartson, "Optimisation of interacting production stations using a Constructive Cooperative Coevolutionary approach," in *2014 IEEE International Conference on Automation Science and Engineering (CASE)*, Aug. 2014, pp. 322–327.

[22] I. Poikolainen, F. Neri, and F. Caraffini, "Cluster-Based Population Initialization for differential evolution frameworks," *Information Sciences*, vol. 297, pp. 216–235, Mar. 2015.

[23] J. Brest, A. Zamuda, I. Fister, and B. Boskovic, "Some Improvements of the Self-Adaptive jDE Algorithm," in *2014 IEEE Symposium on Differential Evolution (SDE)*, Dec. 2014, pp. 1–8.

[24] M. Jamil and X. Yang, "A literature survey of benchmark functions for global optimisation problems," *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 4, no. 2, pp. 150–194, Jan. 2013.

[25] M. N. Omidvar, X. Li, and K. Tang, "Designing benchmark problems for large-scale continuous optimization," *Information Sciences*, vol. 316, pp. 419–436, Sep. 2015.

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

[27] J. Brest, B. Boskovic, A. Zamuda, I. Fister, and M. Maucec, "Self-adaptive differential evolution algorithm with a small and varying population size," in *2012 IEEE Congress on Evolutionary Computation (CEC)*, Jun. 2012, pp. 1–8.

[28] N. S. Teng, J. Teo, and M. H. A. Hijazi, "Self-adaptive population sizing for a tune-free differential evolution," *Soft Computing*, vol. 13, no. 7, pp. 709–724, Jul. 2008.