- ORIGINAL ARTICLE -

Cooperative Coevolutionary Particle Swarms using Fuzzy Logic for Large Scale Optimization

Cúmulo de Partículas Coevolutivo Cooperativo usando Lógica Borrosa para la Optimización a Gran Escala

Fabiola Paz[1], Guillermo Leguizamón[1], and Efrén Mezura Montes[2]

[1]*LIDIC, University National of San Luis, Argentina*
fabypaz@fi.unju.edu.ar, legui@unsl.edu.ar
[2]*Artificial Intelligence Research Center, University of Veracruz, México*
emezura@uv.mx

Abstract

A cooperative coevolutionary framework can improve the performance of optimization algorithms on large-scale problems. In this paper, we propose a new Cooperative Coevolutionary algorithm to improve our preliminary work, FuzzyPSO2. This new proposal, called CCFPSO, uses the random grouping technique that changes the size of the subcomponents in each generation. Unlike FuzzyPSO2, CCFPSO's re-initialization of the variables, suggested by the fuzzy system, were performed on the particles with the worst fitness values. In addition, instead of updating the particles based on the global best particle, CCFPSO was updated considering the personal best particle and the neighborhood best particle. This proposal was tested on large-scale problems that resemble real-world problems (CEC2008, CEC2010), where the performance of CCFPSO was favorable in comparison with other state-of-the-art PSO versions, namely CCPSO2, SLPSO, and CSO. The experimental results indicate that using a Cooperative Coevolutionary PSO approach with a fuzzy logic system can improve results on high dimensionality problems (100 to 1000 variables).

Keywords: Adaptive inertia weight, Cooperative coevolutionary, fuzzy logic, Particle Swarm Optimization.

Resumen

Un marco coevolutivo cooperativo puede mejorar el rendimiento de los algoritmos de optimización en problemas a gran escala. En este trabajo, proponemos un nuevo algoritmo coevolutivo cooperativo para mejorar nuestro trabajo preliminar, FuzzyPSO2. Esta nueva propuesta, denominada CCFPSO, utiliza la técnica de agrupación aleatoria que cambia el tamaño de los subcomponentes en cada generación. A diferencia de FuzzyPSO2, la reinicialización de las variables de CCFPSO, sugerida por el sistema difuso, se realizaron sobre las partículas con los peores valores de fitness. Además, en lugar de actualizar las partículas basándose en la mejor partícula global, CCFPSO se actualizó considerando la mejor partícula personal y la mejor partícula del vecindario. Esta propuesta se probó en problemas a gran escala que se asemejan a los del mundo real (CEC2008, CEC2010), donde el rendimiento de CCFPSO fue favorable en comparación con otras versiones de PSO del estado del arte, a saber, CCPSO2, SLPSO y CSO. Los resultados experimentales indican que el uso de un enfoque PSO coevolutivo cooperativo con un sistema de lógica difusa puede mejorar los resultados en problemas de alta dimensionalidad (de 100 a 1000 variables).

Palabras claves: Coevolución Cooperativa, Lógica Borrosa, Optimización por enjambre de partículas, Peso de inercia adaptativo.

1 Introduction

The problems of large-scale global optimization (LSGO) have long been a question of great interest in the science and engineering field. Major aspects that make them difficult to solve are: a) the exponential growth of the search space size with respect to the number of variables which becomes an extremely complex problem; moreover, the number of local optima increases; b) the high number of fitness evaluations that are required to achieve satisfactory performance; and c) the level of interaction of the decision variables that contributes to the difficulty of the problem [1] [2] [3]. Evolutionary algorithms and swarm intelligence algorithms, such as Genetic Algorithm (GA) [4], Differential Evolution (DE) [5], Particle Swarm Optimization (PSO) [6] [7], Ant Colony Optimization (ACO) [8], and Artificial Bee Colonies (ABC) [9] have been promising in solving many large-scale optimization problems. For this paper, which focuses on both CEC2008 [1] and CEC2010 [2] large-scale optimization problems, evolutionary and swarm intelligence algorithms have scalability issues in problems from a hundred to a thousand decision variables thus it remains an open problem [10].

To address the dimensionality challenge, Jian et al. [11] categorized evolutionary algorithms and swarm intelligence algorithms into two approaches, namely, decomposition and non-decomposition. The decomposition algorithms divide the decision variables into smaller scale groups to be solved by an optimizer and this approach is known as the cooperative coevolutionary (CC) approach [12]. CC has achieved great success in solving many LSGO problems [10], including DECC-G [13], MLCC [14], CCPSO2 [15] DECC-D [16] DECC-DG [17]. Non-decomposition algorithms consider all decision variables of a problem as a whole. Therefore, to improve their performance, these algorithms have included new initialization strategies [10, 11], operator designs (e.g., SLPSO [18], CSO [19]), and parameter self-adaptation [20]. Of the latter group, algorithms have used several strategies to dynamically adapt the parameters, such as the use of fuzzy logic (FL) systems [21]. Fuzzy logic systems have been successfully used to improve the quality of the solutions of these algorithms [22] [23], especially for low dimensionality problems (less than 500 variables) combining mostly with PSO and DE [24] [25].

PSO is a powerful algorithm and has been widely used [26]. However, it suffers from premature convergence getting stuck at local optima [27] [28] [15]. Appropriate adjustment of its parameters improves its performance, but it is a tedious task and usually requires a great deal of effort and time [25]. To overcome these drawbacks, we developed a previous work recently published in CACIC 2020 [24], named FuzzyPSO2, a particle swarm optimization algorithm that dynamically adapted the inertia weight parameter using fuzzy logic to address large-scale problems (up to 1000 variables). The variables defined for the fuzzy system were: iteration number and swarm diversity as input variables; inertia weight and sigma variable (to reinitialize a swarm part) as output variables. FuzzyPSO2 outperformed the standard version of PSO by far. However, there were functions, especially separable ones, where the results were not good enough [24]

In this paper, we present an improvement of previous work FuzzyPSO2 using the cooperative coevolutionary framework. The proposed algorithm, named CCFPSO, is considerably better than FuzzyPSO2 and presents the following improvements:

- To escape local optima, the fuzzy system restarts a proportion of particles with the worst fitness value of the current generation, rather than randomly chosen particles.
- To maintain high diversity in the population, a new velocity update is employed, in which particles follow their personal and neighborhood best position.
- To improve speed, the constriction factor approach is replaced by the inertia weight parameter approach.
- To improve performance on highly separable and non-separable problems, a decomposition strategy similar to MLCC is used.

The experiments were performed on the CEC2008 and CEC2010 benchmark function set and compared with the most prominent versions of PSO on large-scale problems of the last generation published in a recent paper [11] (CCPSO2, SLPSO, and CSO).

This work is organized as follows: Section 2 describes related work. Section 3 presents the standard PSO algorithm with dynamic inertia weight. Section 4 details the proposal. In Section 5 the experiments are performed. Finally, Section 6 presents conclusions and future work.

2 Related Work

A fuzzy logic system can control several variables based on information about the behavior of the algorithm. Using input information and linguistic rules, appropriate values of certain variables that can significantly influence the behavior of the algorithm can be obtained as the system output [29] [30] [25].

In recent years, several works have been presented to adapt the parameters using fuzzy logic (LF): such as Olivas et al. [23] [22] implemented LF in the PSO and the Ant Colony Optimizer (ACO); Perez et al. [31] used Bat Algorithm; Sombra et al. [32] implemented LF in the Gravitational Search Algorithm; Valdez et al. [30] [21] used LF with a set of algorithms including PSO, Genetic Algorithm (GA), and Ant Colony Optimization (ACO); Norouzzadeh et al. [33] employed LF in PSO; Ochoa et al. [34] used LF in the Differential Evolution (DE); and Kumar et al. [35] used LF in the PSO. Despite their success in improving algorithm performance, there are not enough studies on large-scale global optimization problems (more than 100 variables).

A large-scale global optimization problem (LSGO) can involve hundreds and thousands of decision variables. The Cooperative Coevolutionary (CC) approach can help to improve the results. CC algorithms before the optimization process use a variable decomposition strategy to form smaller scale sub-populations where they can be optimized separately [13] [10]. Hence, a decomposition strategy that improves the results of the problem must be selected.

