A Parallel Cooperative Coevolutionary SMPSO Algorithm for Multi-objective Optimization

Arash Atashpendar
SnT, University of Luxembourg, Luxembourg
arash.atashpendar@uni.lu

Bernabé Dorronsoro
University of Cádiz, Spain
bernabe.dorronsoro@uca.es

Grégoire Danoy and Pascal Bouvry
University of Luxembourg, Luxembourg
{gregoire.danoy; pascal.bouvry}@uni.lu

***Abstract*—This paper presents a new parallel multi-objective cooperative coevolutionary variant of the Speed-constrained Multi-objective Particle Swarm Optimization (SMPSO) algorithm. SMPSO adopts a strategy for limiting the velocity of the particles that prevents them from having erratic movements. This characteristic provides the algorithm with a high degree of reliability. The proposed approach, called CCSMPSO, is based on a new design and implementation of SMPSO in a cooperative coevolutionary (CC) framework. In such an architecture, the population is split into several subpopulations, which are in turn in charge of optimizing a subset of the global solution by using the original multi-objective algorithm. We compare our work with two different state-of-the-art multi-objective CC metaheuristics, namely CCNSGA-II and CCSPEA2, as well as the original SMPSO in order to demonstrate its effectiveness. Our experiments indicate that our proposed solution, CCSMPSO, offers significant computational speedups, a higher convergence speed and better and comparable results in terms of solution quality, when compared to the other two CC algorithms and SMPSO, respectively. Three different criteria are used for making the comparisons, namely the quality of the resulting approximation sets, average computational time and the convergence speed to the Pareto front.**

I. Introduction

Multi-objective optimization involves the optimization of several different functions at the same time. These functions must be in conflict, meaning that, given an optimal solution, improving the quality of one of the objectives leads to worsening at least one of the others. Therefore, the result of a multi-objective problem is not one single solution, but a set of non-dominated ones, so that none is better than the other for all the objectives. Evolutionary algorithms (EA) are highly suitable tools for tackling such difficult problems when exact approaches are not applicable. EAs have the advantage of providing highly accurate trade-off solutions in a reasonable time [3], [4].

In order to deal with large problems for which classical EAs fail, another nature inspired process was suggested to extend EAs, referred to as cooperative coevolutionary (CC) EAs [20]. In this approach, instead of having the algorithm evolve one homogeneous population of individuals representing a global solution, the population is broken down into several subpopulations that evolve specific parts of the global solution by cooperating during the evolution. Each subpopulation is in charge of evolving a subset of the decision variables using a standard EA and all subpopulations evaluate complete solutions through a cooperative exchange of individuals. A new and efficient CC framework for multi-objective optimization was presented in [6], [8]. In those works, three new cooperative coevolutionary EAs for multi-objective optimization were implemented using the designed CC framework, and their effectiveness with respect to their original counterparts was also demonstrated.

In this paper, we present a variant of a bio-inspired metaheuristic, called Particle Swarm Optimization (PSO), which mimics the social behavior of bird flocking or fish schooling [13]. We have redesigned an existing PSO-based algorithm using a cooperative coevolutionary (CC) paradigm within the aforementioned CC framework [6], [8], which is built on top of the jMetal framework [10]. The variant of the PSO used in our approach relies on an already enhanced version of another pre-existing multi-objective optimization PSO (MOPSO) [9]. This enhanced version, referred to as SMPSO [17], which stands for Speed-constrained Multi-objective PSO, solves a problem related to the velocity of the particles becoming too high and as a consequence resulting in erratic movements towards the upper and lower limits of the positions of the particles. This phenomenon is called "swarm explosion" [2]. SMPSO tackles this problem and offers a solution that prevents the velocity of the particles from becoming too high by using a velocity constriction mechanism. Since the first attempt at extending PSOs to multi-objective optimization by Moore and Chapman [16], there have been many other proposals. For a detailed survey of Multi-Objective Optimization PSOs (MOPSOs), we refer the reader to [21].

The contributions of this work are as follows. We propose a new multi-objective optimization CC algorithm, called CCSMPSO, and we will show that the inherent properties of the CC version, along with the speed constraining mechanism, lead to significant computational speedups, better and more accurate results, a high degree of reliability, and a faster convergence speed. We provide an analysis of the proposed solution to evaluate its performance and effectiveness using three criteria. First, CCSMPSO is compared with two CC variants of two well-known multi-objective genetic algorithms, namely NSGA-II [5] and SPEA2 [24], as well as the original non-CC version of SMPSO in order to evaluate and compare these techniques in terms of the quality of their solutions by making use of three quality indicators, namely additive epsilon, spread and hypervolume. The second and third parts of our analysis are concerned with the average computational time as well as the convergence speed of CCSMPSO with

respect to the other two CC EAs and the original SMPSO.

The remainder of this paper is structured as follows. Section II provides a brief survey of related work, followed by a description of the original SMPSO in Section III. Section IV gives a thorough presentation of our proposed approach, i.e., CCSMPSO, by describing the used methodology and the details of the CC version. In Section V, a brief description of the other two algorithms considered in this study is provided. Section VI presents the experimental settings, the obtained results and their analysis. Finally, Section VII summarizes our results and presents our conclusions.

II. Related Work

Although models for Cooperative Coevolutionary Evolutionary Algorithms (CCEA) were initially developed for single-objective optimization, more recently, several multi-objective versions have also been shown to be efficient.

A CCMOEA, as proposed in [19], evolves one population per objective function simultaneously. In [1], Barbosa et al. proposed a genetic algorithm with coevolving weighting factors (IGACW) based on two populations, one for the solutions and another for the weights. However, it suffered from a high degree of dependency on the user's solution ranking inputs.

More recently, a new competitive-cooperative coevolution model (COEA) was presented in [12], in which each species subpopulation competes at regular intervals, while the eventual winners cooperate to solve the problem and evolve towards better solutions. Three CCEAs for NSGA-II, SPEA2 and MOCell were also developed by Dorronsoro et al. and their effectiveness was demonstrated in [6], [8]. This model was later extended by García et al. [11] to improve its parallel scalability, reaching up to 128 subpopulations. The new design considers that the search spaces of the different islands overlap with one another, instead of the disjoint model that has been used classically.

