A Bi-population PSO with a Shake-Mechanism for Solving Constrained Numerical Optimization

Leticia C. Cagnina Susana C. Esquivel Carlos A. Coello Coello

Abstract— **This paper presents an enhanced Particle Swarm Optimizer approach, which is designed to solve numerical constrained optimization problems. The approach uses a single method to handle different types of constraints (linear, nonlinear, equality or inequality) and it incorporates a *shake-mechanism* and a dual population in an attempt to overcome the problem of premature convergence to local optima. The proposed algorithm is validated using standard test functions taken from the specialized literature and is compared with respect to algorithms representative of the state-of-the-art in the area. Our preliminary results indicate that our proposed approach is a highly competitive alternative to solve constrained optimization problems.**

I. Introduction

The solution of constrained optimization problems using metaheuristics (particularly, evolutionary algorithms) has attracted a lot of interest in the last few years, since such problems are very common in real-world applications. One of the many metaheuristics that has been adopted to solve such problems is particle swarm optimization (PSO) [13]. PSO has been found to be highly competitive for solving unconstrained optimization problems [14], [15], [9], [3], [4]. However, its use in constrained optimization problems is still relatively scarce (see for example [19], [18], [16]).

PSO was conceived as a simulation of individual and social behavior [12] such as the one observed in flocks of birds and fish. PSO explores the search space using a population of individuals, and the best performers (either within a group or with respect to the entire population) affect the performance of the others. Each individual is named *particle* and represents a possible solution within a multidimensional search space. The particles have their own position and velocity, which are constantly updated. They record their past behavior and use it to move towards promising regions of the search space.

In this paper, we present a PSO algorithm which is designed to solve constrained optimization problems. For that sake, our approach contains a constraint-handling technique as well as a mechanism to update the velocity and position of the particles [2], which is extended by adding to it a bi-population and a *shake-mechanism* as a way to avoid premature convergence.

Leticia C. Cagnina and Susana C. Esquivel are with LIDIC (Research Group). Universidad Nacional de San Luis - Ej. de Los Andes 950 - (D5700HHW) San Luis, ARGENTINA. (emails: {lcagnina,esquivel}@unsl.edu.ar). Carlos A. Coello Coello is with CINVESTAV-IPN (Evolutionary Computation Group), Departamento de Computacion, Av. IPN No. 2508, Col. San Pedro Zacatenco, Mexico D.F. 07300, MEXICO. (email: ccoello@cs.cinvestav.mx).

The remainder of this paper is organized as follows. Section II provides the statement of general constrained optimization problems. Section III briefly discusses the previous related work. In Section IV, we describe in detail our proposed approach. Section V describes the experimental setup and provides an analysis of the results obtained from our empirical study. The conclusions and some directions for future research are stated in Section VII.

II. Statement of the Problem

Without loss of generality, we can consider the general nonlinear optimization problem as a minimization problem, which can be formally stated as the problem of finding $\vec{x}$ which:

$$minf(\vec{x}) \text{ with } \vec{x} = (x_1, x_2, \ldots, x_D) \in \mathcal{F} \subseteq \mathcal{S} \subseteq \mathcal{R}^D \quad (1)$$

subject to:

$$g_i(\vec{x}) \leq 0 \quad i = 1, 2, \ldots, n \ . \quad (2)$$

$$h_e(\vec{x}) = 0 \quad e = 1, 2, \ldots, m \ . \quad (3)$$

Each $x_d \in [l_d, u_d]$ with $d \in [1..D]$. The l_d and u_d are the lower and upper bounds imposed on the decision variables. The g_i and h_e functions are defined on $\mathcal{S}$ (search space), and correspond to the inequality and equality constraint functions, respectively. A constraint delimits the search space splitting it into a feasible and an infeasible region. $\mathcal{S}$ is a D-dimensional rectangle defined by the lower and upper bounds of each variable x_d. All $\vec{x}$ satisfying all inequality and equality constraint functions determine the feasible solution space $\mathcal{F}$.

III. Previous Related Work

As indicated before, despite its success in a variety of optimization problems, the proposals of constraint-handling mechanisms for PSO are relatively scarce. Next, we will review the most representative work done in this area.