Van den Bergh and Engelbrecht [36] presented two PSO models, called CPSO-S$_k$ and CPSO-H$_k$ using CC [12]. Yang et al. [13] [14] proposed a decomposition strategy based on Random Grouping of Variables (RG) in a Differential Evolution algorithm (DE), named DECC-G [13] and later Multilevel Cooperative Coevolution (MLCC) [14]. MLCC not only significantly outperforms many other existing decomposition methods, but also ensures that the CC approach. Later, Omidvar et al. [16] presented Differential Grouping (DG), Sun et al. [37] proposed the Extended Differential Grouping method (XDG), Mei et al. [38] presented Global Differential Grouping (GDG), Omidvar et al. [10] developed Differential Grouping version 2 (DG2) with the Differential Evolution (DE) algorithm. However, most of the current decomposition methods (includes DG, XDG, GDG, DG2) group interacting variables into a single group, thus it is not always possible to reduce the problem size [3] [39]. A recent paper used fuzzy logic in a CC approach to solving large-scale problems [40]. In that study was shown a new algorithm using multiple optimizers in a CC approach to evolving its subcomponents based on fuzzy heuristic rules. This heuristic focused on the most effective subcomponent and its optimizer based on two criteria, namely, fitness improvement and population diversity.

Inspired by these works and to improve our preliminary work FuzzyPSO2, we present CCFPSO, combining fuzzy logic to adapt the inertia weight parameter and the CC approach to improve the performance on a wide range of large-scale optimization problems.

3 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is an algorithm based on Swarm Intelligence theory, which is inspired by the social behavior of certain animals when they interact with another of their same species to achieve a common goal. PSO was proposed by Kennedy and Eberhart in 1995 and was developed to simulate the movements of birds [7] [6]. PSO has attracted the interest of many researchers due to its simple model, easy implementation and good results. Particle flies through the search space in search of an optimal solution. The particle's movement is the result of adding to the current position ($x_i(t)$) a velocity ($v_i(t)$) that is modified according to its personal best position and the global best position. Each particle has a personal-best particle ($p_{best_i}(t)$) which represents its best fitness value reached so far, a global-best particle (($g_{best}(t)$)) that has the best fitness value of the swarm, a velocity $v_i(t)$, and a current position $x_i(t)$. Consequently, each particle is updated according to Eqs. 1 and 2 in the following way:

$$\begin{aligned} v_i(t+1) = w * v_i(t) + c_1 * r_1 * (p_{best_i}(t) - x_i(t)) \\ + c_2 * r_2 * (g_{best}(t) - x_i(t)) \end{aligned} \tag{1}$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{2}$$

where $v_i(t)$ is the velocity of particle i, $x_i(t)$ is the current position of particle i; r_1 and r_2 are random numbers between $[0,1]$; p_{best_i} is the best position found by particle i; and g_{best} is the best swarm particle. The variables c_1 and c_2 represent the cognitive and social learning coefficients, respectively. These values are generally constant, but can also be dynamic. The variable w is called inertial weight which can also be static or dynamic [6] [7]. To improve the control of particle velocities several authors, such as [27] [29] [41] incorporated w into the original PSO algorithm and demonstrated that this weight can influence the exploration and exploitation abilities. Algorithm 1 shows the standard PSO.

Algorithm 1 standard PSO

```
1:  swarm initialization
2:  evaluate swarm
3:  while $Gen \leq MaxGen$ do
4:      for $i = 1 : N$ do
5:          select $g_{best}$
6:          update velocity Eq. (1)
7:          update position Eq. (2)
8:          evaluate $x_i^{Gen+1}$
9:          if $f(x_i^{Gen+1}) < f(p_{best_i}^{Gen})$ then
10:             $p_{best_i}^{Gen+1} = x_i^{Gen+1}$
11:         end if
12:     end for
13: end while
```

4 Algorithm CCFPSO

4.1 Control of parameters through fuzzy logic

PSO is known to be prone to premature convergence [23] [41] [35] [33]. One of the major reasons for this behavior is due the fast movement of particles when exchanging information, leading to low swarm diversity in initial iterations [15]. Therefore, the fuzzy system must know what the state of the swarm is to detect swarm stagnation. For this purpose, a fuzzy system that dynamically adapts the inertia weight parameter was developed. Shi et al. [27] [41] used an inertia weight (w) that decreases linearly during the iterations and achieved to improve the performance in several applications. These studies provide the necessary knowledge to build linguistic rules. The variables and linguistic rules of the fuzzy system are described below.

4.1.1 Input variable

The first input variable is $iteration$. This input variable takes a value between 0 and 1 and is defined in Eq. 3. It can therefore be understood as the degree of progress of the optimization process.

$$iteration = \frac{iteration_{current}}{Maximum\_iterations} \tag{3}$$

The second input variable is $DiverN$ which represents the diversity of the swarm. Diversity can be understood as the distance between the particles (x_{id}) and the best particle ($\dot{x}_{id}$) of the swarm in each generation. Therefore, the Euclidean distance defined in Eq. 4 was calculated, where ns represents the number of particles and nx represents the dimension of particle i. Subsequently, this result ($Diver$) was normalized using the variables $minDiv$ and $maxDiv$ representing the minimum and maximum diversity, respectively, as indicated in Eq. 5. In this way, the variable $DiverN$ will take a value close to 0 when the swarm diversity is low, otherwise close to 1 when the swarm diversity is high [23] [22].

$$Diver = \frac{1}{ns}\sum_{i=1}^{ns}\sqrt{\sum_{d=1}^{nx}(x_{id}-\dot{x}_d)^2} \tag{4}$$

$$DiverN = \begin{cases} 0, & \text{when } minDiv = maxDiv \\ \frac{Diver-minDiv}{maxDiv-minDiv}, & \text{otherwise} \end{cases} \tag{5}$$

For input variables, we selected triangular membership functions in the interval [0,1]. These values were granulated into three triangular membership functions (low, medium, high) based on our previous work and also considering the successful results obtained by other works, such as [29] [23] [22] [25]. The membership functions of each of these fuzzy variables are shown in Figs. 1 and 2.

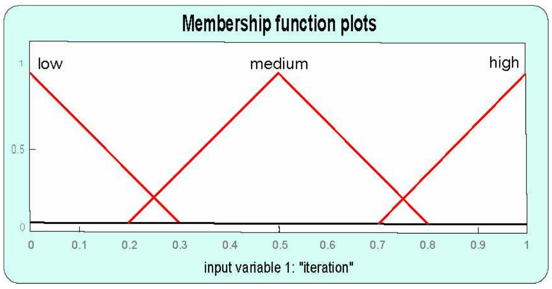


Figure 1: Input variable: $iteration$

4.1.2 Output Variable

The first output variable of the fuzzy system is the inertia weight parameter (w) named $inertiaWeight$. As mentioned above, the inertia weight parameter can influence the ability to explore and exploit within the search space. A higher inertia weight

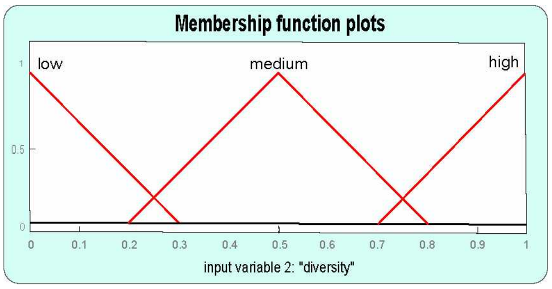


Figure 2: Input variable: $DiverN$

facilitates exploration, while a lower inertia weight facilitates exploitation. Thus, an appropriate value of $inertiaWeight$ provides a balance in the search for the best solutions and requires fewer iterations [27]. Therefore, it is a key parameter to be controlled by the fuzzy system. Consequently, the particle velocity was defined as shown in Eq. 1. The value of variable $inertiaWeight$ was defined in the interval [0,1] and granulated into five triangular membership functions (veryLow, low, medium, high, veryHigh), as shown in Fig. 3.

The second output variable for the fuzzy system is called $sigma$. This variable, unlike FuzzyPSO2, represents the swarm ratio for particle restarting over the particles with worse fitness values at time t. Moreover, it aims to maintain diversity and escape from local optima. Therefore, both $DiverN$ and $iteration$ influence the value it can take. The maximum value of $sigma$ must be chosen carefully, otherwise, the algorithm may not converge to the optimum. The $sigma$ variable was empirically defined in the range [0,0.2] and granulated into five triangular membership functions (veryLow, low, medium, high, veryHigh) (see Fig. 4). Their implementation is shown in Algorithm 2. In Figs. 3 and 4 the membership functions for each of the output variables are shown.