Coevolution has also been considered for PSO algorithms. A team-based competitive coevolutionary PSO has been proposed by Scheepers and Engelbrecht in [22] for soccer agents training. The only existing versions of cooperative coevolutionary PSO algorithms were developed for single-objective optimization by van den Bergh et al. [23], and Li et al. [14], [15], as an attempt to cope with large scale optimization problems. The most recent one, CCPSO2 [15], provides highly competitive results for up to 2000-variable problems by using a mechanism that dynamically sets the groups of variables to optimize in every island, as well as a new update rule that relies on Gaussian and Cauchy distributions.

The proposed CCSMPSO uses the CCMOEA framework presented in [8], [6], which in turn, extends Potter and DeJong's [20] CCGA approach for multi-objective optimization and parallelism. The next section describes the original SMPSO and Section IV will be solely dedicated to dissecting the proposed solution's architecture and the adopted approach for its implementation.

III. Original SMPSO

This section describes the original SMPSO and the characteristic that sets it apart from its predecessor and that also makes it particularly suitable for the CCMOEA framework in which our solution is implemented.

A. Description of the Original SMPSO

Here we present the main property of the standard SMPSO, namely its velocity constriction mechanism, along with the pseudo-code of its algorithm.

Generally speaking, in a PSO algorithm, each potential solution to the problem at hand is called a *particle* and the population of individuals or solutions is referred to as the *swarm*. At its very core, a basic PSO updates the particle vector $\vec{x_i}$ in generation t according to the following formula:

$$\vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t) \ , \tag{1}$$

where the factor $\vec{v}_i(t)$ is known as velocity and is given by

$$\vec{v}_i(t) = w \cdot \vec{v}_i(t-1) + C_1 \cdot r_1 \cdot (\vec{x}_{pi} - \vec{x}_i) + C_2 \cdot r_2 \cdot (\vec{x}_{gi} - \vec{x}_i) \ , \tag{2}$$

where $\vec{x}_{pi}$ is the best known solution that $\vec{x}_i$ has viewed; $\vec{x}_{gi}$ represents the best particle, or the *leader*, that the entire swarm has viewed; w is the inertia weight of the particle that controls the trade-off between global and local experience; r_1 and r_2 represent two uniformly distributed random numbers in the range $[0, 1]$; and C_1 and C_2 are specific parameters that control the effect of the personal and global best particles.

B. Constrained Speed

In order to control the velocity of particles and prevent the kind of erratic movements mentioned previously, the SMPSO algorithm adopts a *constriction coefficient* instead of using upper and lower parameter values which limit the step size of the velocity. The constriction coefficient is obtained from the constriction factor χ, which was originally developed by Clerc and Kennedy in [2].

$$\chi = \frac{2}{2 - \varphi - \sqrt{\varphi^2 - 4\varphi}} \ , \tag{3}$$

where

$$\varphi = \begin{cases} C_1 + C_2 & \text{if } C_1 + C_2 > 4 \\ 1 & \text{if } C_1 + C_2 \leq 4 \end{cases} \ . \tag{4}$$

Moreover, another mechanism is used such that the accumulated velocity of each variable j (for each particle) is further bounded through the following *velocity constriction* equation:

$$v_{i,j}(t) = \begin{cases} \delta_j & \text{if } v_{i,j}(t) > \delta_j \\ -\delta_j & \text{if } v_{i,j}(t) \leq -\delta_j \\ v_{i,j}(t) & \text{otherwise} \end{cases} \ , \tag{5}$$

where

$$\delta_j = \frac{(upper\_limit_j - lower\_limit_j)}{2} \ . \tag{6}$$

Algorithm 1 Pseudocode of the original SMPSO

```
1: initializeSwarm()
2: evaluation()
3: initializeLeadersArchive()
4: generation = 0
5: while generation < maxGenerations do
6:     computeSpeed() // Eqs. 2-6
7:     updatePosition() // Eq. 1
8:     mutation() // Turbulence
9:     evaluation()
10:    updateLeadersArchive()
11:    updateParticlesMemory()
12:    generation++
13: end while
14: returnLeadersArchive()
```

Algorithm 2 Parallel CCMOEA Framework

```
1: t ← 1
2: ∃i ∈ [1, NP] :: setup(P^0, i) // Initialize every subpopulation
3: ∀i ∈ [1, NP] :: broadcast(P^0, i) // Share random local partial solutions in every subpopulation
4: ∃i ∈ [1, NP] :: evaluate(P^0, i) // Evaluate solutions in every subpopulation
5: while not stoppingCondition() do
6:     ∃i ∈ [1, NP] :: generation(P^t, i) // Perform one generation to evolve the population
7:     ∀i ∈ [1, NP] :: broadcast(P^t, i) // Share best local partial solutions in every subpopulation
8:     t ← t + 1
9: end while
10: mergeParetoFronts() //Merge the Pareto fronts found in the subpopulation into a single one
```

The velocity of the particles is calculated using Eq. (2). It is then multiplied by the constriction factor defined in Eq. (3) and the resulting value is constrained by Eq. (5).

This velocity constraining mechanism, its effectiveness when compared to its predecessor, i.e., OMOPSO, and the fact that particles effectively move through the search space without taking limit values, have been demonstrated in [17].

C. Pseudocode of the Original SMPSO Algorithm

Algorithm 1 shows the pseudocode of the original SMPSO. The algorithm starts by initializing the swarm, which includes the position, velocity, and p (individual best) of the particles (line 1). Then, all solutions are evaluated (line 2) and the leaders archive is initialized with the non-dominated solutions in the swarm (line 3). Afterwards, the main loop of the algorithm is executed for a maximum number of iterations (starting in line 5). The velocities and positions of the particles are computed first (lines 6 and 7) and a mutation operator (turbulence factor) is applied with a given probability (line 8). The resulting particles are evaluated (line 9) and both the particle's memory as well as the leaders archive are updated (lines 10 and 11). Finally, the algorithm returns the leaders archive as the approximation set (line 14).

Since it is possible for the leaders archive to become full, the crowding distance of NSGA-II is used to decide which particles must remain. The turbulence factor is implemented using the polynomial mutation operator [4]. In order to choose the p_{best} particle for applying Eq. (2), two solutions are taken from the leaders archive at random and the one that has the largest crowding distance to its nearest neighbors in the archive gets selected.