Zang et al. [23] presented a PSO algorithm that implements a periodic mechanism for handling constraints. The approach makes periodic copies of the search space when the algorithm starts the run, thus avoiding the disorganization that might arise when the mutation operator is applied to those particles lying on the boundary between the feasible and infeasible regions. The authors validated their algorithm with eight test functions and compared their results with respect to whose provided by conventional constraint-handling methods.

Toscano Pulido and Coello Coello [19] added to a basic PSO a simple mechanism for tackling constraints based on how close are the particles from the feasible region. A turbulence operator was incorporated in order to improve the exploration of the search space. The thirteen well-known constrained test functions from [21] were used to show the performance of this PSO algorithm. The authors concluded that their results were highly competitive.

Liang and Suganthan [16] proposed a novel constraint-handling mechanism based on multi-swarms (sub-populations). Each swarm works on a constraint, but these are assigned in an adaptive manner, and are periodically changed. Additionally, a local search mechanism is combined with the algorithm in order to improve the overall quality of the search performed. The authors validated their approach using twenty four test function taken from [8] and expressed their results in terms of Best functions error values. The authors reported that their results were satisfactory for 21 test functions, but the approach did not properly work in the other three.

Muñoz-Zavala et al. [18] presented a standard PSO algorithm improved with two perturbation operators that are only applied to the best particle population (pbest) at two different stages. These operators are used to keep the algorithm's exploration ability. To handle the constraints, they used the summation of constraint violations and a dynamic tolerance factor for the equality constraints (i.e., a higher tolerance is allowed at the beginning of the search, and such value is decreased along the run). The algorithm also maintains a "tolerant file", that is used when the perturbation operators are applied and the tolerance factor is decremented, in order to determine a better leader for the swarm. The authors also used the twenty four test functions from [8] to validate their approach. The results were found to be competitive, but the approach did not properly work in one of the test functions.

The algorithm CPSO-shake, which is proposed in this paper, shares some features with some of the algorithms cited above: the constraint-handling mechanism, the variant tolerance factor and the subpopulation concept. However, as we will see later on, the last two concepts are used in a different manner.

IV. CPSO-SHAKE ALGORITHM

In this section, we describe in detail our proposed approach, which we call the CPSO-shake algorithm.

A. General Model

As stated before, a PSO algorithm operates on a population of particles. Due to the type of problem to optimize (with n decision variables), the particles are n-dimensional real number vectors. The best position found so far for the particles (for the *gbest* model) or in the neighborhood (*lbest* model) is recorded. The best value reached by each particle (*pbest*) is stored, too. As in the standard model, the particles evolve using two update formulas, one for position and another one for velocity.

B. Particular Model

As it was stated in some of our previous work [1], the *gbest* model tends to converge to a local optimum; however, this model works well in a variety of problems. Motivated by this, we proposed a formula to update the velocity, using a combination of both the *gbest* and the *lbest* models [2]. Such a formula is adopted here as well, and is shown in equation (4).

$$\begin{aligned} v_{id} = w(v_{id} + \gamma_1(p_{id} - par_{id}) + \\ \gamma_2(p_{ld} - par_{id}) + \gamma_3(p_{gd} - par_{id})) \end{aligned} \tag{4}$$

where v_{id} is the velocity of particle i at the dimension d; w is the inertia factor [5] whose goal is to balance global exploration and local exploitation. γ_1 is the personal learning factor, and γ_2 and γ_3 are the social learning factors. These 3 values are multiplied by 3 different random numbers within the range [0..1], p_{id} is the best position reached by the particle i; p_{ld} is the best position reached by any particle in the neighborhood, p_{gd} is the best position reached by any particle in the swarm. par_{id} is the value of the particle i at the dimension d.

But in [2] we modified the equation for updating the particles. In that paper, during 10% of the iterations, we applied the normal formula (depicted in equation (5)) as suggested in [13].

$$par_{id} = par_{id} + v_{id} \tag{5}$$

And in the remainder 90% of cases, we used equation (6) proposed by Kennedy [11].

$$par_i = N\left(\frac{p_i + p_l}{2}, |p_i - p_l|\right) \tag{6}$$

where p_i is the position of the particle to be updated, N is the Gaussian random generator, p_i and p_l are the best position reached by the particle par_i and the best position reached by any particle in the neighborhood of par_i, respectively. That probability was empirically found to be the best after performing a series of experiments with all the test functions evaluated.

