Enhancing the Scalability of Metaheuristics by Cooperative Coevolution

Ciprian Crăciun, Monica Nicoară, and Daniela Zaharie

Department of Computer Science, West University of Timişoara, Romania
{ccraciun,mnicoara,dzaharie}@info.uvt.ro

Abstract. The aim of this paper is to analyze the ability of cooperative coevolution to improve the scalability of population based metaheuristics. An extensive set of experiments on high dimensional optimization problems has been conducted in order to study the particularities and effectiveness of some elements involved in the design of cooperative coevolutionary algorithms: groupings of variables into components, choice of the context based on which the components are evaluated, length of evolution for each component. Scalability improvements have been obtained in the case of both analyzed metaheuristics: differential evolution and harmony search.

1 Introduction

Optimization of high dimensional functions is challenging and results obtained by population-based metaheuristics (e.g. evolutionary algorithms, particle swarm methods, differential evolution etc.) for small or medium size problems are not necessarily valid for high dimensional ones. Cooperative coevolution is a framework developed in the context of evolutionary algorithms in order to enhance their ability to solve complex problems. In the case of large scale problems the idea of cooperative coevolution is to decompose the problem into constituent subproblems and solve them by using cooperative evolutionary processes (e.g. multiple subpopulations each one corresponding to a subproblem). Designing a cooperative coevolution variant of a metaheuristic raises several issues, the most important ones being that of identifying the components of the problem and that of designing a cooperation strategy (e.g. choosing the context to evaluate the components). Since the first cooperative coevolutionary genetic algorithm has been proposed in [6], several cooperative approaches for function optimization have been developed [1, 7]. In some recent works [10, 11] a strategy based on a random grouping of variables which is applied to a synchronous differential evolution algorithm is proposed. The aim of this paper is to extend the studies in [10] by analyzing both synchronous and asynchronous coevolutionary strategies and the impact produced by the length of evolution of each component. The influence of the cooperative coevolution on the scalability has been tested for both Differential Evolution and Harmony Search, a recent population-based metaheuristics which has not been tested up to now for high dimensional problems.

The main elements of the cooperative coevolutionary approach are presented in Section 2. Section 3 describes the particularities of Differential Evolution and Harmony Search while Section 4 contains the numerical results. Some conclusions are presented in Section 5.

2 Cooperative Coevolution for Large Scale Optimization

Let us consider the problem of minimizing a function $f : D = [a_1, b_1] \times \ldots \times [a_n, b_n] \rightarrow \mathbb{R}$. Any evolutionary approach for solving such a problem is based on evolving a population of potential solutions, $X = \{x_1, \ldots, x_m\} \subset D$, for a given number of generations, G. As n is larger, the number of generations needed to approximate the optimum with a given accuracy is also larger. If the required G does not depend linearly on n then we face the scalability problem. A possible solution to the scalability issue is to decompose the problem in subproblems by dividing the n-dimensional vector corresponding to a potential solution in smaller size components and by co-evolving in a cooperative way these components [6]. There are two main questions which should be answered when designing a cooperative coevolutionary algorithm: (i) how to divide the population elements in smaller size components? (ii) how to coevolve these components? For each of these questions there are several possible answers which are shortly reviewed in the following.

Defining the components. Dividing a n-dimensional vector in $K \leq n$ components is equivalent with defining a function $c : \{1, \ldots, n\} \rightarrow \{1, \ldots, K\}$ which assigns to each variable i its corresponding component, $c(i)$. Thus for a vector $x = (x^1, x^2, \ldots, x^n)$ the k-th component can be defined as $C_k(x) = \{x^j | j \in c^{-1}(k)\}$, i.e. the set of all variables x^j having its index, j, in the pre-image of k through the function c. The original vector x can be obtained by *merging* all its components, i.e. $x = \langle C_1(x), \ldots, C_K(x) \rangle$. If $X = \{x_1, x_2, \ldots, x_m\}$ denotes a population then $C_k(X) = \{C_k(x_1), \ldots, C_k(x_m)\}$ will denote the (sub)population corresponding to the k-th component. In the traditional cooperative coevolutionary algorithms [6] the number of components is equal to the number of variables ($K = n$) and the function c satisfies $c(i) = i$. Such an approach is rather inappropriate in the case of interrelated variables, when an appropriate decomposition should group in the same component the highly correlated variables. However, in practice, is difficult to know which variables are significantly correlated, therefore a compromise solution is to use random groupings as is proposed in [10]. The groupings used in [10] are constructed based on random permutations and are characterized by equally sized groups. The size of each group depends on the number of groups which should be preset. In order to avoid the choice of the number of groups the same authors recently proposed in [11] an adaptive variant which selects between several possible values for the group sizes (e.g. $\{5, 10, 25, 50, 100\}$) according to some selection probabilities (computed by using historical data on the performance of different group sizes). A simpler approach is that based on randomly choosing the number of components $K \in \{K_{min}, \ldots, K_{max}\}$ and on randomly assigning the variables to components. Thus the components do not