IV. Cooperative Coevolutionary SMPSO

The SMPSO algorithm, presented in the previous section, is redesigned to fit in the CCMOEA framework. The latter provides a quite generic interface for building different CC-MOEAs. This allows us to leverage the highly parallelizable nature of the framework, which comes from the decomposition of the population into several subpopulations. These subpopulations are then evolved by independent MOEAs, which in the case of CCSMPSO, are the standard SMPSO. The parallel implementation of the framework is designed for multi-core architectures. Each subpopulation is evolved using a separate core and a few synchronization points are implemented to allow a safe information exchange between the islands. Please note that CCSMPSO does not reproduce the exact same behavior as the sequential SMPSO algorithm as CCSMPSO involves an SMPSO that evolves in a highly reduced search space, because it only works on a few variables of the problem.

How CCSMPSO works and how it is incorporated in the CCMOEA are best explained using a pseudocode that describes the design and architecture of the parallel CCMOEA framework. SMPSO is redesigned in such a way that it conforms exactly to the CCMOEA algorithm described in Algorithm 2. In the notation used for describing the parallel and sequential parts of the algorithm, the symbol $\forall$ is used for indicating a sequential run and $\exists$ for indicating a parallel run.

The different NP populations/swarms are initialized in line 2 of Algorithm 2, which corresponds to lines 1-3 of the SMPSO in Algorithm 1. The initialization phase creates new subpopulations made of random partial solutions. Each subpopulation then shares N_s randomly selected partial solutions (line 3 of Algorithm 2). Next, every subpopulation evaluates its initial swarm members in parallel, as indicated in line 4 of Algorithm 2. After that, the algorithm enters its main loop (the cooperative loop) and iterates until its termination condition is met. In the case of CCSMPSO, this corresponds to the predicate: $evaluations >= maxEvaluations/islands$. In every iteration, the subpopulations perform one generation of SMPSO in parallel (line 6 in Algorithm 2 and lines 5-11 in Algorithm 1), and then they synchronize to publish their best local partial solutions (line 7 in Algorithm 2). Finally, a single Pareto front is built from the solutions in the local Pareto fronts in every subpopulation (line 10 in Algorithm 2). This final Pareto front, which would be collectively made from the final leaders archive of each island's SMPSO, will be the output of the CCSMPSO algorithm.

Fig. 1 further illustrates the block diagram in the parallel CCMOEA framework. White circles represent synchronization points and black circles are bifurcation states (i.e., several parallel threads are created and run). Arrows indicate the execution sequence order, and the black square denotes the end state. Finally, NP refers to the number of subpopulations.

The following section describes the algorithms that have been used for comparison and benchmarking purposes in our work.

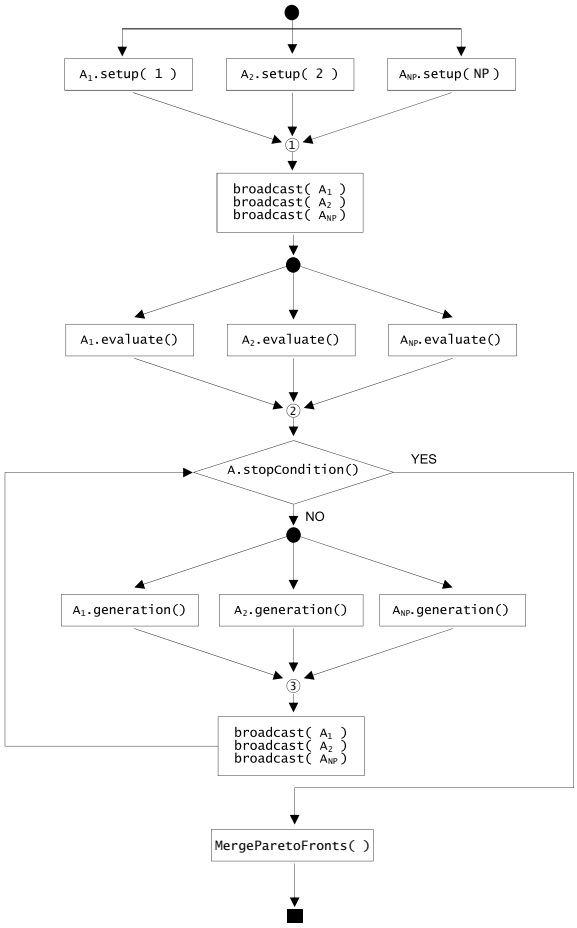

Fig. 1. Design of the parallel cooperative coevolutionary multi-objective CCMOEA framework.

V. Evaluated Algorithms

This section briefly describes the CC versions of the two algorithms that have been used for evaluating the behavior of the proposed CCSMPSO.

The NSGA-II algorithm, proposed in [5], is a genetic algorithm in which a new population is obtained from the original one by applying the typical genetic operators, i.e., selection, crossover and mutation. Then the individuals of the two populations (i.e., the new and old ones) get sorted according to their rank and the best solutions are selected to create a new population. A density estimation based on measuring the crowding distance to the surrounding individuals that belong to the same rank is used to get the most promising solutions in case some individuals with the same rank have to be selected.

In the SPEA2 algorithm, proposed in [24], each individual has a fitness value that is the sum of its strength raw fitness and a density estimation. The selection, crossover and mutation operators are applied to fill an archive of individuals. Then the non-dominated individuals belonging to both the original population and the archive are copied into a new population. In case the number of non-dominated individuals is greater than the population size, a truncation operator based on calculating the distance to the k-th nearest neighbor is used. This leads to the individuals having the minimum distance to any other individual getting chosen.

The performance of the CC versions of both NSGA-II and SPEA2 was shown to be highly competitive with respect to the original algorithms in a number of combinatorial and continuous problems, both on theoretical benchmarks and real-world problems [6], [7], [8], [18].

TABLE I
Parameterization & Experimental Setup

	CCSMPSO	SMPSO	CCNSGA-II	CCSPEA2
Max. evaluations	100,000			
Population initialization	Random			
Number of independent runs	100			
Mutation	Polynomial, $p_m = \frac{1}{\#variables}$			
Distribution index	20			
Number of subpopulations	4	-	4	
Number of cores	4	1	4	
Number of threads	1 per subpopulation	1	1 per subpopulation	
Number of shared solutions	20	-	20	
Population size	-		100	
Swarm size	100 per subpopulation	100	-	
Archive size	100 per subpopulation	100	-	100 per subpopulation
Selection	-		Binary Tournament	
Recombination	-		SBX, $p_c = 0.9$	
Perturbation index	0.5		-	
Hardware used in experiments	Intel(R) Core(TM) i5-4690 CPU @ 3.50Ghz, 4 cores, 8 GB RAM			

VI. Experiments & Results

This section first describes the experimentation settings and algorithm parameterizations that have been used in our experiments for assessing the searching capabilities of the evaluated algorithms. We then present and analyze the results obtained from our experiments.