We use a circle topology [10] to compute the p_{ld} value, in which each particle is connected to k neighbors. The neighbors are determined by the position of the particles in the storage structure. Figure 1 illustrates this concept.

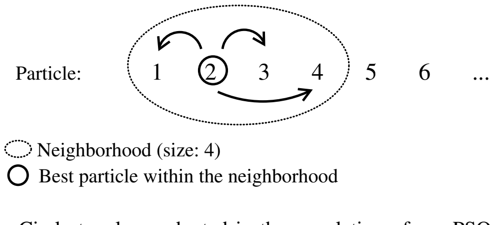

Fig. 1. Circle topology adopted in the population of our PSO approach

To help avoiding convergence to a local optimum, we used a dynamic mutation operator [1] which is applied to each individual with a pm-probability. This value is calculated considering the total number of iterations in the algorithm (*cycles*) and the current cycle number as the following equation indicates:

$$pm = max\_pm - \frac{max\_pm - min\_pm}{max\_cycle} * current\_cycle \quad (7)$$

where max_pm and min_pm are the maximum and minimum values that pm can take, max_cycle is the total number of cycles that the algorithm will iterate, and $current\_cycle$ is the current cycle in the iterative process.

C. Handling Constraints

The constraint-handling scheme adopted in this paper is the simplest method possible to handle both equality and inequality constraints: *we prefer a feasible particle over an infeasible one*. When the two particles compared are infeasible, we choose the one which is closer to the feasible region. In order to do that, our algorithm stores the largest violation obtained for each constraint. When an individual is found infeasible, the amount of violation (this value is normalized with respect to the largest violation stored so far) is added. This strategy is used when the *pbest*, *gbest* and *lbest* particles are chosen. All equality constraints were transformed into inequalities, as is normally done in the evolutionary optimization literature, and we used a *tolerance factor* (ϵ) defined by the following formula:

$$|h_e(\vec{x})| - \epsilon \leq 0 \quad (8)$$

D. Dynamic Tolerance

Our algorithm has a mechanism to adapt the value of ϵ during the run. When the search process starts, the value is initialized in 0.1. As we increase the number of iterations, the value of ϵ is divided by 10 at three different moments (i.e., ϵ takes the values: 0.01, 0.001 and 0.0001 during a run). If we assume that N is the total number of iterations to be performed, the value of ϵ will change according to the scheme graphically shown in Figure 2.

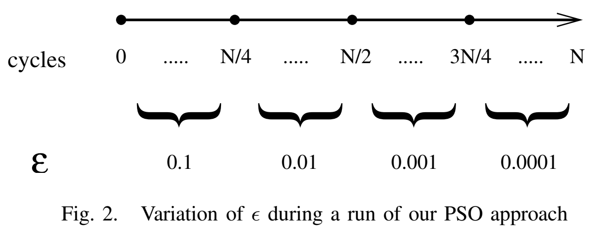

Fig. 2. Variation of ϵ during a run of our PSO approach

The main advantage of using a varying ϵ value is to favor the existence of feasible solutions at the beginning of the search process, so that the search space can be properly sampled (particularly in problems having equality constraints). As the number of iterations increases, ϵ is decreased, so that our approach starts converging towards solutions that satisfy the equality constraints with a higher numerical precision.

E. Bi-population

The CPSO-shake algorithm splits the entire population into two subpopulations each of which is independently evolved. The idea is to maintain more than one group of particles exploring the search space (at the same time). In that way the possibility of falling into local optima is reduced.

One may then wonder why to adopt only two subpopulations and not more. The reason is that it does not make any sense to adopt more than two subpopulations, considering the small number of particles that we use in our original population (only 10). In fact, we believe that our neighborhood topology would not work properly if we adopt less than 5 particles and therefore our choice of adopting only two subpopulations.

All the features stated before for the entire population (neighborhoods, *lbest* and *gbest* approaches, equations for updating the velocity and the positions) still apply, but in this case, they are applied not to a single population, but to each subpopulation. When the iterative process finishes, the best particle from both subpopulations is reported as the final output.

F. Shake-Mechanism

In our PSO-based proposal reported in [2], we had some stagnation problems when trying to obtain values close to the optima for some difficult test functions. In order to overcome this problem, the algorithm reported in this paper incorporates a *shake-mechanism*. This mechanism is applied when the percentage of infeasible individuals is higher than 10%. This value was empirically derived. Note that it is not convenient to keep populations in which all the solutions are feasible, since infeasible solutions play an important role when trying to solve problems with active constraints [17].