Algorithm 1 A generic synchronous cooperative coevolutionary algorithm

`CCEvolveS`(X)

Initialize and evaluate X
while goal for X not reached **do**
  Choose K
  Construct $C_1(X),\ldots,C_K(X)$
  for $k = \overline{1,K}$ **do**
    $C_k(V) \leftarrow$ `evolveS`$(C_k(X),X)$
  end for
  Evaluate V
  $X \leftarrow$ `select`(X,V)
end while
return X

`evolveS`$(C_k(X),X)$

$C_k(W) \leftarrow C_k(X)$
while goal for $C_k(X)$ not reached **do**
  $C_k(Y) \leftarrow \text{generate}(C_k(W))$
  $Y \leftarrow \text{merge}(C_k(Y),X)$
  evaluate Y
  $C_k(Y) \leftarrow \text{select}(C_k(Y), C_k(W))$
  $C_k(W) \leftarrow C_k(Y)$
end while
return $C_k(Y)$

Algorithm 2 A generic asynchronous cooperative coevolutionary algorithm

`CCEvolveA`(X)

Initialize and evaluate X
while goal for X not reached **do**
  Choose K
  Construct $C_1(X),\ldots,C_K(X)$
  for $k = \overline{1,K}$ **do**
    while goal for $C_k(X)$ not reached **do**
      for each $C_k(x) \in C_k(X)$ **do**
        $C_k(x) \leftarrow$ `evolveA`$(C_k(x),C_k(X),X)$
      end for
    end while
  end for
end while
return X

`evolveA`$(C_k(x),C_k(X),X)$

$C_k(y) \leftarrow \text{generate}(C_k(X))$
$y \leftarrow \text{merge}(C_k(y),X)$
evaluate y
$C_k(y) \leftarrow \text{select}(C_k(x),C_k(y))$
return $C_k(y)$

necessarily have exactly the same size, but only their average size is the same. The numerical tests we conducted illustrated the fact that the behavior of this simple variant is similar to that of the adaptive variant proposed in [11].

Coevolving the components. Two variants of coevolving the components, a synchronous and an asynchronous one, are presented in Algorithms 1 and 2, respectively. The main difference between these two variants is related to the moment when the context used in evaluating the components is updated. In the synchronous variant the global population X (which represents the evaluation context) is updated only after all components have been evolved while in the asynchronous case the global population is updated after evolving each component. Since the metaheuristic itself can be synchronous or asynchronous we combined these strategies in order to obtain a fully synchronous (Algorithm 1) and a fully asynchronous (Algorithm 2) variant. Numerical experiments have been conducted for both variants and some comparative results are presented in Section 4.

The function `evolveS`($C_k(X)$, X) returns the population corresponding to component k obtained by applying a population-based metaheuristic to $C_k(X)$ using the context X. The stopping condition of the while loop inside `evolveS` is related to the number of generations allowed for each component. On the other hand the function `evolveA`($C_k(x)$,$C_k(X)$, X) returns just one element corresponding to component k. The new element is generated starting from the current element and other elements selected from the population $C_k(X)$ according to the used metaheuristic. The evaluation of the generated element is made using as context the current global population, X. When implementing the coevolutionary process, besides the choice of a metaheuristic there are two other issues to be solved: (i) how to choose collaborators from the context when evaluating $C_k(X)$? (ii) how many resources (e.g. number of generations or number of function evaluations) should be allocated to the evolution of each component? The first question has been addressed in several previous works [6, 9] and the typical variants to choose the collaborators are: the best element of the population, the current element in the population and a random one. Concerning the second issue some authors use just one generation for each component [1, 6] while others suggest to allocate a given number of function evaluations for each component [10]. However there is no systematic study regarding the influence of the evolution length for each component on the behavior of the coevolutionary approach. Therefore one of the aims of the numerical experiments we conducted is to analyze the influence of the evolution length per component on the behavior of cooperative coevolution.

