Proceedings Paper:
Bali, K, Chandra, R and Omidvar, MN orcid.org/0000-0003-1944-4624 (2016) Contribution based multi-island competitive cooperative coevolution. In: 2016 IEEE Congress on Evolutionary Computation (CEC). 2016 IEEE Congress on Evolutionary Computation (CEC), 24-29 Jul 2016, Vancouver, BC, Canada. IEEE , pp. 1823-1830. ISBN 978-1-5090-0623-6

https://doi.org/10.1109/cec.2016.7744010

Contribution Based Multi-Island Competitive Cooperative Coevolution

Kavitesh Bali*, Rohitash Chandra*, and Mohammad N. Omidvar†
* School of Computing Information and Mathematical Sciences, University of the South Pacific, Suva, Fiji.
Artificial Intelligence and Cybernetics Research Group, Software Foundation, Nausori, Fiji.
http://aicrg.softwarefoundationfiji.org/, Email: {bali.kavitesh, c.rohitash}@gmail.com
† School of Computer Science, University of Birmingham,
Birmingham B15 2TT, U.K., E-mail: m.omidvar@cs.bham.ac.uk)

***Abstract*—Competition in Cooperative Coevolution (CC) has demonstrated success in solving global optimization problems. In a recent study, a multi-island competitive cooperative coevolution (MIC^3) algorithm was introduced, which featured competition and collaboration of several different problem decomposition strategies implemented as independent islands. It was shown that MIC^{3}converges to high quality solutions without the need to find an optimal decomposition. MIC^{3}splits the computational budget in terms of number of function evaluations equally amongst all the islands and evolves them in a round-robin fashion. This overlooks the difference in contributions of the different islands towards improving the overall objective function value. Therefore, a considerable amount of function evaluations are wasted on the low-contributing islands as their problem decomposition strategies may not appeal to the problem at the given stage of the evolutionary process. This paper proposes Contribution Based MIC^{3}algorithms (MIC^4) that quantifies the contributions of each island and allocates the computational budget accordingly. Experimental analysis reveals that MIC^{4}outperforms MIC^3.**

I. Introduction

Coevolutionary algorithms have gained popularity as a vital extension to traditional population based evolutionary algorithms [1, 2]. Cooperative coevolution (CC) divides a large problem into a set of subcomponents [1] in order to simplify its complexities and solve them through decomposition [1, 3, 4]. Applications of CC span across a wide range of areas that include real parameter large scale global optimization [5, 6], neuro-evolution for time series prediction, classification and control problems [2, 4, 7, 8].

Cooperative coevolution features decomposition that is defined by the number and size of subcomponents implemented as sub-populations. A drawback of cooperative coevolution is that it's performance is sensitive to problem decomposition [9]. Cooperative coevolution naturally appeals to fully separable problems; howsoever, many real-world applications are partially separable. In order to make cooperative coevolution effective, it is important to form groups of interacting variables in order to minimize the interdependence between subcomponents [1, 3]. Capturing interacting variables and accurately grouping them into separate subcomponents has been a challenge of cooperative coevolution [10, 11]. Hence, identifying an optimal decomposition strategy is a cumbersome task requiring extensive experimentation. In the literature, various strategies have been utilized for problem decomposition where variables have been grouped based on their interactions[5, 9, 12–17].

The canonical implementations of cooperative coevolution [1] gives all the sub-populations the same local evolution time irrespective of their contributions, which is a waste of the computational budget. Omidvar et al. [18] introduced a Contribution Based Cooperative Coevolution (CBCC) technique that quantifies the contribution of each subcomponent towards improving the overall objective function value and splits the computational budget accordingly [18]. It was shown that CBCC saves considerable amount of resources and outperforms the canonical CC algorithms [5, 18, 19].

Competition and collaboration features have shown to be advantageous in cooperative coevolution [20, 21]. A multi-island competitive cooperative coevolution method (MIC^3) was introduced in which various problem decomposition strategies were implemented as islands that compete and collaborate to optimize a problem [22]. Experimental results demonstrated that MIC^{3}outperforms standalone traditional CC and converges to high quality solutions without having the need to find an optimal decomposition. The current MIC^{3}algorithm splits the computational budget equally amongst all the islands. A scaled up analytical study of MIC^{3}revealed that not all the islands contribute equally towards the overall fitness; hence, a considerable amount of function evaluations are wasted [23].

This gives motivation to divide the computational budget more wisely according to the contribution of each of the islands during the evolutionary process. This paper proposes contribution based multi-island competitive cooperative coevolution (MIC^4) algorithm to improve the performance of MIC^{3}by using more efficient resource management schemes. We introduce two different techniques to quantify the contributions of each of the islands in order to retain the stronger islands and eliminate the weaker islands. In particular, the aim of this paper is to answer the following questions:

- Is it beneficial to eliminate the weaker islands at an early evolution stage and invest time in stronger islands in order to converging to higher quality solutions?
- Will it be favorable to first trigger a warning to the weaker islands at the initial stage and only eliminate if

the performance of the islands still do not improve in the next stage?

To answer the above questions, the performance of MIC^{4}algorithm is evaluated on eight different benchmark functions and compared with MIC^3.

The organization of the rest of this paper is as follows. Section II describes the preliminaries and background information. Section III describes the proposed method and its application to different classes of problems. Experimental results and their analysis are provided in Section IV-B. Section V concludes the paper with a brief discussion of future work.