In order to implement the shake-mechanism, some particles are moved to another place in the search space. Although this can be done by guiding a particle to a random direction, it is undesirable that the particles move away too much (we just want to shake them a little!). So, a particle with a good solution is selected as a reference: a randomly chosen *pbest* particle. Thus, equation (9) is used to move a particle i:

$$v_{id} = w * v_{id} + \gamma_1 * (psel_d) \quad (9)$$

where v_{id} is the dth-position of the velocity vector, w is the inertia factor, γ_1 is the personal learning factor multiplied by a random number within the range [0..1]. $psel_d$ is the dth-position of a (randomly chosen) *pbest* vector.

The shake-mechanism is applied with a 50% probability.

G. CPSO-shake Pseudocode

Figure 3 shows the pseudocode of our proposed CPSO-shake algorithm. At the beginning of the search, we initialize the vectors of position and velocity of each particle in both subpopulations and we initialize the ϵ value, too (lines 2 to 6).

After evaluating the particles and obtaining the best values: *pbest*, *lbest* and *gbest* (lines 7 and 8), the subpopulations begin to evolve. During the evolutionary process, new values of *pbest*, *lbest* and *gbest* are chosen and both, the velocity and the position of each particle are updated (lines 10 to 25). At line 26, a *keeping* mechanism is applied to control that all the dimensions in all particles are within the allowable bounds. We then calculate the percentage of infeasible individuals in the entire population (both subpopulations considered together) and apply the shake-mechanism if the required conditions are fulfilled (lines 27 to 30). The mutation probability is updated and the particles are mutated, if applicable (lines 31 and 32). After that, the particles are evaluated, new "best" values are recorded and the ϵ value is updated (lines 33 to 35). Finally, the best value reached by any subpopulation is taken and compared. The best of them is returned (lines 37 and 38).

```
0.  CPSO-shake:
1.  Swarm Initialization
2.    Initializate subpop1
3.    Initializate velocity for subpop1
4.    Initializate subpop2
5.    Initializate velocity for subpop2
6.    Init epsilon
7.    Evaluate fitness for each subpop
8.    Record pbest and gbest for each subpop
9.  Swarm flights through the search space
10.   DO
11.     FOR each subpop DO
12.         FOR i=1 TO numberOfparticles DO
13.             Search the best leader in the
14.                neighborhood of $part_i$
15.                and record in $lbest_i$
16.             FOR j=1 TO numberOfdimensions DO
17.                 Update $vel_{ij}$
18.                 IF flip(0.1)
19.                    Update $part_{ij}$ with eq.(3)
20.                 ELSE
21.                    Gaussian update with eq.(5)
22.                 END
23.             END
24.         END
25.     END
26.     Keeping particles
27.     Calculating % infeasibles
28.     IF % infeasibles > 10%
29.        Move particles
30.     END
31.     Update $pm$
32.     Mutate every particle depending on $pm$
33.     Evaluate fitness($part_i$)
34.     Record pbest and gbest
35.     Update epsilon
36.   WHILE($current\_cycle < max\_cycle$)
37.   result=BEST($best\_subpop1, best\_subpop2$)
38.   RETURN(result)
```

Fig. 3. Pseudo-code of CPSO*shake*

V. Experimental Study

For validating our proposed approach, we adopted 20 test problems taken from [8]. The detailed description of the test problems may be consulted in its original source [8].

TABLE I
Comparison of the **best** values obtained by our CPSO-shake and the Approximation Evolution Strategy which uses Stochastic Ranking (AESSR).

Funct.	Benchmark	CPSO-shake	AESSR
g1	-15.000	-15.000	-15.000
g2	-0.803	-0.803	-0.739
g3	-1.000	-1.000	-1.000
g4	-30665.539	-30665.538	-30665.539
g5	5126.496	5126.498	5126.497
g6	-6961.813	-6961.825	-6961.814
g7	24.306	24.309	24.306
g8	-0.095	-0.095	-0.096
g9	680.630	680.630	680.630
g10	7049.248	7049.285	7049.408
g11	0.749	0.749	0.750
g12	-1.000	-1.000	-1.000
g13	0.053	0.054	0.054
g14	-47.764	-47.635	-47.765
g15	961.715	961.715	961.715
g16	-1.905	-1.905	-1.905
g17	8853.539	8853.539	8853.540
g18	-0.866	-0.866	-0.866
g19	32.655	34.018	32.665
g20	0.204	0.203	-