3 Cooperative Coevolutionary Differential Evolution and Harmony Search

In order to analyze the impact of the cooperative coevolution on the metaheuristics scalability we implemented cooperative coevolutionary variants for two metaheuristics: Differential Evolution (DE) [3] and Harmony Search (HS) [2]. The reason for choosing the Differential Evolution technique is the existence of some recent results on coevolutionary DE [10], [11] while the reason for choosing Harmony Search is the fact that all reported results concerning HS are for small size problems. Let us shortly describe DE, HS and their cooperative coevolutionary variants. In the case of DE technique for each population element, x_i, a so-called trial element, $y_i = (y_i^1, \ldots, y_i^n)$, is constructed according to Eq. (1). The offspring of x_i is the best element between x_i and y_i.

$$y_i^j = \begin{cases} x_{r_1}^j + F_i \cdot (x_{r_2}^j - x_{r_3}^j) & \text{if } U < CR_i \\ x_i^j & \text{otherwise} \end{cases}, \quad j = \overline{1,n} \tag{1}$$

In Eq. (1), r_1, r_2 and r_3 are distinct random elements from $\{1, \ldots, m\}$ and U is a random value uniformly generated in $[0, 1]$ for each i and j. The control parameters F_i (scale factors) and CR_i (crossover rates) are adapted according to the rules used in Brest's jDE [4] (each element has its own parameters F

and CR, which are randomly initialized and during the evolutionary process are reset, with a small probability, to other random values). Thus the only parameter to be specified is the population size, m. There are two main variants for designing the DE evolutionary process: a synchronous and an asynchronous one. The synchronous DE (denoted as sDE in the experiments) replaces the population elements with their offsprings only after the offsprings were generated for the entire population. The asynchronous DE (denoted as aDE) corresponds to the steady state approach in evolutionary algorithms and replaces the current element with its offspring just after the construction of the offspring. For each DE variant the coevolutionary variant is obtained by using the corresponding DE generation rule in Step 3 of `evolveS` and in Step 1 of `evolveA`, respectively. Thus one obtains two algorithms: a synchronous cooperative coevolutionary DE (denoted as sCCDE) and the asynchronous version (denoted as aCCDE).

Harmony Search (HS) has been developed by Z. W. Geem et al. in 2001 [2], starting from an interesting analogy between the music improvisation process and the search for the optimum. Based on this analogy, HS constructs new potential solutions by combining in a stochastic way the elements of the current population and by randomly perturbing them. The rule to construct a new element, y_i, is given in Eq. (2). In the classical variant this new element replaces the worst element in the population if it is better than it. Thus HS follows an asynchronous strategy, therefore the coevolutionary HS (denoted as CCHS) should also be based on the asynchronous approach. Due to the particularities of the coevolutionary approach the new element replaces not the worst but the current one in the population (ith element).

$$y_i^j = \begin{cases} x_r^j + bw \cdot U_3 & \text{if } U_1 < p_1 \text{ and } U_2 < p_2 \\ x_r^j & \text{if } U_1 < p_1 \text{ and } U_2 \geq p_2 \\ \text{rand}(a_j, b_j) & \text{otherwise} \end{cases}, \quad j = \overline{1,n} \tag{2}$$

In Eq. (2), r is a random element of $\{1, \ldots, m\}$, U_1, U_2 denotes random values uniformly generated in $(0, 1)$ for each j, U_3 is randomly generated in $(-1, 1)$, $p_1 \in (0, 1)$ denotes the probability of taking a variable from the current population (HMCR in [2]), $p_2 \in (0, 1)$ denotes the probability to perturb the variable (PAR in [2]) and $\text{rand}(a_j, b_j)$ is a random value in the domain corresponding to the jth variable. The values of these parameters used in our experiments are $p_1 = 0.99$, $p_2 = 0.75$. The parameter bw controls the size of the perturbation and recently in [5] has been proposed an effective way of choosing bw based on the variance of the current population, i.e. $bw = \sqrt{var(X^j)}$ where $var(X^j)$ is the variance of the jth variable over the current population X. Despite the fact that HS has been successfully applied to various optimization problems its behavior on large scale ones has not been studied up to now. The first tests on problems having a size larger than 50 suggested that HS does not scale well with the problem size. Therefore we designed a cooperative coevolutionary variant (CCHS) based on using in Algorithm 2 the generation rule specified in Eq. (2) .