II. Background

A. Competition in Cooperative Coevolution

In nature, competition is perceived to be ubiquitous as an agent of natural selection that structures the community of species with given resources [24]. In evolution, individuals compete and collaborate with each other for survival when give limited resources [25, 26]. Competition in evolutionary algorithms was introduced using a *host-parasite* model, where two populations competed with each other and sanctioned fitness sharing, elitism and selection [27]. Nitschke and Langenhoven [28] studied the simulation of predator and prey behaviors using artificial neural networks within a competitive coevolution procedure [28].

Competition has also been implemented to address the problem of efficient regulation mechanism between local search and global search in an evolution algorithm based on a cloud model (CEBA) [25]. It also employs competition in evolution between sub-populations to ensure global convergence and stability. Furthermore, competition has been applied in cooperative coevolutionary algorithms to solve multi-objective problems [29]. Enforcing competition has shown to be an ideal approach for multi-objective optimization in dynamic environments. Competition allows an adaptive problem decomposition technique that adapts to environmental changes while solving multi-objective problems [29].

Scheepers and Engelbrecht [30] developed a competitive coevolutionary team-based particle swarm optimization (CCPSO) algorithm to train soccer agents (players) from zero knowledge [30]. A FIFA based fitness function was introduced to show that the competitive algorithm outperforms other unbiased relative fitness functions which initially affected the training results of the players having caused performance outliers. Competitive coevolution has also been applied to evolve playing strategies for the iterated prisoner's dilemma (IPD) [31] and the well-known game of tic-tac-toe (noughts and crosses) [32]. For such games, various particle swarm optimization and co-evolutionary techniques have been utilized to train neural networks to compete well. Chellapilla and Fogel [33] also utilized the principles of biological evolution to train artificial neural networks to play a game of checkers. It was shown that their algorithm could compete with most professional human players [33].

To preserve diversity and avoid premature convergence, various methods have been proposed that simulate distributed evolution through distributed population algorithm for global optimization [20, 34–36]. These distributed population algorithms commonly use panmictic sub-populations that apply the standard evolutionary algorithm within each island in isolation [37]. The strongest individuals are migrated between islands replacing the weaker ones. This particular fitness-based migrant selection and insertion can be considered an added selection pressure during collaboration [34, 37]. The first *island* model for evolutionary algorithms is an example of a distributed population model where sub-populations are isolated during selection, breeding and evaluation [35] . In this method, the islands pivot the evolutionary processes locally within their sub-populations before migrating fitter individuals to other islands after certain generations. The migrant selection in this scenario is done randomly and not probabilistically.

B. Multi-Island Competitive Cooperative Coevolution

Competitive island cooperative coevolution (CICC) [21, 38] and multi-island competitive cooperative coevolution (MIC^3) [22, 23] were proposed for solving global optimization problems. In CICC, two different problem decompositions are implemented as islands that compete and collaborate to solve a problem. MIC^{3}is a successor to CICC, which is generalized to deal with more than two islands and is shown in Algorithm 1. Table I contains a short description important variables and parameters of MIC^3.

Broadly speaking, CICC and MIC^{3}are basically canonical CC's working in parallel on multiple islands, where each island uses a different problem decomposition. The essence of the proposed method is that particular islands employ different decompositions of the original problem, and each of them uses a local CC that obeys that decomposition. Best performing solutions migrate between the islands, which requires them to be first composed from sub-populations of the source islands and then decomposed according to the decomposition scheme of the target island. It was shown that competition and collaboration of different decomposition methods exhibiting various features can yield solutions with a quality better than individual decomposition methods used in isolation [20, 21, 38]. Moreover, competition can ensure

Algorithm 1: $(x^\star, f^\star)$=MIC^3$(f, n, \underline{\mathbf{x}}, \overline{\mathbf{x}}, \mathcal{D}, \mu, C, \gamma, \Gamma_{\max})$

Stage 1: Initialization.
for $i \in \{1, \ldots, I_{\max}\}$ **do**
  $\texttt{rand}(\mathbf{I}_i, \mu, n, \underline{\mathbf{x}}, \overline{\mathbf{x}})$;
  $\mathbf{b}_i = \texttt{eval}(\mathbf{I}_i)$;
  $\Gamma_i = \mu$;
Stage 2: Evolution.
while $\sum_{i=1}^{I_{\max}} \Gamma_i < \Gamma_{\max}$ **do**
  for $i \in \{1, \ldots, I_{\max}\}$ **do**
    for $c \in \{1, \ldots, C\}$ **do**
      for $j \in \{1, \ldots, |\mathcal{D}_i|\}$ **do**
        $\mathbf{b}_i = \texttt{optimizer}(\mathbf{I}_i, \mathbf{b}_i, \mathcal{D}_{ij}, \gamma)$;
        $\Gamma_i = \Gamma_i + \mu \cdot (\gamma + 1)$;
  Stage 3: Competition: Compare and mark the island with the best fitness.
  Stage 4: Collaboration: Injecting the best individual from *Winner* island into all the other islands.

TABLE I
A SHORT DESCRIPTION OF THE IMPORTANT VARIABLES USED IN ALG. 1.