All these test functions were transformed into minimization problems. All the equality constraints were transformed into inequalities. We performed 25 independent runs per problem, with a total of 350,000 evaluations of the objective function per run. Our proposed CPSO-shake used the following parameters: swarm size = 10 particles, $pm\_min = 0.1$, $pm\_max = 0.4$, neighborhood size = 3, inertia factor w = 0.8, personal learning factor and social learning factors for γ_1, γ_2 and γ_3 was set to 1.8. The parameter settings such as swarm size, mutation probability, neighborhood size and learning factors were empirically derived after numerous experiments.

Our results were compared with respect to those obtained by an Approximation Evolution Strategy which uses Stochastic Ranking (AESSR) recently proposed in [20] and with respect to the results produced by the Stochastic Ranking (SR) approach [21]. Both approaches are highly competitive and are representative of the state-of-the-art in the area. Note that in [20], the author validated his approach using 500,000 objective function evaluations per run, while our approach only performs 350,000 evaluations.

Table I shows the best values found by CPSO-shake and AESSR. The conclusion is the following: Despite the fact that the number of evaluations performed by our CPSO-shake approach is 30% lower than the number of evaluations performed by AESSR, our CPSO-shake algorithm obtained results that are very satisfactory. For five functions (g2, g8, g10, g11 y g17) the values obtained by CPSO-shake are better than those obtained by AESSR. In 10 test functions, the results are the same and only for 4 test functions, AESSR obtained better values than CPSO-shake. All the solutions found by both algorithms are feasible, except for the value shown for g20 which is infeasible but is very close to the

TABLE II

BEST VALUES OBTAINED WITH **CPSO-SHAKE** PERFORMING 350,000 OBJECTIVE FUNCTION EVALUATIONS.

Funct.	Benchmark	Best	Mean	Worst
g1	-15.000	-15.000	-15.000	-88.844
g2	-0.803	-0.803	-0.796	-0.064
g3	-1.000	-1.000	-1.000	-1.005
g4	-30665.538	-30665.538	-30646.178	-25186.227
g5	5126.496	5126.498	5240.496	4153.354
g6	-6961.813	-6961.825	-6859.075	-6482.554
g7	24.306	24.309	24.912	47.401
g8	-0.095	-0.095	-0.095	-72.264
g9	680.630	680.630	681.373	8825.132
g10	7049.248	7049.285	7850.401	1810.354
g11	0.749	0.749	0.749	1.361
g12	-1.000	-1.000	-1.000	-0.718
g13	0.053	0.054	0.450	1.082
g14	-47.764	-47.635	-45.665	-517.919
g15	961.715	961.715	962.516	768.737
g16	-1.905	-1.905	-1.795	0.179
g17	8853.539	8853.539	8894.708	9003.614
g18	-0.866	-0.866	-0.787	5.438
g19	32.655	34.018	64.505	55.835
g20	0.204	0.203	0.416	17.048

TABLE III

BEST VALUES OBTAINED WITH **AESSR** PERFORMING 500,000 OBJECTIVE FUNCTION EVALUATIONS.

Funct.	Benchmark	Best	Mean	Worst
g1	-15.000	-15.000	-15.000	-15.000
g2	-0.803	-0.739	-0.697	-0.657
g3	-1.000	-1.000	-1.000	-1.000
g4	-30665.538	-30665.539	-30665.539	-30665.539
g5	5126.496	5126.497	5126.497	5126.497
g6	-6961.813	-6961.814	-6961.814	-6961.814
g7	24.306	24.306	24.307	24.312
g8	-0.095	-0.096	-0.096	-0.096
g9	680.630	680.630	680.630	680.630
g10	7049.248	7049.408	7061.087	7126.958
g11	0.749	0.750	0.750	0.750
g12	-1.000	-1.000	-1.000	-1.000
g13	0.053	0.054	0.116	0.439
g14	-47.764	-47.765	-47.762	-47.753
g15	961.715	961.715	961.715	961.715
g16	-1.905	-1.905	-1.905	-1.905
g17	8853.539	8853.540	8853.717	8857.816
g18	-0.866	-0.866	-0.851	0.675
g19	32.655	32.665	32.829	33.242
g20	0.204	-	-	-

solution (which is also infeasible) reported as the best known for this test function in [8].