4 Numerical results

For both classical and coevolutionary DE and HS variants we analyzed two aspects: (i) the amount of resources needed to reach a given performance, i.e. the number of functions evaluations, nfe_ϵ, needed to approximate the optimal value with accuracy ϵ; (ii) the performance reached by using a given amount of resources, i.e. the best value, f_*, obtained after a given number of function evaluations, nfe_*. In all experiments the accuracy is $\epsilon = 10^{-10}$, the maximal number of functions evaluations is $nfe_* = 5000 \cdot n$ and the reported values are obtained as averages over 30 independent runs. As benchmark we used a subset, consisting of two non-separable functions (Ackley and Griewank) and a separable one (Rastrigin), of the test suite provided in [8]. The experimental design consisted in combining for each coevolutionary variant (sCCDE, aCCDE and CCHS) two values for the number of components ($K = 10$ and $K = 20$) with six possible values for the length of evolution per component (expressed as number of generations: $G_k \in \{1, 5, 10, 50, 100, n/K\}$). In a set of preliminary tests we analyzed several variants of choosing the collaborators when evaluating a component: the current element in the population, the best one and a randomly selected one. Since the results of these tests suggested that the variant using as collaborator the current element behaves the best this is the variant we used in all experiments presented here. The population size was set to $m = 100$ for all coevolutionary variants. For non-coevolutionary variants (sDE, aDE and HS) m was set to 100 when $n < 500$ while for $n \geq 500$ it was set to $m = 200$.

Table 1 contains for each method the results corresponding to the pair (K, G_k) which led to the smallest averaged f_*. The results illustrate that while for DE small number of components and large value of the evolution length are more beneficial, in the case of HS the situation is the reversed (a larger number of components and a short evolution per component). Table 2 shows that all coevolutionary variants have a good reliability (the success ratio is 1) and the scalability factor, which should be closer to $n/100$, is significantly improved especially in the case of DE. Concerning the comparison between the synchronous and asynchronous variants, by applying a t-test with a significance level of 0.05, one obtained that the number of functions evaluations needed to reach a given accuracy (nfe_ϵ) is smaller in the case of aCCDE than in the case of sCCDE. On the other hand, with respect to f_* values, the synchronous and asynchronous variants behaves almost similarly (only in three cases sCCDE produced results significantly better than aCCDE).

5 Conclusions

Several cooperative coevolutionary variants obtained by combining different ways of updating the contextual population, different numbers of components, different lengths of the evolution/component have been numerically analyzed. The obtained results illustrate the fact that the asynchronous variant behaves slightly better than the synchronous one with respect to the number of function evaluations and that the adequate length of the evolution/component depends on the

Table 1. Influence of the method, the number of components (K) and length of evolution per component (G_k) on the performance (f_*) reached after $5000 \cdot n$ function evaluations.