Variable	Description
$\mathbf{x}^\star$	the best solution vector found by the algorithm.
$f^\star$	the objective value of $\mathbf{x}^\star$.
f	the function handle of the objective function.
$\underline{\mathbf{x}}$	vector of lower bound constrains of the decision variables.
$\overline{\mathbf{x}}$	vector of the upper bound constrains of the decision variables.
μ	the population size.
n	the dimensionality of the objective function.
$\Gamma_{\max}$	the maximum number of available objective function evaluations.
Γ_i	the objective function evaluations used by the ith island.
γ	the number of times that the subcomponent optimizer optimizes each subcomponent in a CC context.
$\mathbf{b}_i$	the best solution found by the ith island. This is also used as a context vector by the optimizer to construct a complete solution for evaluation of subcomponents.
$\mathcal{D}$	A set containing a decomposition for each island. For example, $\mathcal{D}_i$ contains the decomposition for the ith island.
C	the number of times an island is optimized before optimizing the next island.

that these different problem decomposition methods are given an opportunity during the course of optimization phase, and there is no problem in finding the right decomposition method at a particular time according to the degree of separability [4]. CICC and MIC^{3}alleviate the need to find an optimal decomposition and generates high quality solutions than standalone CC.

III. CONTRIBUTION BASED MULTI-ISLAND COMPETITIVE COOPERATIVE COEVOLUTION

In this section, we propose two multi-island contribution based competitive cooperative coevolution (MIC^4) for optimizing the performance of traditional MIC^3. The contribution of each of the islands is quantified by measuring the number of times an island wins and loses at different stages of optimization. A higher win count determines the superiority of stronger islands over the weaker ones.

Bali and Chandra [23] conducted an analysis of MIC^{3}where they discovered that some of the islands may get stagnant and do not contribute for several phases during the optimization process. Moreover, they noted that there are cases in which some of the islands may not contribute in the beginning, but become helpful at later stages of evolution. In order to effectively utilize the islands, we propose two different strategies to eliminate the poor performing islands. In the first strategy (a.k.a. the kill strategy) the islands that do not make a significant contribution for several rounds are eliminated from the evolutionary process. In the other strategy (a.k.a. the warn-then-kill strategy), a warning is issued to the poor performing island, and then eliminated if no improvement is seen.

A. The Kill Strategy

In this strategy, MIC^{4}employs a straightforward greedy approach and eliminates the poor performing islands. More specifically, this strategy measures the contribution of every island by counting the number of times it loses/wins a tournament (competition among all islands). All the islands that do not win a tournament for τ_{k} consecutive runs will be eliminated from the evolutionary process. Several consecutive losses implies that the solution quality of an islands is poor; hence, it is concluded that it's contribution to the overall fitness is minimal, and it should be terminated to save the limited computational resources.

Algorithm 2: $(x^\star,f^\star)$=MIC$^4(f,n,\underline{\mathbf{x}},\overline{\mathbf{x}},\mathcal{D},\mu,C_{\max},C_{\mathrm{pen}},\gamma,\Gamma_{\max},\tau_{\mathrm{k}},\tau_{\mathrm{w}},s)$

Stage 1: Initialization.
for $i \in \{1,\ldots,I_{\max}\}$ **do**
  rand$(\mathbf{I}_i,\mu,n,\underline{\mathbf{x}},\overline{\mathbf{x}})$;
  $\mathbf{b}_i$ = eval$(\mathbf{I}_i)$;
  $\Gamma_i = \mu$;
Stage 2: Evolution.
$\mathcal{K} = \{\}$; $\mathcal{W} = \{\}$;
$w_i = 0, \forall i \in \{1,\cdots,I_{\max}\}$;
$t = 0$;
while $\sum_{i=1}^{I_{\max}} \Gamma_i < \Gamma_{\max}$ **do**
  for $i \in \{1,\ldots,I_{\max}\} \setminus \mathcal{K}$ **do**
    if $i \in \mathcal{W}$ **then**
      $C = C_{\mathrm{pen}}$;
    else
      $C = C_{\max}$;
    for $c \in \{1,\ldots,C\}$ **do**
      for $j \in \{1,\ldots,|\mathcal{D}_i|\}$ **do**
        $\mathbf{b}_i$ = optimizer$(\mathbf{I}_i,\mathbf{b}_i,\mathcal{D}_{ij},\gamma)$;
        $\Gamma_i = \Gamma_i + \mu\cdot(\gamma+1)$;
  Stage 3: Competition: Compare and mark the island with the best fitness.
  $f^\star = \infty$;
  for $i \in \{1,\ldots,I_{\max}\}$ **do**
    if $f(\mathbf{b_i}) < f^\star$ **then**
      $I_{\mathrm{best}} = i$; $x^\star = \mathbf{b}_i$; $f^\star = f(x^\star)$;
  for $i \in \{1,\ldots,I_{\max}\}$ **do**
    if $i = I_{\mathrm{best}}$ **then**
      $w_i = w_i + 1$;
  $\mathcal{W} = \{\}$;
  for $i \in \{1,\ldots,I_{\max}\}$ **do**
    if $w_i < w_{\min}$ **and** $t > \tau_{\mathrm{k}}$ **then**
      $\mathcal{K} = \mathcal{K} \cup i$;
    if $s =$ "warn" **then**
      if $w < w_{\min}$ **and** $t > \tau_{\mathrm{w}}$ **then**
        $\mathcal{W} = \mathcal{W} \cup i$;
  $t = t + 1$;
  Stage 4: Collaboration: Injecting the best individual from *Winner* island (I_{best}) into all the other islands.