Table II shows the Best, Mean and Worst values obtained by CPSO-shake for the 20 constrained test functions adopted in our empirical study.

Table III shows the Best, Mean and Worst values obtained by AESSR for the 20 constrained test functions adopted for our empirical study.

We also compared results with respect to the SR algorithm, using the same number of evaluations of the objective function (350,000). We downloaded the source code of the SR algorithm from Runarsson's web page [7] and ran it ourselves in order to perform a direct comparison of results. We had to implement the 7 last functions (from g14 to g20) and re-run the algorithm, since such functions were not available in the original source code. To implement those 7 functions we used the code available at [8]. The results obtained are compared with respect to our CPSO-shake and shown in Table IV. The best, mean and worst values obtained by SR are shown in Table V.

TABLE IV

COMPARISON OF THE **best** VALUES OBTAINED BY OUR CPSO-SHAKE AND STOCHASTIC RANKING (SR).

Funct.	Benchmark	CPSO-shake	SR
g1	-15.000	-15.000	-15.000
g2	-0.803	-0.803	-0.803
g3	-1.000	-1.000	-1.000
g4	-30665.539	-30665.538	-30665.539
g5	5126.496	5126.498	5126.497
g6	-6961.813	-6961.825	-6961.814
g7	24.306	24.309	24.310
g8	-0.095	-0.095	-0.095
g9	680.630	680.630	680.630
g10	7049.248	7049.285	7050.194
g11	0.749	0.749	0.749
g12	-1.000	-1.000	-1.000
g13	0.053	0.054	0.053
g14	-47.764	-47.635	-41.551
g15	961.715	961.715	961.715
g16	-1.905	-1.905	-1.905
g17	8853.539	8853.539	8811.692
g18	-0.866	-0.866	-0.866
g19	32.655	34.018	33.147
g20	0.204	0.203	0.057

Table IV shows the best values found by our CPSO-shake and SR. The results of both algorithms are comparable: in 11 test functions both approaches reached the optimum (or best known solution). In 5 test functions (g7, g10, g14, g17 y g20), our CPSO-shake overperforms the SR algorithm, while in other 4 test functions (g4, g5, g6, g13) SR outperforms our CPSO-shake. However, it is important to emphasize that the previous version of CSPO, reported in [2], was able to outperforms SR only in one test function (g10). This is a clear indication of the improvements achieved by the new version of our PSO-based approach presented in this paper.

Figures 4, 5 and 6 show how far are the best values found by the algorithms with respect to the values reported to be the optima or best known for each test function. In particular, Figure 4 shows the best performance of the new algorithm presented in this paper with respect to our previous version, reported in [2].

VI. STATISTICAL ANALYSIS

Another way to compare the performance of our algorithms is with a statistical test. The comparison with AESSR is not possible because we do not have the best values reached at each run. So, we performed an analysis of variance between SR and CPSO-shake using the best values of the 25 independent runs performed with each one. We applied the Kruskal-Wallis [6] nonparametric one-way analysis because

TABLE V

Best Values obtained with **SR** performing 350,000 objective function evaluations.

Funct.	Benchmark	Best	Mean	Worst
g1	-15.000	-15.000	-15.000	-15.000
g2	-0.803	-0.803	-0.784	-0.734
g3	-1.000	-1.000	-1.000	-1.000
g4	-30665.538	-30665.539	-30665.480	-30664.216
g5	5126.496	5126.497	5130.752	5153.757
g6	-6961.813	-6961.814	-6863.645	-6267.787
g7	24.306	24.310	24.417	24.830
g8	-0.095	-0.095	-0.095	-0.095
g9	680.630	680.630	680.646	680.697
g10	7049.248	7050.194	7423.434	8867.844
g11	0.749	0.750	0.750	0.751
g12	-1.000	-1.000	-1.000	-1.000
g13	0.053	0.053	0.061	0.128
g14	-47.764	-41.551	-41.551	-40.125
g15	961.715	961.715	961.731	962.008
g16	-1.905	-1.905	-1.703	-1.587
g17	8853.539	8811.692	8805.990	8559.613
g18	-0.866	-0.866	-0.786	-0.457
g19	32.655	33.147	34.337	37.477
g20	0.204	0.057	0.052	0.062