Method	Ackley			Griewank			Rastrigin		
	K	G_k	f_*	K	G_k	f_*	K	G_k	f_*
n=100									
sDE	-	-	$(8.8 \pm 2.2) \cdot 10^{-14}$	-	-	$(3.0 \pm 0.7) \cdot 10^{-14}$	-	-	$(3.2 \pm 17) \cdot 10^{-2}$
aDE	-	-	$(9.9 \pm 2.0) \cdot 10^{-14}$	-	-	$(3.1 \pm 8.5) \cdot 10^{-14}$	-	-	$(3.2 \pm 17) \cdot 10^{-2}$
sCCDE	2	50	$(8.8 \pm 1.3) \cdot 10^{-14}$	2	50	$(2.9 \pm 0.5) \cdot 10^{-14}$	2	50	0 ± 0
aCCDE	2	50	$(8.5 \pm 1) \cdot 10^{-14}$	2	50	$2.8 \cdot 10^{-14} \pm 0$	2	50	0 ± 0
HS	-	-	$(8.7 \pm 1.1) \cdot 10^{-5}$	-	-	$(3.3 \pm 2.7) \cdot 10^{-8}$	-	-	150 ± 4.5
CCHS	10	1	$2.7 \cdot 10^{-13} \pm 0$	10	1	$(5.9 \pm 0.8) \cdot 10^{-14}$	10	1	$(2.0 \pm 2.7) \cdot 10^{-14}$
$n = 500$									
sDE	-	-	$(1.0 \pm 3) \cdot 10^{-11}$	-	-	$(2.4 \pm 1.3) \cdot 10^{-3}$	-	-	1.59 ± 1.50
aDE	-	-	$(4.3 \pm 6.5) \cdot 10^{-12}$	-	-	$(2.3 \pm 0.1) \cdot 10^{-2}$	-	-	1.8 ± 1.3
sCCDE	5	100	$(4.6 \pm 0.2) \cdot 10^{-13}$	5	100	$(1.6 \pm 0.08) \cdot 10^{-13}$	10	50	$(1.3 \pm 2.4) \cdot 10^{-14}$
aCCDE	5	100	$(4.8 \pm 0.3) \cdot 10^{-13}$	5	100	$(1.7 \pm 0.1) \cdot 10^{-13}$	10	50	$(9.4 \pm 21) \cdot 10^{-15}$
HS	-	-	16 ± 0.07	-	-	$(1.9 \pm 0.04) \cdot 10^{3}$	-	-	$(1.6 \pm 0.01) \cdot 10^{3}$
CCHS	20	1	$(9.1 \pm 0.1) \cdot 10^{-13}$	20	1	$(3.8 \pm 1.8) \cdot 10^{-13}$	20	5	$(5.1 \pm 1.7) \cdot 10^{-14}$
$n = 1000$									
sDE	-	-	$(4.0 \pm 4.6) \cdot 10^{-1}$	-	-	$(4.1 \pm 4.4) \cdot 10^{-1}$	-	-	34.5 ± 14.7
aDE	-	-	$(9.3 \pm 3.7) \cdot 10^{-1}$	-	-	$(3.8 \pm 5.5) \cdot 10^{-1}$	-	-	47.6 ± 13.6
sCCDE	10	100	$(1.0 \pm 0.04) \cdot 10^{-12}$	10	100	$(3.6 \pm 0.1) \cdot 10^{-13}$	20	5	$(4.5 \pm 2.2) \cdot 10^{-14}$
aCCDE	10	100	$(1.0 \pm 0.05) \cdot 10^{-12}$	10	100	$(3.7 \pm 0.1) \cdot 10^{-13}$	20	50	$(4.5 \pm 2.2) \cdot 10^{-14}$
HS	-	-	19 ± 0.03	-	-	$(8.6 \pm 0.09) \cdot 10^{3}$	-	-	$(4.6 \pm 0.04) \cdot 10^{3}$
CCHS	20	1	$(2.5 \pm 0.04) \cdot 10^{-12}$	20	1	$(5.8 \pm 0.1) \cdot 10^{-13}$	20	1	$(2.0 \pm 0.6) \cdot 10^{-13}$

method (it should be small in the case of HS and can be larger in the case of DE). Concerning the scalability of HS one can conclude that it can be significantly improved by coevolution.

Acknowledgments. This work is supported by Romanian project PNCD II 11-028/ 14.09.2007 (NatComp).

References

1. van den Bergh, F., Engelbrecht, A.P.: A Cooperative Approach to Particle Swarm Optimization. IEEE Transactions on Evolutionary Computation, vol. 8, no. 3, 225–239 (2004)
2. Geem, Z. W., Kim, J., Loganathan, G., A New Heuristic Optimization Algorithm: Harmony Search, Simulation, 76(2), 60–68 (2001)
3. K.V. Price, R. Storn, J. Lampinen, Differential Evolution. A Practical Approach to Global Optimization, Springer, 2005.
4. J. Brest, B. Bošković, S. Greiner, V. Žurner, M.S. Maučec, Performance comparison of self-adaptive and adaptive differential evolution algorithms, Soft Computing, 11, issue 7, 617 – 629 (2007)

Table 2. Influence of the method on the success ratio (SR), scalability factor ($SF = nfe_\epsilon(n)/nfe_\epsilon(100)$) and efficiency expressed as the number of functions evaluations needed to reach the accuracy $\epsilon = 10^{-10}$ (nfe_ϵ).