A. Parameterization & Settings

A series of benchmark problems were chosen to evaluate the following four algorithms: CCSMPSO, SMPSO, CCNSGA-II and CCSPEA2. These problems include the ZDT (Zitzler-Deb-Thiele) and DTLZ (Deb-Thiele-Laumanns-Zitzler) test suites. The DTLZ problems have been used with their bi-objective formulation. In order to assess the performance of the algorithms, three quality indicators have been considered: Additive unary epsilon indicator ($I^1_\epsilon+$), spread (Δ) and hypervolume (HV). The first two indicators measure the convergence and the diversity of the resulting Pareto fronts, respectively, and the last one measures both convergence and diversity. The next subsections describe the parameter settings used for the algorithms and the adopted experimentation methodology.

1) Parameterization: A set of parameter settings has been chosen to guarantee a fair comparison among the evaluated algorithms. Both CCNSGA-II as well as CCSPEA2 use a population of 100 individuals. The size of the archive in CCSPEA2, SMPSO and CCSMPSO is also 100. SMPSO and CCSMPSO have been configured with 100 particles. SBX and polynomial mutation have been used as crossover and mutation operators, respectively. The distribution indices for both operators are $\eta_c = 20$ and $\eta_m = 20$, respectively. The crossover probability is $p_c = 0.9$ and the mutation probability is $p_m = 1/L$, where L represents the number of decision

variables. Selection is done by binary tournament. SMPSO and CCSMPSO use polynomial mutation. The adopted parameter settings are the same as the ones proposed for the original algorithms, and they are summarized in Table I.

2) Methodology: In order to assess the search capabilities of the algorithms, 100 independent runs of each experiment have been performed. The median, $\tilde{x}$, and interquartile range, IQR, have been computed as measures of location (or central tendency) and statistical dispersion, respectively. In addition, we performed the Wilcoxon test in order to assess statistical confidence in the comparison of CCSMPSO with respect to the other algorithms, with 95% confidence level. For the sake of a better understanding and an easier interpretation, the best result for each problem has been highlighted in **bold font**, while the results in dark gray and light gray background indicate statistically better and worse performance according to the Wilcoxon test (with respect to CCSMPSO), respectively. Those results without any background color indicate that there is no statistical difference with respect to the performance of CCSMPSO.

B. Results & Evaluation

Here we analyze the quality of the results obtained from our experiments, after 100,000 function evaluations. The evaluation of the obtained results is based on three criteria, namely (i) the *quality of the approximated Fronts*, (ii) *the computational time* and (iii) the *convergence speed*. Tables II to IV present the results of all the studied algorithms in terms of the previously mentioned statistical measurements, applied to the selected set of quality indicators.

TABLE II
MEDIAN AND INTERQUARTILE RANGE OF THE I_ϵ^1+ INDICATOR

	CCSMPSO	SMPSO	CCNSGA-II	CCSPEA2
ZDT1	$5.35e-03_{1.8e-04}$	$\mathbf{5.22e-03_{1.7e-04}}$	$7.30e-03_{9.9e-04}$	$6.84e-03_{6.3e-04}$
ZDT2	$5.35e-03_{1.9e-04}$	$\mathbf{5.19e-03_{1.6e-04}}$	$7.16e-03_{8.0e-04}$	$6.79e-03_{6.8e-04}$
ZDT3	$4.90e-03_{2.9e-04}$	$\mathbf{4.82e-03_{2.4e-04}}$	$5.50e-03_{9.0e-04}$	$7.19e-03_{1.3e-03}$
ZDT4	$5.56e-03_{1.9e-04}$	$\mathbf{5.31e-03_{2.3e-04}}$	$1.32e-01_{1.3e-01}$	$1.32e-01_{2.7e-01}$
ZDT6	$4.52e-03_{5.5e-04}$	$\mathbf{4.36e-03_{2.5e-04}}$	$5.48e-03_{4.2e-04}$	$5.77e-03_{5.6e-04}$
DTLZ1	$2.90e-03_{1.6e-04}$	$\mathbf{2.81e-03_{1.4e-04}}$	$3.78e-03_{2.8e-04}$	$3.91e-03_{4.7e-04}$
DTLZ2	$5.27e-03_{2.2e-04}$	$\mathbf{5.08e-03_{2.2e-04}}$	$7.13e-03_{7.3e-04}$	$6.87e-03_{6.9e-04}$
DTLZ3	$5.34e-03_{2.5e-04}$	$\mathbf{5.13e-03_{2.6e-04}}$	$9.29e-03_{2.2e-03}$	$1.36e-02_{6.1e-03}$
DTLZ4	$5.39e-03_{2.4e-04}$	$\mathbf{5.16e-03_{1.9e-04}}$	$7.08e-03_{1.1e-03}$	$6.87e-03_{9.9e-01}$
DTLZ5	$5.00e-03_{2.4e-04}$	$\mathbf{4.90e-03_{2.5e-04}}$	$6.61e-03_{8.9e-04}$	$6.33e-03_{7.1e-04}$
DTLZ6	$5.03e-03_{2.3e-04}$	$\mathbf{5.00e-03_{2.7e-04}}$	$6.10e-03_{5.3e-04}$	$5.96e-03_{5.6e-04}$
DTLZ7	$\mathbf{2.31e+00_{4.9e-07}}$	$2.31e+00_{1.0e-05}$	$2.31e+00_{1.9e-06}$	$2.31e+00_{1.4e-05}$