B. The Warn-then-Kill Strategy

The warn-then-kill strategy is more lenient one, and grants a second chance to the poor performing islands to improve their contribution towards the improvement of the overall objective value before terminating them. The overall optimization process happens in the following two phases:

1) The algorithm calculates and updates contributions of each of the islands by measuring their win and loss scores after each tournament. In this scenario, the islands that have a zero win score are issued a warning and their evolution time is reduced.
2) The algorithm rechecks the contributions of all the islands by monitoring their win counts. If there is no substantial improvement of the weaker islands for τ_{w} tournaments, the algorithm terminates them.

Algorithm 2 shows the details of MIC^{4}algorithm that includes both the kill and warn-then-kill strategies. The algorithm starts by initializing all islands in a round-robin fashion. This is labeled as the initialization stage (Alg. 2, lines 2-5). The function `rand` takes the ith island ($\mathbf{I}_i$) and randomly initializes it with μ random solution within the upper ($\overline{\mathbf{x}}$) and the lower ($\underline{\mathbf{x}}$) bound limits. The evolution stage is very similar to the MIC^{3}with the exception of including the required mechanism to deal with stagnant islands. The loop on line 10 forms the main evolutionary loop. While the sum of fitness evaluations used by all islands (Γ_i) is less than the maximum available budget ($\Gamma_{\max}$), the algorithm evolves the islands independently. The sets $\mathcal{W}$ and $\mathcal{S}$, which are initialized on line 7 are used to implement the kill and the warn-then-kill strategies. The set $\mathcal{K}$ track the islands should be excluded from the evolutionary process, and the set $\mathcal{S}$ tracks the islands that were issued a warning.

On line 11, the algorithm iterates over all the islands excluding the ones which are in the kill set ($\mathcal{S}$). In the case of warn-then-kill stage, the algorithm penalizes the islands to which a warning is issued by reducing the number of cycles that it is optimized. This is done on lines 12 to 15. The variable C is the maximum number of times that an islands is optimized. For the penalized islands this is initialized to C_{pen}, and to $C_{\max}$ for the remaining islands. It is clear that $C_{\max} > C_{\text{pen}}$.

It was previously mentioned that a CC framework is used to optimize each island. On line 17, the subcomponents of an islands are iterated over and optimized using the `optimizer` function. For the purposes of this study, we have adopted G3-PCX [39] as the subcomponent optimizer. It should be noted that $\mathcal{D}_i$ contains the decomposition of the ith island, and $\mathcal{D}_{ij}$ is the jth subcomponent of the ith island. The `optimizer` function evolves the jth subcomponent of the ith island for γ iterations. The vector $\mathbf{b}_i$ is the current best solution of the ith island, which is also used as a *context vector* in the evolutionary framework to form complete solutions. It is clear that because the kill and the warn sets are initialized to an empty set, all islands will be optimized at least $C_{\max}$ times. Then, in Stage 3, all islands are examined to find the best performing island (Algorithm 2, lines 22-24). The variable w_i counts the number of times that an island wins a tournament.

Next, the kill or the warn-then-kill strategies are applied (Alg. 2, lines 28-34). The parameter τ_{k} on line 30 is a threshold beyond which an islands should be terminated (killed) indefinitely if its win count (w_i) is less than a predefined value ($w_{\min}$). Therefore, if the tournament index t is larger than τ_{k} and the win count of the ith island is less than $w_{\min}$, then the ith island is added into the kill set. Alternatively, if the active strategy is warn-then-kill, all the islands that satisfy $w_i < w_{\min}$ at any time when the tournament count is in the range $(\tau_{\text{w}}, \tau_{\text{k}})$ will be placed in the warn set ($\mathcal{W}$). The parameter τ_{w} is the warning threshold and should satisfy the following condition: $\tau_{\text{w}} > \tau_{\text{k}}$. It should be noted that as soon as an island wins a tournament, it will be removed form the warn set because at each tournament the warn set is initialized to an empty set (line 28). Finally, in stage 4, the best solution of the best performing island is injected into other islands and the main loop is continued.

In order to give an equal chance to all the islands to win a tournament, the parameter τ_{k} should be an integer multiple of $I_{\max}$. By the same token, the variable τ_{w} should also be initialized to an integer multiple of $I_{\max}$ such that $\tau_{\text{k}} < \tau_{\text{w}}$. In this paper, we initialized τ_{k} to $I_{\max}$ in the case of the kill strategy. When the strategy is warn-then-kill, the parameter τ_{w} is set to $I_{\max}$, and $\tau_{\text{k}} = 2\tau_{\text{w}}$. A possible extension of MIC^{3}is to adaptively change these variables based on the overall performance of the islands over the course of optimization. However, a further investigation this approach is beyond the scope of this paper. In the next section, we evaluate and compare the performance of MIC^{3}and MIC^{4}on a set of well-known benchmark functions.

IV. Experiments and Results

In this section, we evaluate the performance of the proposed contribution based algorithm (MIC^4) on several well-known benchmark problems of 100 dimensions. These benchmark problems have been selected by considering their level of difficulty and separability, and their modality (Table II). We first benchmark the performance of the two strategies of MIC^4, and then provide further analysis about the islands that have been most dominant during the course of evolution.