Method	Ackley			Griewank			Rastrigin		
	SR	SF	nfe_ϵ	SR	SF	nfe_ϵ	SR	SF	nfe_ϵ
n=100									
sDE	1	1	238766 ± 3399	1	1	141766 ± 3726	0.96	1	153548 ± 4533
aDE	1	1	221766 ± 4533	1	1	132433 ± 4229	0.96	1	144582 ± 4973
sCCDE	1	1	322963 ± 1813	1	1	193683 ± 6432	1	1	200080 ± 4040
aCCDE	1	1	297433 ± 4422	1	1	179100 ± 7000	1	1	186100 ± 4898
HS	0	-	-	0	-	-	0	-	-
CCHS	1	1	306433 ± 4818	1	1	182433 ± 6155	1	1	218433 ± 4533
$n = 500$									
sDE	0.96	5.9	$1427786 \pm 6E5$	0.96	6.4	$910544 \pm 4E4$	0.36	6.1	$9456540 \pm 6E4$
aDE	1	6.0	$1350200 \pm 4E4$	0.9	6.7	$896496 \pm 4E4$	0.20	6.5	$9502000 \pm 5E4$
sCCDE	1	3.8	$1257610 \pm 1E4$	1	4.1	$798360 \pm 1E4$	1	5.2	$1050530 \pm 8E3$
aCCDE	1	4.0	1200100 ± 0	1	4.1	$751766 \pm 2E4$	1	5.3	$986766 \pm 2E4$
HS	0	-	-	0	-	-	0	-	-
CCHS	1	5.2	1600100 ± 0	1	5.4	1001767 ± 8975	1	7.1	$1551767 \pm 1E4$
$n = 1000$									
sDE	0.1	18	$4400200 \pm 3E5$	0.16	18	$2560200 \pm 2E5$	0	-	-
aDE	0.06	17	$3900200 \pm 3E5$	0.46	18	$2400200 \pm 2E5$	0	-	-
sCCDE	1	7.9	$2576007 \pm 4E4$	1	8.3	$1615047 \pm 3E4$	1	10	$2016733 \pm 1E4$
aCCDE	1	8.0	2400100 ± 0	1	8.4	$1513433 \pm 3E4$	1	10	2000100 ± 0
HS	0	-	-	0	-	-	0	-	-
CCHS	1	14	4300100 ± 0	1	15	2800100 ± 0	1	20	$4396767 \pm 6E4$

5. Mukhopadhyay, A., Roy, A., Das, S., Das, S., Abraham, A.: Population variance and explorative power of Harmony Search: An analysis, Proc. ICDIM 2008, 775–781 (2008)
6. Potter, M., De Jong, K.: A cooperative coevolutionary approach to function optimization. In Proc. of PPSN, 249–257 (1994)
7. Shi, Y., Teng, H., Li, Z.: Cooperative Co-evolutionary Differential Evolution for Function Optimization. In L. Wang, K. Chen and Y.S. Ong (eds): ICNC 2005, LNCS 3611, 1080–1088 (2005)
8. Tang,K., Yao, X., Suganthan,P.N., MacNish,C., Chen, Y.P., Chen, C.M. and Yang, Z.: Benchmark Functions for the CEC2008 Special Session and Competition on Large Scale Global Optimization, Technical Report, USTC, China, http://nical.ustc.edu.cn/cec08ss.php (2007)
9. Wiegand, R.P, Liles, W.C., De Jong, K.A.: An Empirical Analysis of Collaboration Methods in Cooperative Coevolutionary Algorithms. Proc. of Genetic and Evolutionary Computation Conference, Morgan Kaufmann Publ., 1235–1242 (2001)
10. Yang, Z., Tang, K., Yao, X.: Large scale evolutionary optimization using cooperative coevolution. Information Sciences, 178, 2985–2999 (2008)
11. Yang, Z., Tang, K., Yao, X.: Multilevel Cooperative Coevolution for Large Scale Optimization. Proc. of the 2008 IEEE Congress on Evolutionary Computation, IEEE Press, 1663–1670 (2008)