TABLE III
MEDIAN AND INTERQUARTILE RANGE OF THE HYPERVOLUME INDICATOR

	CCSMPSO	SMPSO	CCNSGA-II	CCSPEA2
ZDT1	$6.62e-01_{3.8e-05}$	$\mathbf{6.62e-01_{2.8e-05}}$	$6.62e-01_{1.4e-04}$	$6.62e-01_{7.8e-05}$
ZDT2	$\mathbf{3.29e-01_{3.6e-05}}$	$\mathbf{3.29e-01_{3.6e-05}}$	$3.28e-01_{1.1e-04}$	$3.29e-01_{9.9e-05}$
ZDT3	$5.16e-01_{2.9e-05}$	$\mathbf{5.16e-01_{2.2e-05}}$	$5.16e-01_{4.9e-05}$	$5.16e-01_{4.7e-05}$
ZDT4	$6.62e-01_{5.8e-05}$	$\mathbf{6.62e-01_{5.6e-05}}$	$4.95e-01_{1.7e-01}$	$4.94e-01_{3.2e-01}$
ZDT6	$\mathbf{4.01e-01_{5.6e-05}}$	$4.01e-01_{6.1e-05}$	$4.01e-01_{2.3e-04}$	$4.00e-01_{3.7e-04}$
DTLZ1	$\mathbf{4.95e-01_{3.8e-05}}$	$4.95e-01_{4.5e-05}$	$4.94e-01_{8.8e-04}$	$4.93e-01_{1.2e-03}$
DTLZ2	$\mathbf{2.11e-01_{5.1e-05}}$	$2.11e-01_{5.8e-05}$	$2.10e-01_{1.5e-04}$	$2.10e-01_{8.7e-05}$
DTLZ3	$\mathbf{2.11e-01_{5.1e-05}}$	$2.11e-01_{9.7e-05}$	$2.06e-01_{3.0e-03}$	$2.00e-01_{7.8e-03}$
DTLZ4	$2.11e-01_{5.9e-05}$	$\mathbf{2.11e-01_{4.4e-05}}$	$2.10e-01_{1.2e-04}$	$2.10e-01_{2.1e-01}$
DTLZ5	$\mathbf{2.12e-01_{5.2e-05}}$	$2.12e-01_{5.4e-05}$	$2.12e-01_{1.0e-04}$	$2.12e-01_{9.0e-05}$
DTLZ6	$2.12e-01_{5.3e-05}$	$\mathbf{2.12e-01_{3.9e-05}}$	$2.12e-01_{5.1e-05}$	$2.12e-01_{5.4e-05}$
DTLZ7	$3.09e-01_{4.7e-06}$	$\mathbf{3.09e-01_{4.1e-06}}$	$3.09e-01_{9.3e-06}$	$3.09e-01_{1.1e-05}$

1) Quality of the Approximated Fronts: In the case of the I_ϵ^1+ values of the resulting approximated fronts, as shown in Table II, the best solutions are obtained by SMPSO, that performs statistically better than its proposed CC version in 9 out of 12 problems, while it is outperformed by CCSMPSO for problems DTLZ3 and DTLZ7. However, the differences reported by the two algorithms in the table are less than 3.6% in all cases. Compared to the other CC algorithms included in the study, CCSMPSO outperforms them in all cases, with statistical confidence.

TABLE IV
MEDIAN AND INTERQUARTILE RANGE OF THE SPREAD INDICATOR

	CCSMPSO	SMPSO	CCNSGA-II	CCSPEA2
ZDT1	$7.31e-02_{1.3e-02}$	$\mathbf{6.19e-02_{1.7e-02}}$	$1.57e-01_{1.6e-02}$	$1.14e-01_{1.9e-02}$
ZDT2	$6.96e-02_{2.0e-02}$	$\mathbf{5.98e-02_{1.8e-02}}$	$1.57e-01_{1.6e-02}$	$1.12e-01_{1.9e-02}$
ZDT3	$\mathbf{7.01e-01_{7.3e-04}}$	$7.01e-01_{9.8e-04}$	$7.01e-01_{1.5e-03}$	$7.02e-01_{2.2e-03}$
ZDT4	$7.94e-02_{1.5e-02}$	$\mathbf{7.15e-02_{1.5e-02}}$	$3.07e-01_{1.7e-01}$	$2.79e-01_{2.7e-01}$
ZDT6	$6.76e-02_{1.9e-01}$	$\mathbf{5.88e-02_{7.2e-02}}$	$1.47e-01_{1.7e-02}$	$1.31e-01_{5.7e-02}$
DTLZ1	$5.44e-02_{1.5e-02}$	$\mathbf{4.39e-02_{1.3e-02}}$	$1.37e-01_{1.1e-02}$	$1.15e-01_{1.4e-02}$
DTLZ2	$\mathbf{1.13e-01_{1.9e-02}}$	$1.14e-01_{2.0e-02}$	$1.66e-01_{1.7e-02}$	$1.25e-01_{1.6e-02}$
DTLZ3	$\mathbf{1.07e-01_{2.0e-02}}$	$1.10e-01_{3.1e-02}$	$1.82e-01_{3.0e-02}$	$2.60e-01_{1.4e-01}$
DTLZ4	$1.02e-01_{1.9e-02}$	$\mathbf{1.02e-01_{1.8e-02}}$	$1.69e-01_{2.2e-02}$	$1.22e-01_{8.9e-01}$
DTLZ5	$1.17e-01_{2.2e-02}$	$\mathbf{1.17e-01_{1.9e-02}}$	$1.73e-01_{2.1e-02}$	$1.27e-01_{2.2e-02}$
DTLZ6	$\mathbf{9.99e-02_{2.5e-02}}$	$1.00e-01_{2.5e-02}$	$1.70e-01_{1.9e-02}$	$1.22e-01_{1.6e-02}$
DTLZ7	$\mathbf{7.77e-01_{9.5e-05}}$	$7.77e-01_{2.9e-04}$	$7.79e-01_{2.5e-03}$	$7.80e-01_{3.6e-03}$

As for the results of the HV value indicator, presented in Table III, we can see that the median values reported for both CCSMPSO and SMPSO are identical for all the compared problems. However, SMSPO offers in general a more reliable performance than CCSMPSO (its interquartile range values are lower). That is the reason behind the statistical differences between the algorithms: SMPSO is statistically better than CCSMPSO in 8 problems, and worse in DTLZ1 and DTLZ3 (no statistical difference was found for DTLZ2 and DTLZ5). Again, as in the case of the I_ϵ^1+ metric, CCSMPSO is statistically better than the other CC algorithms for all problems.

Finally, when it comes to the Δ indicator, shown in Table IV, SMPSO outperformed CCSMPSO in the five ZDT problems and DTLZ1, with statistical confidence, while it was worse for DTLZ3 and no statistical difference was found for the rest of the DTLZ benchmark problems. It is worth noting that CCSMPSO remains the dominant solver with respect to the other two CC algorithms, i.e., CCNSGA-II and CCSPEA2.