Algorithm 2 Sigma

```
1: particl=sort(fitness,'descend')    ▷ sort fitness values in decreasing order
2: Numpartic=$sigma$*numSwarm
3: for i=1:Numpartic do
4:     reset $x_{particl(i)}$
5: end for
```

4.1.3 Linguistic rules

To design the rules of the fuzzy system, the idea of improving the balance in the abilities to explore and exploit the search space of the PSO algorithm is addressed. It is known that some problems need more exploration than exploitation, while others, more exploitation than exploration [33]. Therefore, we decided that when the diversity is low in the

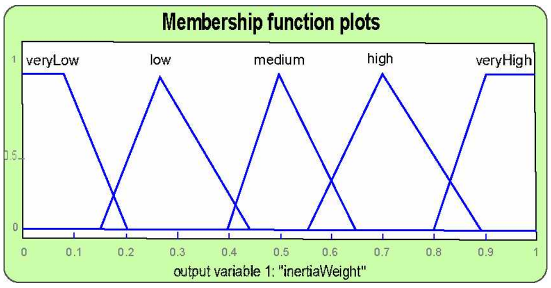


Figure 3: Output variable: *inertiaWeight*

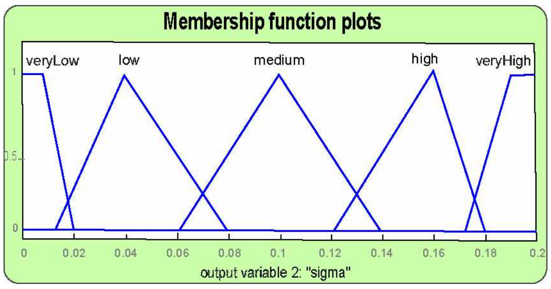


Figure 4: Output variable: *sigma*

early iterations, the value of *inertiaWeight* and *sigma* should take a high value to help explore more promising regions, but when the diversity is low in the final iterations, both values of *inertiaWeight* and *sigma* should take a low value to exploit the zone and not abandon the promising zones. The rules of the fuzzy system are listed below.

1. If (iteration is low) and (DiverN is low) then (inertiaWeight is veryHigh) (sigma is veryHigh).
2. If (iteration is medium) and (DiverN is low) then (inertiaWeight is medium) (sigma is low).
3. If (iteration is high) and (DiverN is low) then (inertiaWeight is veryLow) (sigma is veryLow).
4. If (iteration is low) and (DiverN is medium) then (inertiaWeight is high) (sigma is high).
5. If (iteration is medium) and (DiverN is medium) then (inertiaWeight is medium) (sigma is medium).
6. If (iteration is high) and (DiverN is medium) then (inertiaWeight is veryLow) (sigma is veryLow).
7. If (iteration is low) and (DiverN is high) then (inertiaWeight is veryHigh) (sigma is high).
8. If (iteration is medium) and (DiverN is high) then (inertiaWeight is medium) (sigma is low).
9. If (iteration is high) and (DiverN is high) then (inertiaWeight is veryLow) (sigma is veryLow).

4.1.4 Fuzzy System

The designed fuzzy systems are of the Mamdani type and are ideal for this type of control [31] [23] [22] [30] [21] [42]. To obtain the values of *inertiaWeight* and *sigma*, the fuzzy system (FS) is called in each iteration before updating the particles. The assigned value for these variables is obtained using the centroid method. The complete fuzzy system is shown in Fig. 5.

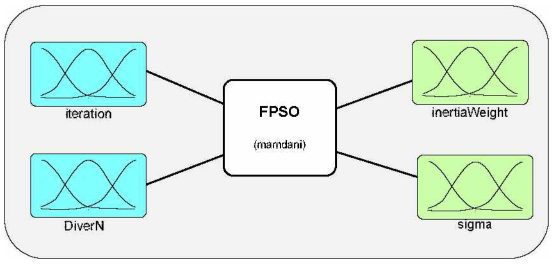


Figure 5: Fuzzy System (FS)

4.2 FPSO Cooperative Coevolutionary

To improve the performance of the previously proposed FuzzyPSO2, we conducted changes as described below:

- We incorporate a CC approach to the FuzzyPSO2 algorithm and adopt the decomposition strategy of [16] [15]. This strategy is similar to MLCC [14] since the variables are randomly grouped, but the subcomponent size is chosen randomly from the decomposer set at each generation. This strategy was chosen not only because it does not consume FEs (number of evaluations) in the decomposition, but also because it ensures the CC approach. Hence, it can provide better performance on highly separable and non-separable problems [15].
- We choose to update the particle velocities with the inertia weight parameter approach as shown in Eq. 6, since better exploration and exploitation efficiency is achieved when a reasonable search area is limited [27].
- We use a neighborhood structure with a ring topology of size 3 for each particle. This neighborhood structure and size improves the standard PSO search ability, achieves good performance on multimodal problems [43] [15], and helps to maintain higher diversity in the population [7] [26]. Therefore, the velocity update was defined as shown in Eq. 6.

$$v_i(t+1) = (inertiaWeight) * v_i(t) + c_1 * r_1 * (p_{best_i}(t) - x_i(t)) + c_2 * r_2 * (l_{best}(t) - x_i(t)) \quad (6)$$

The CCFPSO algorithm can be summarized in Algorithm 3. First, the subcomponent size is randomly chosen to generate the subswarms at each generation. Second, the sub-swarms are constructed by permuting and grouping the indices of the n dimensions into K groups of s dimensions where $K * s = n$. Third, the values of *inertiaWeight* and *sigma* parameters are obtained through a fuzzy logic system. Fourth, the particles are updated using

Eqs. 6 and 2. Fifth, information exchange is performed to update the context vector with the best particles and continue with the optimization of the next sub-swarm. Finally, a generation is completed when all the sub-swarms are optimized. In this case, the algorithm continues with the next generation and chooses other subcomponent sizes (K).

Algorithm 3 CCFPSO

```
 1: swarm initialization
 2: evaluate swarm
 3: Define the decomposer group S
 4: while $FEs \leq MaxFEs$ do
 5:     randomly choose $s$ from $S$        ▷ obtain the
    $k - subswarm$ of $n/s$, where $n$ is the dimension of
    the problem
 6:     randomly permute all indices of dimension $n$
 7:     construct $K - subswarms$, each with $s$
    dimensions
 8:     for k=1:$K - subswarm$ do
 9:         calculate $iteration$ Eq. 3
10:         calculate $DiverN$ Eqs. 4 and 5
11:         [$inertiaWeight, sigma$]=FS($iteration, DiverN$)
12:         reinitialize particles using Alg. 2
13:         for $i = 1 : NP$ do
14:             update $l_{best}$
15:             update velocity Eq. (6)
16:             update position Eq. (2)
17:             evaluate $x_i^{Gen+1}$
18:             if $f(x_i^{Gen+1}) < f(p_{best_i}^{Gen})$ then
19:                 $p_{best_i}^{Gen+1} = x_i^{Gen+1}$
20:             end if
21:         end for
22:         information exchange
23:     end for
24: end while
```

5 Experiments

In this section, we evaluate the performance and scalability of CCPSO as well as other state-of-the-art algorithms. Therefore, we use four experimental sets of dimensions 100, 500, and 1000 and develop two well-known metrics in the literature, namely the Wilcoxon-Rank-Sum statistical test and Friedman ranks for a thorough evaluation of all compared algorithms [44].

5.1 Benchmark Functions

Benchmark functions are fundamental to validate and compare the performance of optimization algorithms [16] [1] [2]. The CEC2008 functions are the classical large-scale benchmark functions, such as Schewefel, Rosenbrock, Rastrigrin, Griewank, Ackley, and Sphere. Table 1 details the properties of modality, separability (variable boundary), and domain of the search space of these functions, which can scale to any dimension. In this case, we use the scales to 100, 500, and 1000 dimensions as the first three sets of experiments. For all CEC2008 functions, the minimum is 0 except for f7 which is unknown. Finally, the fourth set of experiments are the CEC2010 functions that consist of 20 benchmark functions, all of them 1000-dimensional. Unlike CEC2008, CEC2010 incorporates partially separable functions. The modality, separability, and domain properties of these functions are detailed in [2]. It is important to mention that separability is a property that can affect the convergence of algorithms. This challenge depends on the level of interrelated variables (non-separable variables) that the problems can have [3] [28].