A. Parameter Settings

The generalized generation gap with parent centric crossover (G3-PCX) evolutionary algorithm [39] was employed for optimizing the sub-populations of islands in MIC^3. In the current implementation, the G3-PCX employs a mating pool size of 2 offspring, a family size of 2 parents, and a generation gap model for selecting the sub-populations in the cooperative coevolution framework, and a population for local search. This parameter set-up has demonstrated good performance in solving global optimization problems [39]. In MIC^4, $C_{\max}$ is set to 25, and $C_{\text{pen}} = \frac{1}{5}C_{\max} = 5$. In MIC^3, the parameter C is set to 25. The parameter γ of both MIC^{3}and MIC^{4}is set to 1.

The five different uniform problem decomposition strategies that were implemented as islands of MIC^{3}and MIC^{4}are shown in Table IV. The column that marks "Error" in Table II is the desired accuracy of solutions, which determines one of the termination criteria before reaching the maximum number of function evaluations ($\Gamma_{\max}$) fixed at 1.5×10^6. A run is successful only when the algorithm halts with the minimum error. A total of 25 independent runs were conducted with different random initializations in all of the respective sub-populations of the islands. In each case, the mean fitness value (error), the corresponding function evaluations and the success rates are been reported. The results are presented and discussed in Section IV-B.

B. Results

In this section, we analyze the performance of MIC^{4}in terms of function evaluations and solution quality. Table V contains

TABLE II
PROBLEM DEFINITIONS

Fun.	Name	Optimum	Range	Unimodal	Separable	Error
f_1	Ellipsoid	0	[-5,5]	Yes	Yes	1E-20
f_2	Shifted Sphere	-450	[-100,100]	Yes	Yes	1E-10
f_3	Schwefel's Problem 1.2	0	[-5,5]	Yes	Yes	1E-20
f_4	Rosenbrock	0	[-5,5]	No	No	1E-20
f_5	Shifted Rosenbrock	390	[-100,100]	No	No	1E-10
f_6	Rastrigin	0	[-5,5]	No	Yes	1E-20
f_7	Shifted Rastrigin	-330	[-5,5]	No	Yes	1E-10
f_8	Shifted Griewank	-180	[-600,600]	No	No	1E-10

TABLE III
SUMMARY OF THE ALGORITHMS

Algorithm	Description
MIC^3 [22]	Multi-Island Competitive Cooperative Coevolution
MIC^4(Kill)	Contribution based variant of MIC^{3}that greedily eliminates the weaker islands at the initial stage.
MIC^4(Warn-then-Kill)	Contribution based variant of MIC^{3}that triggers a warning to the poor performing islands at the initial stage and provides a second chance to improve. In the next stage, the islands are only eliminated if there is no substantial improvement in their performance.

the experimental results for comparing the performance of the proposed MIC^{4}algorithm against MIC^3 [22].

We first evaluate the performance of MIC^{4}with the Kill strategy against MIC^3. According to Table V, it can be observed that MIC^4(Kill) has a better performance on 7 out of 8 functions. It is clear that MIC^4(Kill) has managed to find better solutions using less computational resources. This is clearly evident on all the separable and unimodal functions (f_1-f_3), which shows that a considerable number of function evaluations has been saved. On closer inspection, we can see that MIC^4(Kill) also outperformed MIC^{3}and found better solutions on both instances of the non-separable and multi-modal functions (f_4 and f_5). Another observation is that MIC^4(Kill) has produced slightly better solutions than MIC^{3}on instances of the Rastrigin function (f_6 and f_7). However, MIC^4(Kill) has performed slightly worse than MIC^{3}on the multi-modal and non-separable f_8. This suggests that this greedy early termination mechanism may not always be feasible.

A similar trend exist when comparing MIC^4(Warn-then-Kill) with MIC^3, as shown in Table V. The experimental results reveal that while converging to high quality solutions, MIC^4(Warn-then-Kill) has managed to save more function evaluations than the traditional MIC^{3}on almost all of the benchmark functions with the exception of the Rastrigin function (f_6). On functions f_1-f_3 and f_8, it can be observed that MIC^4(Warn-then-Kill) has managed to shorten the optimization time by a factor of approximately 30%-60%. This suggests that a less greedy elimination strategy can further improve the solutions quality of MIC^3. Overall, we can see that both elimination strategies of MIC^{4}improve the performance of MIC^3, but MIC^4(Warn-then-Kill) performs better than MIC^3(Kill) on 6 out of 8 benchmark functions.

TABLE IV
ISLAND IMPLEMENTATIONS OF MIC^3. A DECOMPOSITION OF THE FORM $x \times y$ HAS x COMPONENTS OF SIZE y.

Island	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	$\mathcal{D}_4$	$\mathcal{D}_5$
Problem Decomposition	20×5	50×2	10×10	4×25	5×20

TABLE V
COMPARISON OF MIC^{4}VERSIONS WITH TRADITIONAL MIC^3 [22]