This degree of reliability and accuracy achieved by CCSMPSO can be traced back to the underlying SMPSO and its velocity constriction mechanism. This feature, which is responsible for maintaining and moving the particles through the search space without allowing them to have erratic movements, had already been proven to be effective in the original SMPSO algorithm and it is further leveraged by the decomposition of the population in the cooperative coevolutionary variant. As a consequence, although CCSMPSO, compared to SMPSO alone, does obtain the second best values in some instances, it emerges as the dominant solver with respect to its competing CC algorithms, i.e., CCNSGA-II and CCSPEA2 in all cases for I_ϵ^1+ and HV and in most cases for Δ, according to the Wilcoxon test used.

The box plots given in Figures 2 to 4 based on the I_ϵ^1+ and HV values, respectively, further illustrate the high degree of reliability of the CCSMPSO algorithm. The figures clearly

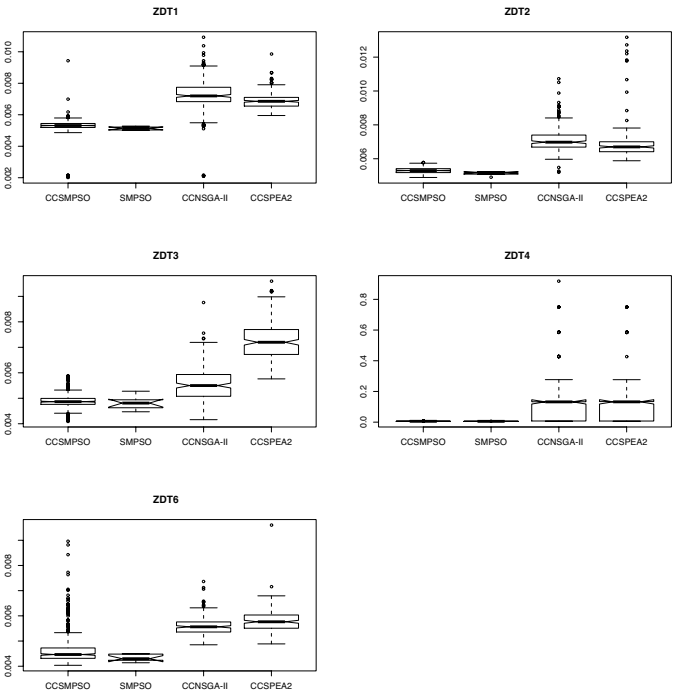

Fig. 2. I_ϵ^1+ box plot for ZDT benchmark problems (lower values are better)

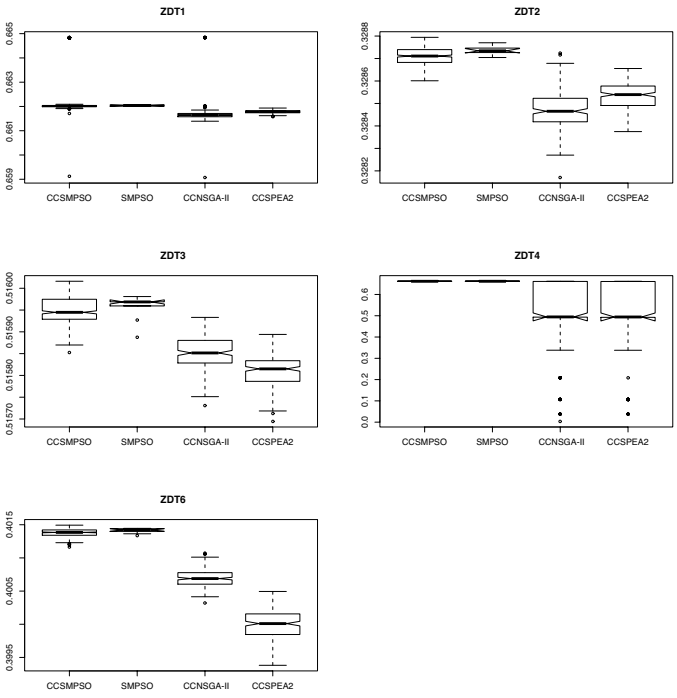

Fig. 3. HV box plot for ZDT benchmark problems (higher values are better)

show the superiority of the two SMPSO versions over the other compared algorithms. We can see the low degree of dispersion in the distribution of the CCSMPSO and SMPSO results (represented as smaller boxes with shorter whiskers), with respect to the other two algorithms. This is a reflection of their accuracy and high reliability.

Figure 4 shows how CCSMPSO offers statistically better performance than SMPSO (in terms of both hypervolume and I_ϵ^1+ indicators) despite its worse median value (shown in tables II and III). This is due to the outlier values reported by SMPSO in some of its runs, reducing its reliability.

2) Computational Speedup: We now analyze the performance of the parallel algorithms, i.e., the CC ones, with respect to the sequential SMPSO algorithm. Table V provides a comparison of the computational time of the proposed CCSMPSO with respect to the original sequential SMPSO as well as the other two CC algorithms, i.e., CCNSGA-II and CCSPEA2. Note that the computational times are averaged over the 100 independent runs, and four threads are evolved in parallel in the CC algorithms (one per subpopulation). The speedups achieved by CCSMPSO, as shown in Fig. 5, are computed as the time of the sequential SMPSO over the time of the parallel CCSMPSO, which range from 3.5 to 4.6 (i.e., super-linear speedup). Note that the obtained results indicate consistent improvements in performance, with the speedup factor having a lower bound of 3.5, in addition to three instances in which we have obtained super-linear speedups ranging from 4.11 to 4.67, i.e., ZDT2, ZDT4 and DTLZ6.