Table 1: CEC2008 Benchmark Functions

Modality	Functions	Separability	*Domain*
unimodal	f1:Shifted Sphere	Separable	[-100,100]
	f2:Shifted Schwefel 2.21	Non-separable	[-100,100]
multimodal	f3:Shifted Rosenbrock	Non-separable	[-100,100]
	f4:Shifted Rastrigin	Separable	[-5,5]
	f5:Shifted Griewank	Non-separable	[-600,600]
	f6:Shifted Ackley	Separable	[-32,32]
	f7:FastFractal "DoubleDip"	Non-separable	[-1,1]

5.2 Experimental Settings

The CCFPSO parameters were calibrated using the IRACE algorithm [45]. To do this, we used a subset of the CEC2008 functions with different dimensions (100, 500, and 1000) and the CEC2010 functions in the "instances.txt" file (of IRACE) to find a parameter configuration that works well in most of the functions. The parameters to be calibrated by IRACE were: population size (NP), the social and cognitive learning coefficients ($c1$, $c2$). A $maxExpermients$=300 and a default setting of NP=50 and $c1$=$c2$=1.49445 in the "default.txt" file was used. Since IRACE could not find other better configurations, we chose to delete the "$c1$ and $c2$ calibration parameters. Therefore, IRACE only calibrated the population size of CCFPSO (see Table 2). An important aspect to remember is that in the CC approach, the number of particles is usually smaller due to the internal logic structure, therefore, it is crucial to calibrate this parameter. A set of possible subcomponent sizes for all dimensions was defined empirically in S=$\{D/2, D/4, D/5, D/10, D/20, D/50\}$, where D is the problem dimension, in the two test function sets (CEC2008 and CEC2010). The size of the subcomponents is dynamic and is selected randomly for each generation.

To compare CCFPSO with state-of-the-art algorithms, we selected three outstanding algorithms from recently published comparative studies for large-scale optimization problems, namely, SLPSO and CSO by the work of Jian et al. [11], and CCPSO2

Table 2: Configuration of the parameters of the involved PSO algorithms

	Parameter Setting
CCFPSO	*NP*=34 (with IRACE), *c*1=*c*2=1.494, *w* adaptive (with FS)
CCPSO2	*NP*=30, *p*=0.5 (probability value)
FuzzyPSO2	*NP*=100, *c*1=2, *c*2=3, *w* adaptive (FS)
CSO	*NP*=100/250/500, ϕ=0-0.15 (soc. fact.)
SLPSO	*NP*=110/150/500/200, α=0.5

by the work of Ullmann et al. [28]. Both CSO [19] and SLPSO [18] are non-decomposition algorithms, whereas CCPSO2 [15] is a decomposition algorithm. In addition, our preliminary version of FuzzyPSO2 [24] was considered to validate the CCFPSO improvements. Each algorithm had its own parameter setting. The parameter values used in this paper for CSO, SLPSO, and CCPSO2 were extracted from the authors' original papers. Table 2 presents the configurations of all the compared algorithms.

To be fair, we have chosen to follow the guidelines of the CEC2008 and CEC2010 benchmark functions for all these algorithms. Thus, for both CEC2008 and CEC2010, 25 runs were performed for each function. Furthermore, for CEC2008, $MaxFEs = 5E + 03 * D$ fitness evaluations (FEs) were used, where D is the problem dimension; whereas, for CEC2010, $MaxFEs = 3E + 06$ fitness evaluations (FEs) were used. In this study, the number of fitness evaluations is a fundamental measure to achieve satisfactory performances in the algorithms.

5.3 Analysis of Results

In this section, CCFPSO, FuzzyPSO2, CSO, SLPSO, and CCPSO2 experiments were performed on 7 CEC2008 benchmark functions in 100, 500, and 1000 dimensions as shown in Table 3, and on 20 CEC2010 benchmark functions in 1000 dimensions as illustrated in Table 4.

The first statistical metric required to evaluate the performance of these algorithms is the Wilcoxon-Rank-Sum statistical test. Since the population of these experiments does not have a normal distribution, the nonparametric Wilcoxon-Rank-Sum statistical test with 95% confidence is used. This test was performed on the best mean fitness values for each algorithm obtained from the 25 runs for each function. The symbols "(+)", "(-)" and "(=)" mean that the CCFPSO results are significantly better, significantly worse, and equivalent to the compared algorithms. When the results are statistically different, the better is highlighted in bold (the best average of the best fitness values). In addition, $w/l/t$ in the last row of Tables 3 and 4 indicate that CCFPSO wins on "w" functions, loses on l functions, and ties on t functions.

On the other hand, to analyze the experimental results in Tables 3 and 4, the tolerance error ($f(x)$ - $f(x^{*})$) at $1.0E-8$ was considered, where $f(x)$ is the function value and $f(x^{*})$ is the global optimum [28] [2] [1]. Finally, to plot the convergence curves of all compared algorithms, six representative functions, namely f1-f6 for the first benchmark set CEC2008, and eight, such functions f1, f3, f10, f12, f15, f17, f18 and f20 for the second benchmark set CEC2010 were selected. These plots are shown in Figs. 6-9.

5.3.1 Comparison of CCFPSO with FuzzyPSO2

The mean best fitness value of 25 independent runs of CCFPSO and FuzzyPSO2 are summarized in Table 3. The comparison of results shows that CCFPSO outperforms FuzzyPSO2 on 19 out of 21 functions (f1, f2, f4, f5, f6, and f7 in all dimensions). On f3 (unimodal and non-separable) FuzzyPSO2 was better than CCFPSO in 500D, but not statistically different in 100D. Furthermore, Figs. 6 and 7 show that CCFPSO achieved better convergence on f1, f2 (unimodal), and f4-f6 (multimodal), while similar convergence to FuzzyPSO2 on f3. Note that f3 is non-separable and presents a greater challenge for any algorithm. CCFPSO was able to find the optimal values in 7 functions, since most of them are within the established tolerance error, while FuzzyPSO2 could not find any optimal value. An important difference between CCFPSO and FuzzyPSO2 is that the diversity of CCFPSO is likely to be much greater than that of FuzzyPSO2. In CCFPSO each particle has its own l_{best} in each update, whereas, FuzzyPSO2 all particles are updated using the (same) g_{best}. Therefore, FuzzyPSO2 may converge prematurely and very quickly over several functions.

5.3.2 Comparison of CCFPSO with CCPSO2

CCPSO2 also adopts a CC approach (like CCFPSO) with different subcomponents sizes. In CEC2008 (see Table3), CCFPSO was superior to CCPSO2 on 7 out of 21 functions (f1 of 100D and 500D; f2-f3 and f5 of 1000D; and f6 of 500D and 1000D), while CCFPSO loses to CCPSO2 in 6 out of 21 functions (f2, f4 of 100D and 500D; and f7 of 500D and 1000D). Finally, there is not statistical different in 8 out of 21 functions (f1, f3, f5, f6, f7 in 100D, f3, f5 in 500D, and f4 in 1000D). Although these algorithms used a CC approach, the improvement of CCFPSO over CCPSO2 on certain functions (specifically from 500 to 1000 dimensions) may be due to that the fuzzy system was less affected by scalability. In addition, CCFPSO had better convergence on unimodal functions. The CCPSO2 only found optimal values in 6 functions (f1 (100D, 500D and 1000D), f4 (100D), and f6 (100D and 500D). The plots in Figs. 6 and 7 show the

Table 3: Experimental results of 5 algorithms on 7 test suits (CEC2008) of 100, 500 and 1000 dimensions.