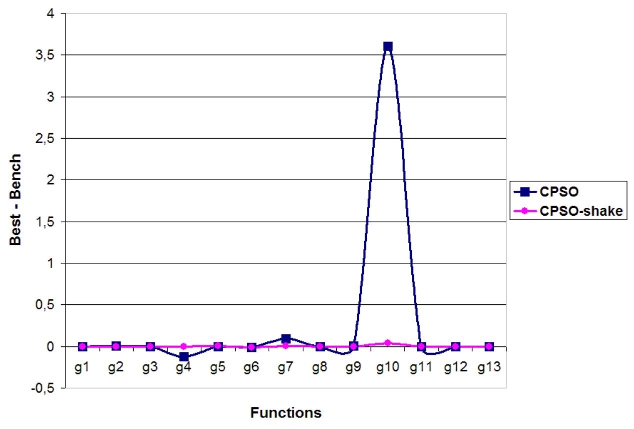

Fig. 4. Comparison between our original CPSO [2] and our CPSO-shake (this paper)

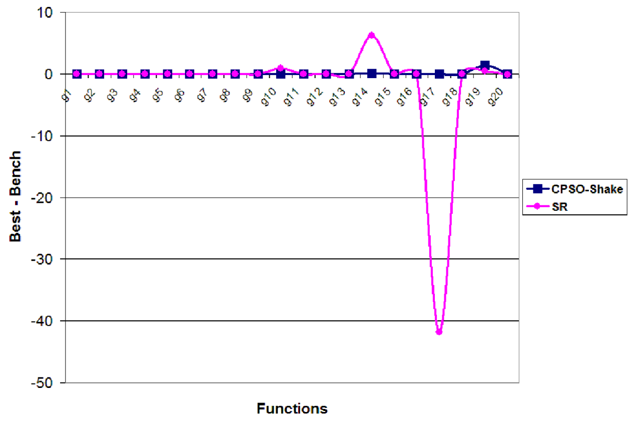

Fig. 5. Comparison between our CPSO-shake (this paper) and Stochastic Ranking (SR) [21]

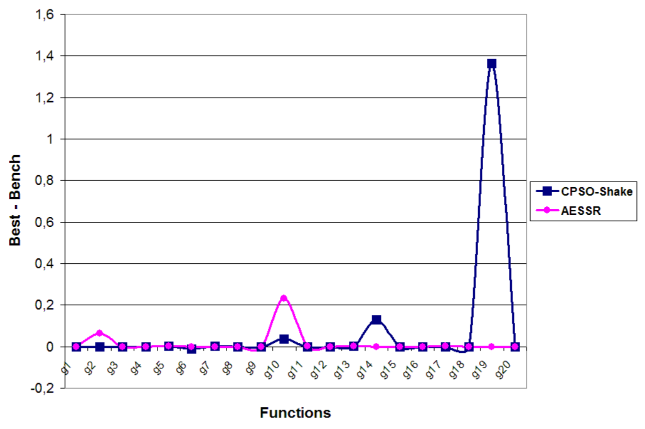

Fig. 6. Comparison between our CPSO-shake (this paper) and Approximate Evolution Strategy with Stochastic Ranking (AESSR) [20]

the values (the sample) do not have a normal distribution (determined with the Kolmogorov-Smirnov test).
The Kruskal-Wallis test returns the *p-values* for the null hypothesis for all samples. If the *p-values* are near zero, that suggests that at least one sample is significantly different (or *statistically significant*) than the other samples. Usually, if *p-values* are less than 0.05, we declare that the results are significant.
Table VI shows the *p-value* for each function. The results indicate the values reached with CPSO-shake (for all functions except: g5, g12 and g16), are statistically significant from those of SR.
For those functions significantly different we apply the Tukey test to determine under which experimental conditions the differences are significant. The test returns: an estimate value (EV) into a range ([LI,LS]). If the range does not contain the zero-value, then the results are confirmed (significantly different). As we observe in Table VI, for every function statistically significant, the ranges do not have the zero-value.