These speedups are achieved thanks to the fact that the total number of evaluations performed by the CCMOEAs is the same as for the sequential MOEAs: 100,000. In other words,

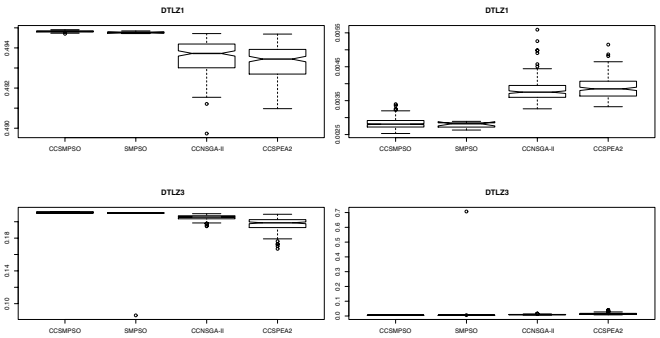

Fig. 4. HV (left) and I_ϵ^1+ (right) box plot for DTLZ benchmark problems

in the CCMOEAs, with the 4 subpopulations used in our parameterizations, every subpopulation performs only 25,000 evaluations and they evolve in parallel. The speedup can be further improved by:

1) Adopting our asynchronous implementation of the framework [18], which reports speedup improvements of up to 90% with respect to the equivalent synchronous algorithm.
2) Increasing the number of islands [6], [8], which implies a reduction in the number of evaluations performed in every island (proportional to the number of islands). However, the scalability of this solution is limited by the number of variables of the problem that is to be solved.

Furthermore, we can also see that CCSMPSO outperforms the other CCMOEAs, i.e., CCNSGA-II and CCSPEA2. For a more thorough analysis of the superiority of CCMOEAs

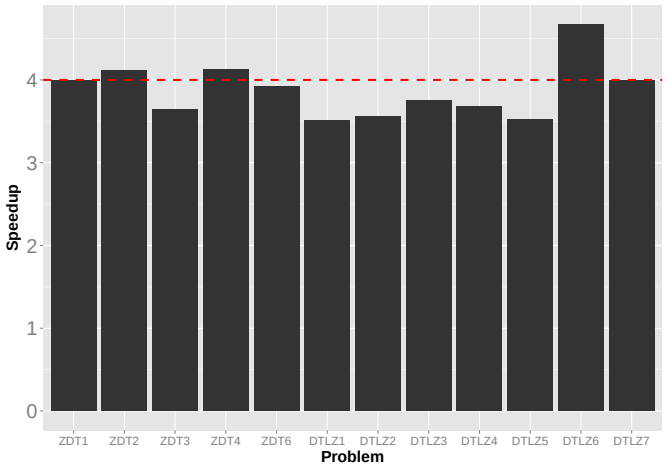

Fig. 5. Speedup factor computed as the time of the sequential SMPSO over the time of the parallel CCSMPSO. Values lying above the red dashed line indicate super-linear speedups.

and their super-linear speedup with respect to their sequential counterparts in continuous and combinatorial problems, the reader is encouraged to refer to [6], [7], [8], [18].

The timing results, shown in Table V, clearly indicate that CCSMPSO outperforms all the other three algorithms in terms of computational time. Apart from three instances where SMPSO yields the second best timing, CCNSGA-II emerges as the second best solver. The immediate conclusion that can be drawn here is that CCSMPSO is evidently the fastest algorithm among the four compared techniques. The time CCSMPSO takes is, in the best case, 26.11% the time of the second fastest compared algorithm, and 35% on average for all the studied problems.

TABLE V
COMPUTATIONAL TIME AVERAGED OVER 100 INDEPENDENT RUNS

	CCSMPSO	SMPSO	CCNSGA-II	CCSPEA2
ZDT1	**188.0ms**	751.0ms	465.0ms	1166.0ms
ZDT2	**189.0ms**	778.0ms	468.0ms	1126.0ms
ZDT3	**81.0ms**	295.0ms	456.0ms	1107.0ms
ZDT4	**106.0ms**	437.0ms	406.0ms	935.0ms
ZDT6	**164.0ms**	643.0ms	392.0ms	951.0ms
DTLZ1	**121.0ms**	424.0ms	393.0ms	875.0ms
DTLZ2	**154.0ms**	548.0ms	374.0ms	1260.0ms
DTLZ3	**106.0ms**	398.0ms	425.0ms	742.0ms
DTLZ4	**129.0ms**	474.0ms	387.0ms	1045.0ms
DTLZ5	**155.0ms**	545.0ms	364.0ms	1256.0ms
DTLZ6	**159.0ms**	743.0ms	381.0ms	1328.0ms
DTLZ7	**117.0ms**	467.0ms	441.0ms	1117.0ms

3) Convergence Speed: It is worth comparing the four algorithms in terms of the evolution of the HV value during their execution. Fig. 6 compares the evolution of the HV values when solving ZDT1 and ZDT2 with a granularity of 100 evaluations per data point.

We can observe that CCSMPSO is the first algorithm to reach the highest HV value of the Pareto front. It also converges to the true Pareto front faster than its sequential counterpart as well as the other two CC algorithms. Focusing

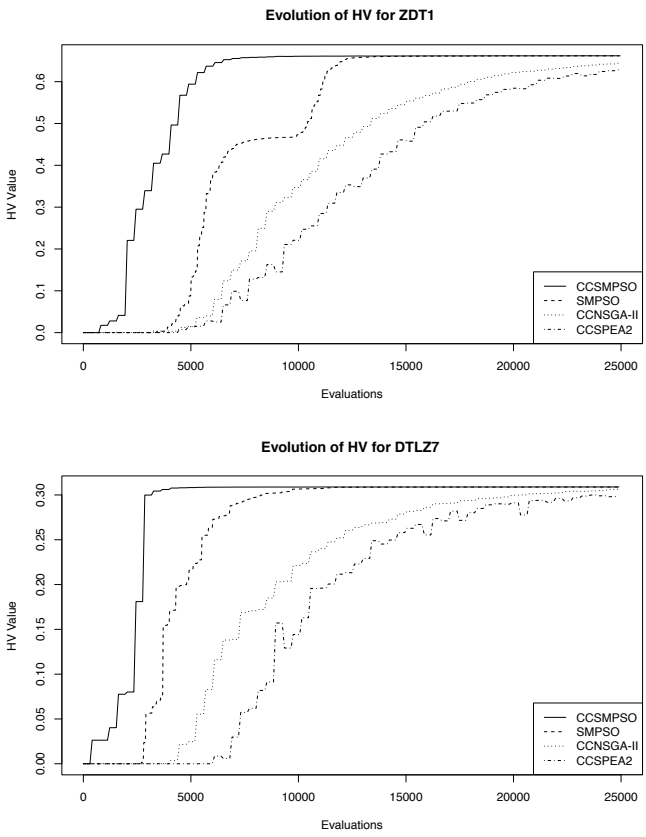

Fig. 6. Evolution of the HV through different generations for the problems ZDT1 and DTLZ7

solely on SMPSO and CCSMPSO, it is worth noting that in the case of ZDT1, there is a region in which SMPSO remains close to a fixed value, between 6,000 and 10,000 evaluations, but then it continues converging after a few thousand evaluations of stagnation. However, CCSMPSO does not exhibit this behavior and instead it rises steadily and converges towards the Pareto front in a faster and more reliable fashion. We would like to remark that CCSMPSO converges to the solution in about 5,000 evaluations in the plots shown, while SMPSO requires around 15,000 evaluations, and the other CC algorithms continue converging after 25,000.