Fun	D	CCFPSO	CCPSO2	CSO	SLPSO	FuzzyPSO2
	100	1.702E-23	4.426E-23 (=)	**0.000E+00 (-)**	1.047E-27 (-)	1.605E+03 (+)
f1	500	**0.000E+00**	1.832E-15 (+)	6.565E-23 (+)	7.517E-24 (+)	2.901E+04 (+)
	1000	**7.863E-29**	6.697E-11 (+)	1.077E-21 (+)	7.480E-23 (+)	9.227E+04 (+)
	100	9.829E+00	3.984E+00 (-)	3.332E+01 (+)	**6.040E-06 (-)**	9.465E+01 (+)
f2	500	1.777E+01	**1.088E+01 (-)**	1.466E+01 (-)	3.525E+01 (+)	9.491E+01 (+)
	1000	**1.745E+01**	2.704E+01 (+)	3.222E+01 (+)	9.018E+01 (+)	9.535E+01 (+)
	100	2.812E+02	3.133E+02 (=)	**1.790E+02 (-)**	2.110E+02 (-)	2.040E+02 (=)
f3	500	1.205E+03	9.939E+02 (=)	5.368E+02 (-)	**5.211E+02 (-)**	1.198E+03 (-)
	1000	1.992E+03	2.149E+03 (+)	**1.002E+03 (-)**	1.029E+03 (-)	3.379E+03 (+)
	100	8.376E-12	**3.695E-15 (-)**	5.469E+01 (+)	7.589E+01 (+)	2.906E+02 (+)
f4	500	2.546E+02	**4.851E+00 (-)**	3.213E+02 (+)	2.940E+03 (+)	3.467E+03 (+)
	1000	**9.759E+02**	3.574E+02 (=)	7.057E+02 (=)	5.740E+02 (=)	9.231E+03 (+)
	100	**4.325E-03**	7.186E-03 (=)	4.926E-04 (=)	4.947E-02 (=)	1.137E+01 (+)
f5	500	9.855E-04	1.774E-03 (=)	**2.220E-16 (-)**	3.375E-16 (-)	1.989E+02 (+)
	1000	1.656E-15	9.860E-04 (+)	**2.220E-16 (-)**	5.507E-16 (-)	7.828E+02 (+)
	100	4.623E-13	5.546E-13 (=)	**1.066E-14 (-)**	1.833E-14 (-)	1.484E+01 (+)
f6	500	**1.128E-13**	1.706E-09 (+)	4.106E-13 (+)	1.494E-13 (+)	1.775E+01 (+)
	1000	**1.640E-13**	2.879E-07 (+)	1.210E-12 (+)	3.524E-13 (+)	1.958E+01 (+)
	100	**-1.488E+03**	-1.486E+03 (=)	-1.471E+03 (+)	-1.438E+03 (+)	-1.203E+03 (+)
f7	500	-6.737E+03	**-7.095E+03 (-)**	-7.060E+03 (-)	-7.040E+03 (-)	-4.943E+03 (+)
	1000	-1.209E+04	-1.334E+04 (-)	**-1.402E+04 (-)**	-1.396E+04 (-)	-9.316E+03 (+)
		w / 1 / t	7/6/8	9/10/2	9/10/2	19/1/1

convergence of CCPSO2 concerning CCFPSO, where good convergence is observed on f3 and f4, similar convergence on f6, and poor convergence on f1 and f2.

In CEC2010, experimental results shown that CCFPSO wins against CCPSO2 in 13 of the 20 functions, such as, 3 are separable functions (f1-f3); 8 are partially separable functions (f9, f10, f12-15, f17 and f18); and 2 corresponding to non-separable functions (f19 and f20). CCFPSO loses in 4 out of 20 functions (f4, f5, f7 and f11) and ties in 3 out of 20, which are partially separable functions (f6, f8 and f16) (see Table 4). CCPSO2 used small subcomponent sizes compared with CCFPSO, which could cause poor performance on non-separable or partially separable functions where large groups of strongly related variables exist. CCFPSO was able to find the optimal values in two separable functions (f3 and f6), while CCPSO2 was not able to find the optimum in any function. Figs. 8 and 9 illustrate that, in the first third of the generations, CCPSO2 has better convergence than CCFPSO, but then it is outperformed by CCFPSO because it converges faster and better.

5.3.3 Comparison of CCFPSO with CSO

In CEC2008 (see Table 3), CCPSO2 outperformed CSO on 9 out of 21 functions, which 4 are unimodal functions (f1 of 500D and 1000D and f2 of 100D and 1000D) and 5 are multimodal functions (f4 in 100D; f4, f6 in 500D; and f6, f7 in 1000D). However, CSO outperforming CCFPSO on 10 out of 21 functions (f1-100D, f2-500D, f3 in all dimensions, f5 in 500D and 1000D, f6-100D, and f7in 500D and 1000D), but its performance is not statistically different from CCFPSO on f4 in 1000D and f5 in 100D. On the other hand, CSO was able to find the optimum of 8 functions, namely f1 and f6 in all dimensions, f5 of 500D and 1000D. It is important to mention that CSO used a solid configuration for each dimension [19], i.e., for 100, 500, and 1000 dimensions, it used 100, 250, and 500 particles, respectively. CCFPSO was calibrated to use the same population size for 100, 500, and 1000 dimensions. Despite this, the CCFPSO achieved outstanding performance in functions f6-D500, f1-D1000 and f2-D1000, see Figs. 6 and 7.

In CEC2010, Table 4 shows that CSO aoutperforms CCFPSO in most of the functions, namely on 13 out of 20 functions, in some partially separable (f4-f9, f11, f13, f14, f16 and f18) and non-separable (f19 and f20) functions. However, CCFPSO is significantly better on separable functions (f1-f3) and partially separable functions f10, f12, f15 and f17. The fact that CCFPSO has been outperformed in partially separable and non-separable functions may be due to the decomposition approach used, where the variables are randomly grouped without considering the interactions between them, thus reducing the overall performance of these functions. Both algorithms were able to reach the optimum on functions f1 and f3. Figs. 8 and 9 show that CSO has good convergence on most functions, except for f10-1000D and f15-1000D, which stagnate in the first few iterations. However, it converges faster and better than CCPSO2 at f19-1000D and f20-1000D.

5.3.4 Comparison of CCFPSO with SLPSO

In CEC2008, Table 3 shows that CCFPSO performed significantly better than SLPSO on 5 functions (f4 of 100D and 500D, f6 of 500D and 1000D, and f7 of 100D), while losing to SLPSO on 10 functions (f1

Table 4: Experimental results of 4 algorithms on 20 test suits (CEC2010) of 1000 dimensions.

Fun	CCFPSO	CCPSO2	CSO	SLPSO
f1	**1.6678E-16**	1.3295E+04 (+)	4.6384E-12 (+)	1.6678E-14 (+)
f2	**4.5657E+02**	7.1253E+02 (+)	7.4868E+03 (+)	2.8994E+03 (+)
f3	**9.4755E-12**	8.5013E-02 (+)	2.5548E-09 (+)	1.8808E-10 (+)
f4	8.3437E+12	4.7278E+12 (-)	**1.2089E+12 (-)**	1.3005E+12 (+)
f5	6.6259E+08	5.0671E+08 (-)	4.3055E+06 (-)	1.0749E+07 (-)
f6	1.9782E+07	1.8685E+07 (=)	7.9519E-07 (-)	**2.3397E-07 (-)**
f7	1.2125E+09	2.7969E+08 (-)	**1.0767E+04 (-)**	8.2008E+04 (-)
f8	2.6038E+08	1.5114E+08 (=)	**4.3211E+07 (-)**	4.3677E+07 (-)
f9	3.0808E+08	4.2367E+08 (+)	6.3573E+07 (-)	**5.6100E+07 (-)**
f10	**2.9501E+03**	3.4783E+03 (+)	9.5923E+03 (+)	8.9146E+03 (+)
f11	2.3241E+02	2.2307E+02 (-)	4.1343E-08 (-)	**3.0612E-09 (-)**
f12	**2.3174E+05**	4.6024E+05 (+)	5.2595E+05 (+)	5.5336E+05 (+)
f13	1.1041E+04	1.2900E+04 (+)	**9.3061E+02 (-)**	1.1393E+03 (-)
f14	1.0681E+09	1.6073E+09 (+)	2.8014E+08 (-)	**2.7267E+08 (-)**
f15	**5.7851E+03**	1.6053E+04 (+)	1.0066E+04 (+)	1.0205E+04 (+)
f16	4.1903E+02	4.0881E+02 (=)	5.9757E-08 (-)	**3.7584E-09 (-)**
f17	**1.4000E+06**	2.1142E+06 (+)	2.1935E+06 (+)	2.5888E+06 (+)
f18	5.3487E+04	1.5252E+05 (+)	**3.4612E+03 (-)**	5.2207E+03 (-)
f19	2.0337E+07	2.6769E+07 (+)	**9.5672E+06 (-)**	1.0482E+07 (-)
f20	1.9168E+03	7.0587E+04 (+)	**1.0029E+03 (-)**	1.0347E+03 (-)
	w/l/t	13/4/3	7/13/0	7/13/0

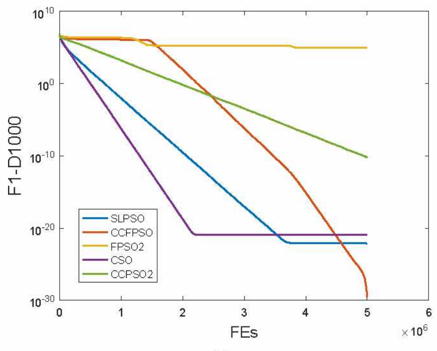
and f2 of 100-D; f6 of 100D; f3 of 100D and 500D; f5 and f7 of 500D and 1000D). For the remaining two test functions, there is no statistical difference between the two compared algorithms (f4 of 1000D, f5 of 100D). SLPSO found the optimal values for f1 and f6 in all dimensions and f5 in 500D and 1000D since the average of their best fitness values fell within the acceptable error (8 functions in total). As can be seen, the statistical results of the comparison with SLPSO are similar to those compared with CSO. For most of the functions observed in Fig. 6, SLPSO converges faster than CCFPSO and has better convergence at f3-1000D and f4-1000D. However, CCFPSO converges better on most of the unimodal functions.