Fun.	Alg.	FE	Error	Success/25
f_1	MIC^3	393024	1.50E-21 $\pm$ 2.96E-21	25
	MIC^4(Kill)	323094	3.73E-21 $\pm$ 2.43E-21	25
	MIC^4(Warn-then-Kill)	262026	3.58E-21 $\pm$ 2.17E-21	25
f_2	MIC^3	283554	-450 $\pm$ 2.28E-11	25
	MIC^4(Kill)	201960	-450 $\pm$ 2.01E-11	25
	MIC^4(Warn-then-Kill)	188826	-450 $\pm$ 3.39E-11	25
f_3	MIC^3	398898	2.21E-21 $\pm$ 2.64E-21	25
	MIC^4(Kill)	293611	2.39E-21 $\pm$ 2.79E-21	25
	MIC^4(Warn-then-Kill)	276582	4.32E-21 $\pm$ 4.10E-21	25
f_4	MIC^3	1508550	79.53 $\pm$ 8.19	0
	MIC^4(Kill)	1508100	76.18 $\pm$ 43.35	0
	MIC^4(Warn-then-Kill)	1504500	73.74 $\pm$ 46.75	0
f_5	MIC^3	1508550	502.78 $\pm$ 28.83	0
	MIC^4(Kill)	1507111	481.26 $\pm$ 65.72	0
	MIC^4(Warn-then-Kill)	1504718	461.45 $\pm$ 66.79	0
f_6	MIC^3	1508550	0.25E+01 $\pm$ 0.19E+01	0
	MIC^4(Kill)	1508100	9.90E-01 $\pm$ 2.60E-01	1
	MIC^4(Warn-then-Kill)	1514400	9.90E-01 $\pm$ 1.00E-03	1
f_7	MIC^3	1508550	-216.06 $\pm$ 17.74	0
	MIC^4(Kill)	1505172	-219.52 $\pm$ 15.99	0
	MIC^4(Warn-then-Kill)	1505345	-220.03 $\pm$ 15.29	0
f_8	MIC^3	1001250	-179.99 $\pm$ 2.25E-02	9
	MIC^4(Kill)	1026852	-179.99 $\pm$ 4.66E-02	9
	MIC^4(Warn-then-Kill)	440226	-179.99 $\pm$ 3.95E-03	19

In addition to benchmarking the overall performance, it is also important to analyze the contribution of the two strategies of MIC^{4}during the course of evolution. We are interested in finding why the contribution based approach improves MIC^3, and how the minor difference between the two strategies resulted in a major difference in their performance. For the sake of brevity, we limit our analysis to MIC^4(Warn-then-Kill) algorithm and focus on the islands that have been most dominant during the course of optimization. Note that the elimination in MIC^4(Warn-then-Kill) happens right at the end of the *Kill* phase. Figure 1 shows the performance (win count) of each island during the two phases of MIC^4(Warn-then-Kill) algorithm just before the elimination occurs.

Due to space constraints, the analysis is done on four functions (f_1, f_4, f_6, and f_8). However, these contain both separable and nonseparable functions as well as unimodal and multi-modal functions. For each of the aforementioned functions, the average win counts over 25 runs for each of five islands are recorded (Figure 1). The plots on the left correspond to the warning phase and the ones on the right correspond to the kill phase of MIC^4. An important observation is that even though few of the poor performing islands are deprived of equal evolution time during the *warn*

phase, they have shown to be beneficial in the *kill phase* and has shown to improve the solution quality for functions f_4a (island 4) and f_8a (islands 1 and 5) Another observation is that the stronger islands in the early stage may not necessarily be dominant in the future stages of evolution e.g. Island 1 of f_4b, and Islands 1, 4, and 5 of f_6b. This indicates that it is indeed beneficial to preserve the weaker islands to help converge to better quality solutions in the later stages. The results and analysis from Table V and Figure 1 justify the superior performance of MIC^4(Warn-then-Kill) over the greedy MIC^4(Kill) as well the traditional MIC^3.

C. Discussion

The results in general have been very promising that shows that a small alteration in island based algorithm can be providing significant improvements in the results. The contribution based strategy is analogous to a class full of students where the teachers give more emphasis to the strong students in order to get higher class average scores. This from perspective of education, this would be negative as the weaker students performance is important, but in our case of evolution and the islands, elimination saves computation time. In the analogy, the question that teachers would find difficult to answer is weather to eliminate the weaker students or give them amnesty for a while where they can improve their scores.

The results also review the number of times an island wins and loses during different phases of evolution. This sets the basis for making decisions based on contributions, whether to eliminate or reduce the evolution time. We are interested to find if the weak islands contribute, or if retaining them is helpful during the later stages of the evolutionary process. The solutions in the weaker islands can be helpful in creating diverse solutions at later stages or evolution. The weaker islands have different problem decomposition strategies that can appeal in the later stages - when the nature of the problem changes in terms of separability. For instance, if an island appeals to fully separable functions, then it will not be helpful if the function is partially separable. However, what if the function definition changes with time, i.e. if the function becomes fully separable at the later stage of evolution, then the fully separable island would be helpful. This seems to be the case in the warn - then - kill strategy as it has shown to be better than the direct kill strategy in most of the problems.

V. Conclusion

In this paper, we proposed two contribution based MIC^{4}strategies which were implemented as 'warn then kill' and 'direct kill' strategy to eliminate the islands that have weak performance in MIC^3. This was implemented by splitting the evolution time according to the contributions of each of the different islands.

The results and analysis have shown that the two contribution based strategies have proven to be advantageous during the optimization process. Furthermore, another important observation was that the warn then kill strategy further improves the overall optimization performance than the direct kill strategy. The warn then kill strategy allows the weaker islands to evolve that has shown to be beneficial in promoting diversity at the later stages of optimization.

In future work, it would be beneficial to apply the proposed approach to multi-objective optimization problems and combinatorial optimization problems.

References

[1] M. A. Potter and K. A. De Jong, "A cooperative coevolutionary approach to function optimization," in *Parallel problem solving from naturePPSN III*. Springer, 1994, pp. 249–257.

[2] ——, "Cooperative coevolution: An architecture for evolving coadapted subcomponents," *Evolutionary computation*, vol. 8, no. 1, pp. 1–29, 2000.