VII. CONCLUSION

This paper presented a new cooperative coevolutionary (CC) optimization algorithm called CCSMPSO, which is a novel variant of the Speed-constrained Multi-Objective Particle Swarm Optimization (SMPSO) algorithm. This new cooperative coevolutionary design offers improvements in terms of performance due to the high degree of parallelization that is made possible by the CCMOEA framework, which has been used to implement CCSMPSO. The velocity constriction mechanism of the original SMPSO limits the maximum velocity of particles in order to enhance the search capability of the algorithm. CCSMPSO further leverages this property and yields a quick convergence speed and highly accurate

and reliable results on two well-known benchmark families, ZDT and DTLZ. CCSMPSO has been compared with the CC versions of two state-of-the-art genetic algorithms, which were reported to be highly competitive with the original algorithms, namely NSGA-II and SPEA2, along with the standard SMPSO.

In addition to yielding linear, and sometimes even super-linear computational speedups, CCSMPSO also provides comparable results in terms of solution quality when compared to the original SMPSO. Moreover, compared to the CC algorithms, experiments indicate that CCSMPSO is superior in terms of both the quality of the approximations to the Pareto front as well as the convergence speed.

As future work, we plan to study new designs whose scalability will not depend on the number of variables of the problem. Additionally, we intend to analyze the performance of our CCMOEAs for solving more complex benchmark problems.

ACKNOWLEDGEMENT

Bernabé Dorronsoro would like to acknowledge the Spanish MINECO for the support provided under contracts TIN2014-60844-R (the SAVANT project) and RYC-2013-13355.

REFERENCES

[1] Helio JC Barbosa and André MS Barreto. An interactive genetic algorithm with co-evolution of weights for multiobjective problems. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 203–210, 2001.

[2] Maurice Clerc and James Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58–73, 2002.

[3] C. A. Coello Coello, G. B. Lamont, and D. A. Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer, 2007. Second edition.

[4] Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons, 2001.

[5] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

[6] B. Dorronsoro, G. Danoy, A. J. Nebro, and P. Bouvry. Achieving super-linear performance in parallel multi-objective evolutionary algorithms by means of cooperative coevolution. *Computers & Operations Research*, 40(6):1552–1563, 2013.

[7] B. Dorronsoro, P. Ruiz, G. Danoy, Y. Pigné, and P. Bouvry. *Evolutionary Algorithms for Mobile Ad Hoc Networks*. Wiley/IEEE Computer Society, 2014.

[8] Bernabé Dorronsoro, Grégoire Danoy, Pascal Bouvry, and Antonio J Nebro. Multi-objective cooperative coevolutionary evolutionary algorithms for continuous and combinatorial optimization. In *Intelligent Decision Systems in Large-Scale Distributed Environments*, pages 49–74. Springer, 2011.

[9] Juan J Durillo, José García-Nieto, Antonio J Nebro, Carlos A Coello Coello, Francisco Luna, and Enrique Alba. Multi-objective particle swarm optimizers: An experimental comparison. In *Evolutionary Multi-Criterion Optimization*, pages 495–509. Springer, 2009.

[10] Juan J. Durillo and Antonio J. Nebro. jMetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42:760–771, 2011.

[11] Pablo García-Sánchez, Julio Ortega, Jesús González, Pedro A. Castillo, and Juan J. Merelo. *Proceedings of the 19th European Conference on Applications of Evolutionary Computation, EvoApplications, Part II*, chapter Addressing High Dimensional Multi-objective Optimization Problems by Coevolutionary Islands with Overlapping Search Spaces, pages 107–117. Springer International Publishing, 2016.

[12] Chi-Keong Goh and Kay Chen Tan. A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization. *Evolutionary Computation, IEEE Transactions on*, 13(1):103–127, 2009.

[13] James Kennedy and Russell C. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

[14] Xiaodong Li and Xin Yao. Tackling high dimensional nonseparable optimization problems by cooperatively coevolving particle swarms. In *IEEE Congress on Evolutionary Computation*, pages 1546–1553, 2009.

[15] Xiaodong Li and Xin Yao. Cooperatively coevolving particle swarms for large scale optimization. *IEEE Transactions on Evolutionary Computation*, 16(2):210–224, 2012.

[16] Jacqueline Moore and Richard Chapman. Application of particle swarm to multiobjective optimization. *Department of Computer Science and Software Engineering, Auburn University*, 32, 1999.

[17] Antonio J Nebro, JJ Durillo, Jose Garcia-Nieto, CA Coello Coello, Francisco Luna, and Enrique Alba. Smpso: A new pso-based metaheuristic for multi-objective optimization. In *Computational intelligence in miulti-criteria decision-making, 2009. mcdm'09. ieee symposium on*, pages 66–73. IEEE, 2009.

[18] S. S. Nielsen, B. Dorronsoro, G. Danoy, and P. Bouvry. Novel efficient asynchronous cooperative co-evolutionary multi-objective algorithms. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1–7, 2012.

[19] Ian C Parmee and Andrew H Watson. Preliminary airframe design using co-evolutionary multiobjective genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1657–1665, 1999.

[20] Mitchell A. Potter and Kenneth A. De Jong. A cooperative co-evolutionary approach to function optimization. In *Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature*, PPSN III, pages 249–257, London, UK, UK, 1994. Springer-Verlag.

[21] Margarita Reyes-Sierra and CA Coello Coello. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International journal of computational intelligence research*, 2(3):287–308, 2006.

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

[23] F. van den Bergh and A. Engelbrecht. A cooperative approach to particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):225–239, 2004.

[24] Eckart Zitzler, Marco Laumanns, Lothar Thiele, Eckart Zitzler, Eckart Zitzler, Lothar Thiele, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm, 2001.