In CEC2010, SLPSO also performs similarly to CSO, outperforming CCFPSO on 13 out of 20 functions,(f4-f9, f11, f13, f14, f16 and f18-f20), but it lost to CCFPSO on 7 out of 20 functions (f1-f3, f10, f12, f15 and f17) as shown in Table 4. Figs. 8 and 9 show that the convergence of SLPSO is still faster than that of CCFPSO in most functions, except for f2-D1000, which falls prematurely into a local optimum. Note that for functions f10 and f15, CCFPSO was able to escape local optima, possibly due to the sigma variable used by the fuzzy system.

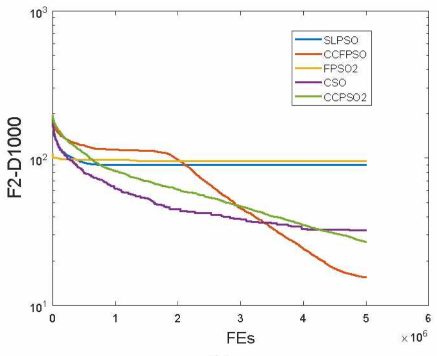
A second metric was used to rank the algorithms. This metric is the Friedman rank [44] and is shown in Tables 5 and 6 for the benchmark test functions CEC2008 and CEC2010. The last rows of the tables show that the CSO algorithm is the best algorithm with a score of 2.14 and 2, respectively.

In CEC2008, Table 5 shows that CSO obtained the best results for most functions. Therefore, it ranked first among the compared algorithms with a score of 2.14. This can be attributed not only to the robust configuration for this test set but also to the social factor adjusting to the dimensions and separability properties of the functions. In second place were the SLPSO and CCFPSO algorithms which tied with a point score of 2.57. Although CCFPSO lost to SLPSO in 10 functions, it won to SLPSO in 9 functions with a significantly better solution quality than SLPSO. CCFPSO obtains better values in most separable functions, so this advantage is attributed to the CC method. Third place in the ranking was for CCPSO2 with a score of 2.81. Although it found the best value for four functions, its convergence rate was slower for most functions. Finally, the last place went to FuzzyPSO2 for obtaining the worst performance concerning the other algorithms. The poor performance of FuzzyPSO2 indicates that a fuzzy system is not sufficient to improve PSO performance. However, it shows that combined with a CC approach (as CCFPSO) it can improve the performance of PSO on separable and non-separable problems. Due to its poor performance was not chosen in the comparison with the CEC2010 benchmark functions.

In CEC2010, CSO also achieved the best performance in 7 out of 20 functions. Thus, the first place was for CSO, which could converge to better solutions in most of the partially separable and non-separable functions. Second place went to SLPSO with a score of 2.1 (see Table 6). Both CSO and SLPSO performed better on most functions compared to the CCFPSO and CCPSO2 decomposition algorithms. The reason for this may be due to the designed operators they use, which improve the exploration and exploitation ability. These versions are very different from the standard PSO, as they do not use p_{best} and g_{best} in the particle updates, which are the heart of the PSO algorithm. In third place was CCFPSO with a 2.65 score. Like CSO,



(a)



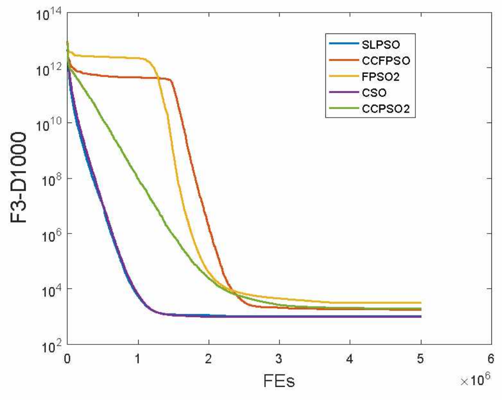
(b)

Figure 6: Convergence curves of CCFPSO, CSPSO2, CSO, SLPSO and FuzzyPSO2 (FPSO2) on $f1, f4$ in 1000 dimensions - CEC2008

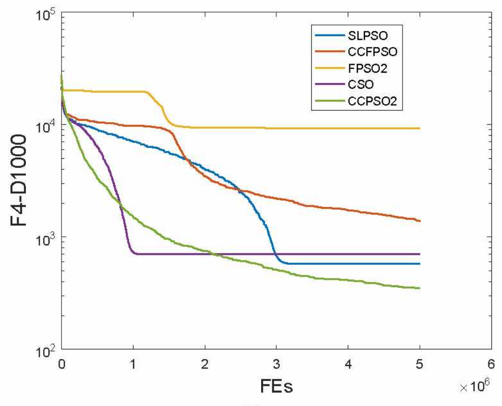
CCFPSO achieved the best performance in 7 out of 20 functions. It was outstanding in the separable functions and several partially separable functions, both unimodal and multimodal. Finally, fourth place was for the CCPSO2 algorithm with 3.25 points. Its performance was more reduced in the non-separable functions, which may be due to the decomposition method used.

6 Conclusions

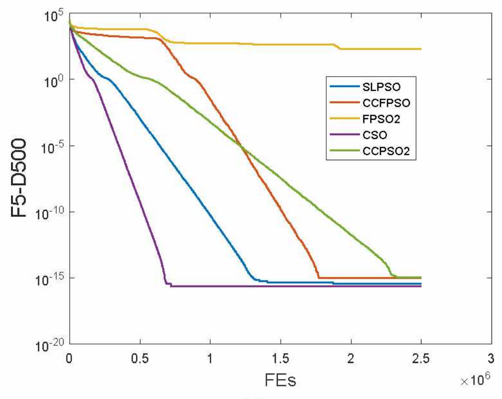
In this paper, a cooperative coevolutionary framework with dynamic adjustment w using a fuzzy system was proposed to address large-scale optimization problems. In addition, we incorporate a decomposition strategy based on random grouping of variables, which change size dynamically at each generation. This decomposition strategy is suitable for optimizing separable functions. To maintain the diversity of the population, we define a local neighborhood of size 3 for each particle using the ring topology. Therefore, to demonstrate the improvements of this proposal, it was compared with three state-of-the-art algorithms and with our preliminary work FuzzyPSO2. Moreover, we adopted two commonly used large-scale benchmark function sets: the CEC2008 and CEC2010 test sets to compare the performance of these five



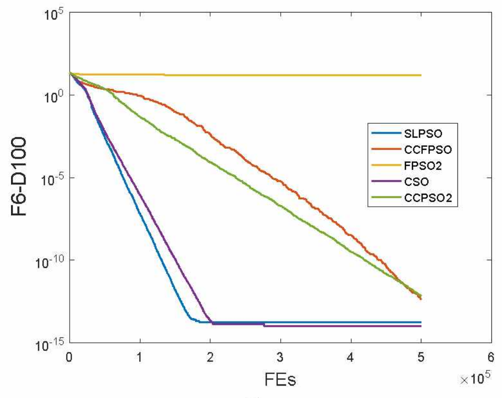
(a)



(b)

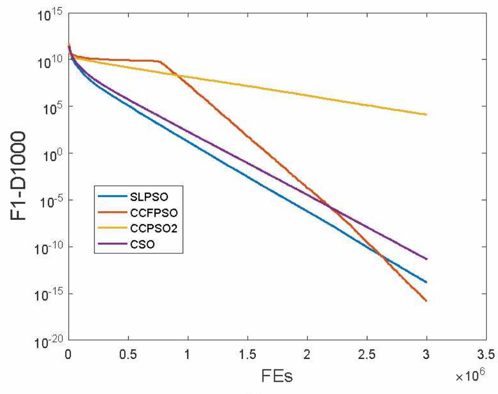


(c)