[3] R. Salomon, "Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. a survey of some theoretical and practical aspects of genetic algorithms," *BioSystems*, vol. 39, no. 3, pp. 263–278, 1996.

[4] R. Chandra, M. Frean, and M. Zhang, "On the issue of separability for problem decomposition in cooperative neuro-evolution," *Neurocomputing*, vol. 87, pp. 33–40, 2012.

[5] M. Omidvar, X. Li, Y. Mei, and X. Yao, "Cooperative co-evolution with differential grouping for large scale optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 18, no. 3, pp. 378–393, June 2014.

[6] X. Li and X. Yao, "Cooperatively coevolving particle swarms for large scale optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 16, no. 2, pp. 210–224, 2012.

[7] N. García-Pedrajas and D. Ortiz-Boyer, "A cooperative constructive method for neural networks for pattern recognition," *Pattern Recognition*, vol. 40, no. 1, pp. 80–98, 2007.

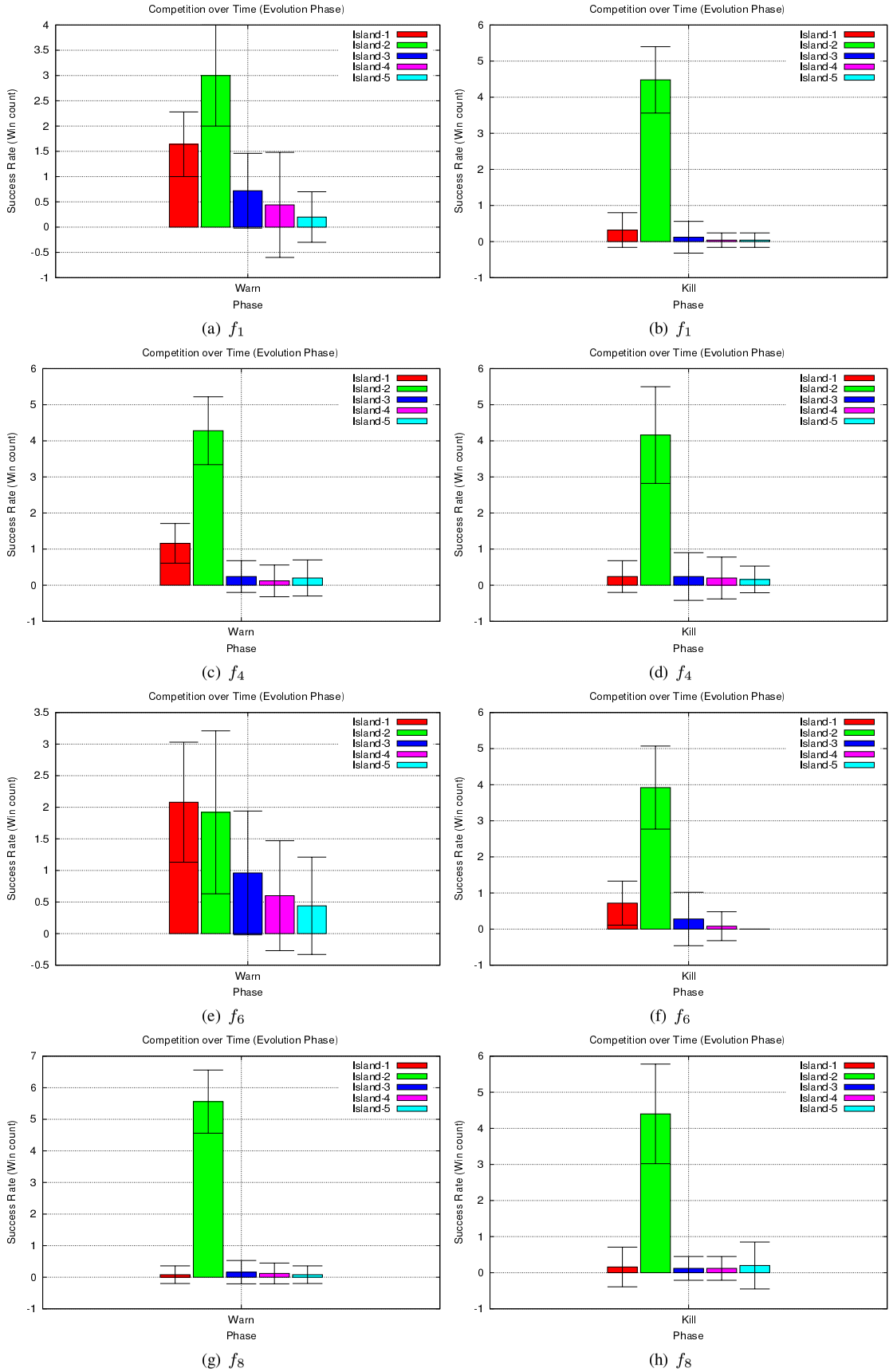
[8] R. Chandra and M. Zhang, "Cooperative coevolution of elman recurrent neural networks for chaotic time series prediction," *Neurocomputing*, vol. 86, pp. 116–123, 2012.

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

[10] X. Li, K. Tang, M. N. Omidvar, Z. Yang, and K. Qin, "Benchmark functions for the CEC'2013 special session and competition on large-scale global optimization," RMIT University, Melbourne, Australia, Tech. Rep., 2013.

[11] 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, USTC, China, Tech. Rep., 2009.