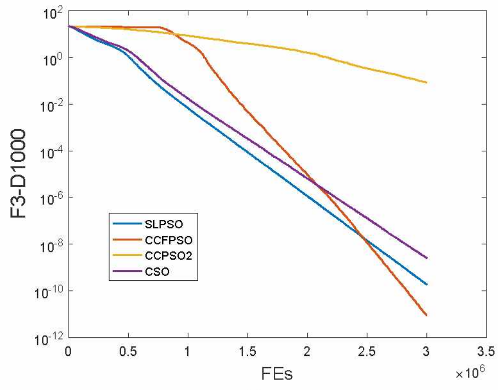


(d)

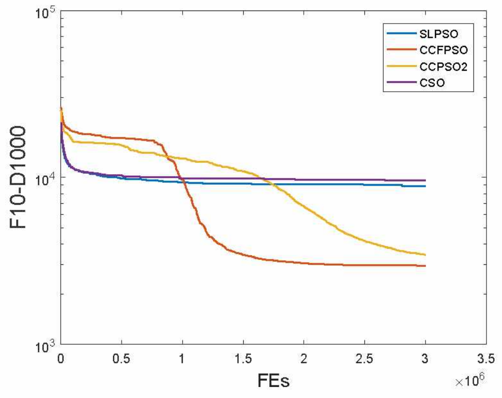
Figure 7: Convergence curves of CCFPSO, CSPSO2, CSO, SLPSO and FuzzyPSO2 (FPSO2) on $f3 - f6$ - CEC2008



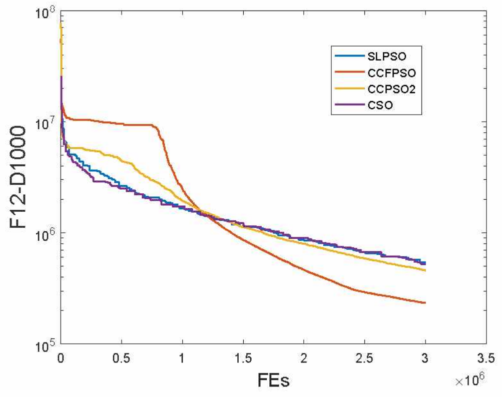
(a)



(b)

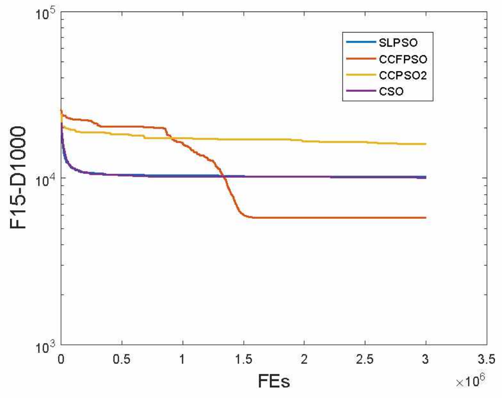


(c)

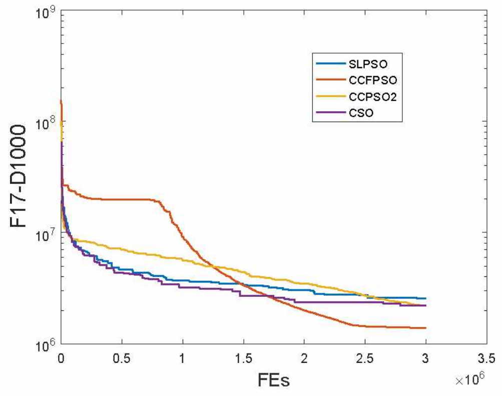


(d)

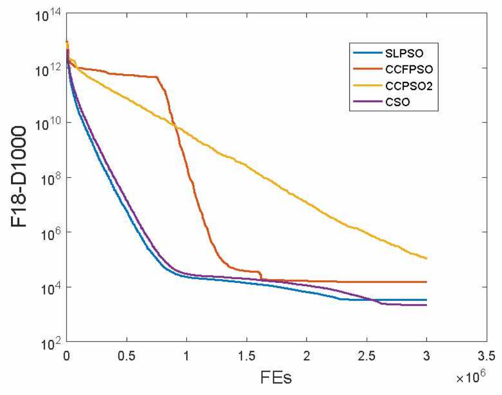
Figure 8: Convergence curves of CCFPSO, CCPSO2, CSO and SLPSO on $f1, f3, f10, f12$ - CEC2010



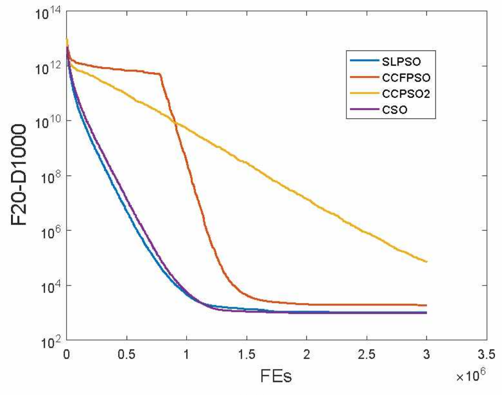
(a)



(b)



(c)



(d)

Figure 9: Convergence curves of CCFPSO, CCPSO2, CSO and SLPSO on $f15, f17, f18, f20$ - CEC2010

Table 5: Ranking results of 5 algorithms on CEC2008 test set

Fun	D	CCFPSO	CCPSO2	CSO	SLPSO	FuzzyPSO2
	100	3	4	1	2	5
f1	500	1	2	3	2	5
	1000	1	4	3	2	5
	100	3	2	4	1	5
f2	500	3	1	2	4	5
	1000	1	2	3	4	5
	100	4	5	1	3	2
f3	500	5	3	2	1	4
	1000	3	4	1	2	5
	100	2	1	3	4	5
f4	500	2	1	3	4	5
	1000	4	1	3	2	5
	100	2	3	1	4	5
f5	500	3	4	1	2	5
	1000	3	4	1	2	5
	100	3	4	1	2	5
f6	500	1	4	3	2	5
	1000	1	4	3	2	5
	100	1	2	3	4	5
f7	500	4	1	2	3	5
	1000	4	3	1	2	5
		2.57	2.81	2.14	2.57	4.81
final rank		2	3	1	2	4

algorithms. These studies reveal that CCFPSO substantially improved on the previously proposed FuzzyPSO2, significantly outperforming FuzzyPSO2 in 19 out of 21 functions in CEC2008. This improvement demonstrated that combining fuzzy logic for parameter adaptation with a cooperative coevolutionary framework is advantageous for addressing a wide range of large-scale problems, including their scalability. In addition, for the CEC2008 and CEC2010 test set, CCFPSO achieved the best results on separable functions, on relatively complex partially-separable functions, and performed reasonably well on non-separable functions. CCFPSO was the second-best algorithm of the five algorithms compared for CEC2008 and the third-best of the four algorithms compared for CEC2010. Therefore, CCFPSO proved to be competitive in tackling high-dimensional problems. In the future, more intelligent strategies could be used for the selection of the size of the decomposition groups, especially for completely non-separable functions. The incorporation of CC into other evolutionary and swarm intelligence algorithms that dynamically adapt their parameters using a fuzzy system could also be studied.

Competing interests

The authors have declared that no competing interests exist.

Authors' contribution

FP wrote the algorithm code, conducted the experiments, analyzed the results and wrote the manuscript; GL and EMM analyzed the results and revised the manuscript. All authors read and approved the final manuscript.

Table 6: Ranking results of 4 algorithms on CEC2010 test set

Fun	CCFPSO	CCPSO2	CSO	SLPSO
f1	1	4	3	2
f2	1	2	4	3
f3	1	4	3	2
f4	4	3	1	2
f5	4	3	1	2
f6	4	3	2	1
f7	4	3	1	2
f8	4	3	1	2
f9	3	4	2	1
f10	1	2	4	3
f11	4	3	2	1
f12	1	2	3	4
f13	3	4	1	2
f14	3	4	2	1
f15	1	4	2	3
f16	4	3	2	1
f17	1	2	3	4
f18	3	4	1	2
f19	3	4	1	2
f20	3	4	1	2
	2.65	3.25	2	2.1
final rank	3	4	1	2

References

[1] K. Tang, X. Yáo, P. N. Suganthan, C. MacNish, Y.-P. Chen, C.-M. Chen, and Z. Yang, "Benchmark functions for the cec'2008 special session and competition on large scale global optimization," *Nature inspired computation and applications laboratory, USTC, China*, vol. 24, pp. 1–18, 2007.

[2] 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," tech. rep., Nature Inspired Computation and Applications Laboratory, 2009.

[3] L. Li, W. Fang, Y. Mei, and Q. Wang, "Cooperative coevolution for large-scale global optimization based on fuzzy decomposition," *Soft Computing*, vol. 25, no. 5, pp. 3593–3608, 2021.

[4] J. H. Holland, "Genetic algorithms," *Scientific american*, vol. 267, no. 1, pp. 66–73, 1992.

[5] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[6] R. Eberhart and J. Kennedy, "Particle swarm optimization," in *Proceedings of the IEEE international conference on neural networks*, vol. 4, pp. 1942–1948, Citeseer, 1995.

[7] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks*, vol. 4, pp. 1942–1948, IEEE, 1995.

[8] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 1, pp. 29–41, 1996.

[9] L. Cui, G. Li, Y. Luo, F. Chen, Z. Ming, N. Lu, and J. Lu, “An enhanced artificial bee colony algorithm with dual-population framework,” *Swarm and Evolutionary Computation*, vol. 43, pp. 184–206, 2018.

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

[11] J.-R. Jian, Z.-H. Zhan, and J. Zhang, “Large-scale evolutionary optimization: a survey and experimental comparative study,” *International Journal of Machine Learning and Cybernetics*, vol. 11, no. 3, pp. 729–745, 2020.

[12] M. A. Potter and K. A. De Jong, “A cooperative coevolutionary approach to function optimization,” in *International Conference on Parallel Problem Solving from Nature*, pp. 249–257, Springer, 1994.

[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] Z. Yang, K. Tang, and X. Yao, “Multilevel cooperative coevolution for large scale optimization,” in *2008 IEEE congress on evolutionary computation (IEEE World Congress on Computational Intelligence)*, pp. 1663–1670, IEEE, 2008.

[15] X. Li and X. Yao, “Cooperatively coevolving particle swarms for large scale optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 2, pp. 210–224, 2011.

[16] M. N. Omidvar, X. Li, and X. Yao, “Cooperative co-evolution with delta grouping for large scale non-separable function optimization,” in *IEEE congress on evolutionary computation*, pp. 1–8, IEEE, 2010.

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

[18] R. Cheng and Y. Jin, “A social learning particle swarm optimization algorithm for scalable optimization,” *Information Sciences*, vol. 291, pp. 43–60, 2015.

[19] R. Cheng and Y. Jin, “A competitive swarm optimizer for large scale optimization,” *IEEE transactions on cybernetics*, vol. 45, no. 2, pp. 191–204, 2014.

[20] J.-i. Kushida, A. Hara, and T. Takahama, “Rank-based differential evolution with multiple mutation strategies for large scale global optimization,” in *2015 IEEE Congress on Evolutionary Computation (CEC)*, pp. 353–360, IEEE, 2015.

[21] F. Valdez, P. Melin, and O. Castillo, “A survey on nature-inspired optimization algorithms with fuzzy logic for dynamic parameter adaptation,” *Expert systems with applications*, vol. 41, no. 14, pp. 6459–6466, 2014.

[22] F. Olivas and O. Castillo, “Particle swarm optimization with dynamic parameter adaptation using fuzzy logic for benchmark mathematical functions,” in *Recent Advances on Hybrid Intelligent Systems*, pp. 247–258, Springer, 2013.

[23] F. Olivas, F. Valdez, and O. Castillo, “Particle swarm optimization with dynamic parameter adaptation using interval type-2 fuzzy logic for benchmark mathematical functions,” in *2013 World Congress on Nature and Biologically Inspired Computing*, pp. 36–40, IEEE, 2013.

[24] F. Paz, G. Leguizamón, and E. Mezura-Montes, “Particle swarm optimization with adaptive inertia weight using fuzzy logic for large-scale problems,” in *XXVI Congreso Argentino de Ciencias de la Computación (CACIC)(Modalidad virtual, 5 al 9 de octubre de 2020)*, 2020.

[25] F. Valdez, J. C. Vazquez, P. Melin, and O. Castillo, “Comparative study of the use of fuzzy logic in improving particle swarm optimization variants for mathematical functions using co-evolution,” *Applied Soft Computing*, vol. 52, pp. 1070–1083, 2017.

[26] J. Kennedy and R. Mendes, “Population structure and particle swarm performance,” in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC’02 (Cat. No. 02TH8600)*, vol. 2, pp. 1671–1676, IEEE, 2002.

[27] Y. Shi and R. C. Eberhart, “Parameter selection in particle swarm optimization,” in *International conference on evolutionary programming*, pp. 591–600, Springer, 1998.

[28] M. R. Ullmann, K. F. Pimentel, L. A. de Melo, G. da Cruz, and C. Vinhal, “Comparison of pso variants applied to large scale optimization problems,” in *2017 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*, pp. 1–6, IEEE, 2017.

[29] R. C. Eberhart, Y. Shi, and J. Kennedy, *Swarm intelligence*. Elsevier, 2001.

[30] F. Valdez, P. Melin, and O. Castillo, “An improved evolutionary method with fuzzy logic for combining particle swarm optimization and genetic algorithms,” *Applied Soft Computing*, vol. 11, no. 2, pp. 2625–2632, 2011.

[31] J. Perez, F. Valdez, O. Castillo, P. Melin, C. Gonzalez, and G. Martinez, “Interval type-2 fuzzy logic for dynamic parameter adaptation in the bat algorithm,” *Soft Computing*, vol. 21, no. 3, pp. 667–685, 2017.

[32] A. Sombra, F. Valdez, P. Melin, and O. Castillo, “A new gravitational search algorithm using fuzzy logic to parameter adaptation,” in *2013 IEEE congress on evolutionary computation*, pp. 1068–1074, IEEE, 2013.

[33] M. S. Norouzzadeh, M. R. Ahmadzadeh, and M. Palhang, “Ladpso: using fuzzy logic to conduct pso algorithm,” *Applied Intelligence*, vol. 37, no. 2, pp. 290–304, 2012.

[34] P. Ochoa, O. Castillo, and J. Soria, “Differential evolution using fuzzy logic and a comparative study with other metaheuristics,” in *Nature-inspired design*

of hybrid intelligent systems, pp. 257–268, Springer, 2017.

[35] S. Kumar and D. Chaturvedi, "Tuning of particle swarm optimization parameter using fuzzy logic," in *2011 International Conference on Communication Systems and Network Technologies*, pp. 174–179, IEEE, 2011.

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

[37] Y. Sun, M. Kirley, and S. K. Halgamuge, "Extended differential grouping for large scale global optimization with direct and indirect variable interactions," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 313–320, 2015.

[38] 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)*, vol. 42, no. 2, pp. 1–24, 2016.

[39] Y. Sun, X. Li, A. Ernst, and M. N. Omidvar, "Decomposition for large-scale optimization problems with overlapping components," in *2019 IEEE congress on evolutionary computation (CEC)*, pp. 326–333, IEEE, 2019.

[40] M. A. Meselhi, S. M. Elsayed, R. A. Sarker, and D. L. Essam, "Contribution based co-evolutionary algorithm for large-scale optimization problems," *IEEE Access*, vol. 8, pp. 203369–203381, 2020.

[41] M. Clerc, "The swarm and the queen: towards a deterministic and adaptive particle swarm optimization," in *Proceedings of the 1999 congress on evolutionary computation-CEC99*, pp. 1951–1957, IEEE, 1999.

[42] F. Olivas, F. Valdez, O. Castillo, C. I. Gonzalez, G. Martinez, and P. Melin, "Ant colony optimization with dynamic parameter adaptation based on interval type-2 fuzzy logic systems," *Applied Soft Computing*, vol. 53, pp. 74–87, 2017.

[43] S.-Z. Zhao, J. J. Liang, P. N. Suganthan, and M. F. Tasgetiren, "Dynamic multi-swarm particle swarm optimizer with local search for large scale global optimization," in *2008 IEEE congress on evolutionary computation (IEEE world congress on computational intelligence)*, pp. 3845–3852, IEEE, 2008.

[44] J. Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3–18, 2011.

[45] M. López-Ibánez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43–58, 2016.

Citation: F. Paz, G. Leguizamón and E. Mezura Montes. *Cooperative Coevolutionary Particle Swarms using Fuzzy Logic for Large Scale Optimization.* Journal of Computer Science & Technology, vol. 21, no. 2, pp. 112-125, 2021.
DOI: 10.24215/16666038.21.e11
Received: February 19, 2021 **Accepted:** June 23, 2021.