[12] Y.-j. Shi, H.-f. Teng, and Z.-q. Li, "Cooperative coevolutionary differential evolution for function optimiza-



(a) f_1 (b) f_1

(c) f_4 (d) f_4

(e) f_6 (f) f_6

(g) f_8 (h) f_8

Fig. 1. Competition over Time for the success rate (Win count) of different islands on selected functions . The performance of each of the Islands is monitored at two different phases of Warn-then-Kill. The mean win counts and standard deviation for each island over 25 independent runs are plotted.

tion," in *Advances in natural computation*. Springer, 2005, pp. 1080–1088.

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

[14] ——, "Multilevel cooperative coevolution for large scale optimization," in *IEEE Congress on Evolutionary Computation, CEC 2008.* IEEE, 2008, pp. 1663–1670.

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

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

[17] Y. Mei, M. N. Omidvar, X. Li, and X. Yao, "A competitive divide-and-conquer algorithm for unconstrained large-scale black-box optimization," *ACM Transactions on Mathematical Software (TOMS)*, 2016.

[18] M. N. Omidvar, X. Li, and X. Yao, "Smart use of computational resources based on contribution for cooperative co-evolutionary algorithms," in *Proc. of Genetic and Evolutionary Computation Conference*. ACM, 2011, pp. 1115–1122.

[19] B. Kazimipour, M. N. Omidvar, X. Li, and A. Qin, "A sensitivity analysis of contribution-based cooperative co-evolutionary algorithms."

[20] R. Chandra, "Competition and collaboration in cooperative coevolution of Elman recurrent neural networks for time-series prediction," *Neural Networks and Learning Systems, IEEE Transactions on*, p. In Press, 2015.

[21] R. Chandra and K. Bali, "Competitive two island cooperative coevolution for real parameter global optimisation," in *IEEE Congress on Evolutionary Computation*, Sendai, Japan, May 2015, pp. 93–100.

[22] K. K. Bali and R. Chandra, "Multi-island competitive cooperative coevolution for real parameter global optimization," in *Neural Information Processing*. Springer, 2015, pp. 127–136.

[23] ——, "Scaling up multi-island competitive cooperative coevolution for real parameter global optimisation," in *AI 2015: Advances in Artificial Intelligence*. Springer, 2015, pp. 34–48.

[24] W. Michiels and S.-I. Niculescu, *Stability and Stabilization of Time-Delay Systems (Advances in Design and Control)*. Society for Industrial and Applied Mathematics, 2007.

[25] W. Li and L. Wang, "A competitive-cooperative coevolutionary optimization algorithm based on cloud model," in *Advanced Computational Intelligence (IWACI), 2011 Fourth International Workshop on*. IEEE, 2011, pp. 662–669.

[26] M. A. Nowak, "Five rules for the evolution of cooperation," *science*, vol. 314, no. 5805, pp. 1560–1563, 2006.

[27] C. D. Rosin and R. K. Belew, "New methods for competitive coevolution," *Evolutionary Computation*, vol. 5, no. 1, pp. 1–29, 1997.

[28] G. S. Nitschke and L. H. Langenhoven, "Neuro-evolution for competitive co-evolution of biologically canonical predator and prey behaviors," in *Nature and Biologically Inspired Computing (NaBIC), 2010 Second World Congress on*. IEEE, 2010, pp. 546–553.

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

[30] C. Scheepers and A. P. Engelbrecht, "Competitive coevolutionary training of simple soccer agents from zero knowledge," in *Evolutionary Computation (CEC), 2014 IEEE Congress on*. IEEE, 2014, pp. 1210–1217.

[31] N. Franken and A. P. Engelbrecht, "Particle swarm optimization approaches to coevolve strategies for the iterated prisoner's dilemma," *Evolutionary Computation, IEEE Transactions on*, vol. 9, no. 6, pp. 562–579, 2005.

[32] ——, "Evolving intelligent game-playing agents," *South African Computer Journal*, no. 32, pp. p–44, 2004.

[33] K. Chellapilla and D. B. Fogel, "Evolving an expert checkers playing program without using human expertise," *Evolutionary Computation, IEEE Transactions on*, vol. 5, no. 4, pp. 422–428, 2001.

[34] D. Whitley, S. Rana, and R. B. Heckendorn, "Island model genetic algorithms and linearly separable problems," in *Evolutionary computing*. Springer, 1997, pp. 109–125.

[35] J. Cohoon, S. Hegde, W. Martin, and D. Richards, "Punctuated equilibria: a parallel genetic algorithm," in *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*. L. Erlbaum Associates Inc., 1987, pp. 148–154.

[36] E. CANTU-PAZ, "Topologies, migration rates, and multi-population parallel genetic algorithms," in *Proc. Genetic and Evolutionary Computation Conference (GECCO'99)*, vol. 1, 1999, pp. 91–98.

[37] S. M. Gustafson, "An analysis of diversity in genetic programming," Ph.D. dissertation, University of Nottingham, 2004.

[38] K. K. Bali, R. Chandra, and M. N. Omidvar, "Competitive island-based cooperative coevolution for efficient optimization of large-scale fully-separable continuous functions," in *Neural Information Processing*. Springer, 2015, pp. 137–147.

[39] K. Deb, A. Anand, and D. Joshi, "A computationally efficient evolutionary algorithm for real-parameter optimization," *Evolutionary computation*, vol. 10, no. 4, pp. 371–395, 2002.