VII. Conclusions

In this paper, we have introduced an enhanced PSO algorithm called CPSO-shake, which is proposed to solve constrained numerical optimization problems. The results reached by CPSO-shake are very competitive with respect to Stochastic Ranking (which is one of the best constraint-handling techniques known to date) and to Approximation Evolution Strategy which uses Stochastic Ranking (recently proposed [20]) which is also a very competitive constraint-handling technique. Our preliminary results indicate that CPSO-shake is at least comparable to Stochastic Ranking and is also very satisfactory with respect to AESSR, since it obtained similar results despite performing 30% less objective function evaluations than AESSR. SR and AESSR presented a lower variability of the results than CPSO-shake, but this is due to the mechanisms implemented in our approach to maintain diversity. However, we argue that such mechanisms provide a tradeoff that we consider acceptable, since the best

TABLE VI

KRUSKAL-WALLIS' P-VALUES AND TUKEY'S RESULTS FOR SR VS CPSO-SHAKE

Funct.	*p-value*	LI	EV	LS
g1	9.00796e-011	17.4420	25.0000	32.5580
g2	0.0011	-20.9348	-13.0800	-5.2252
g3	1.28334e-010	17.3787	25.0000	32.6213
g4	9.08263e-011	-32.5598	-25.0000	-17.4402
g5	0.6735	-	-	-
g6	5.35295e-005	-22.2776	-15.0000	-7.7224
g7	8.32678e-007	-26.5573	-19.0000	-11.4427
g8	2.55962e-012	-31.9998	-25.0000	-18.0002
g9	5.98833e-009	-30.8563	-23.0800	-15.3037
g10	6.75438e-009	-30.7763	-23.0000	-15.2237
g11	0.0003	6.5685	14.2917	22.0149
g12	1	-	-	-
g13	3.35698e-010	-32.6957	-24.9200	-17.1443
g14	6.75279e-010	17.0609	25.0000	32.9391
g15	1.53751e-009	-32.7940	-24.7600	-16.7260
g16	0.2651	-	-	-
g17	3.60989e-010	-32.8147	-25.0000	-17.1853
g18	6.73625e-009	-30.7757	-23.0000	-15.2243
g19	3.3067e-010	-32.4829	-24.7600	-17.0371
g20	1.28627e-010	-32.6218	-25.0000	-17.3782

values found by our approach remained very competitive despite the larger variability of results obtained (we sacrificed some online performance for the sake of improving offline performance).

Our proposed approach is one of the most competitive constraint-handling techniques currently available for the PSO algorithm (both SR and AESSR use an evolution strategy [22] as their search engine).

As part of our future work, we aim to improve the robustness of our CPSO-shake algorithm, by introducing alternative mechanisms to maintain diversity. We believe that this goal is achievable, but a much more careful study of the behavior of our PSO-based approach in constrained search spaces is required for that sake.

ACKNOWLEDGMENT

The third author gratefully acknowledges support from CONACyT project no. 45683-Y.

REFERENCES

[1] L. Cagnina, S. Esquivel, and R. Gallard. Particle swarm optimization for sequencing problems: a case study. In *Congress on Evolutionary Computation*, pages 536–541, Portland, Oregon, USA, 2004. http://www.lidic.unsl.edu.ar/publicaciones/info_publicacion.php?id_publicacion=200.

[2] L. C. Cagnina, Susana C. Esquivel, and C. A. Coello Coello. A particle swarm optimizer for constrained numerical optimization. In *9th International Conference - Parallel problem Solving from Nature - PPSN IX*, pages 910–919, Reykjavik, Island, 2006.

[3] W. Cedeno and D. Agrafiotis. Particle swarms for drug design. In *Proceeding of the IEEE Congress on Evolutionary Computation 2005*, pages 479–486, Edinburgh, Scotland, 2005.

[4] E. Correa, A. Freitas, and C. Johnson. A new discrete particle swarm algorithm applied to attribute selection in a bioinformatics data set. In *Proceeding of the GECCO 2006*, pages 35–42, Seattle, Washington,USA, 2006.

[5] R. Eberhart and Y. Shi. A modified particle swarm optimizer. In *International Conference on Evolutionary Computation, IEEE Service Center*, Anchorage, AK, Piscataway, NJ, 1998.

[6] M. Hollander and D. A. Wolfe. *Nonparametric Statistical Methods*. Wiley, 1973.

[7] http://cerium.raunvis.hi.is/ tpr/software/sres/index.html.

[8] http://www.ntu.edu.sg/home/EPNSugan/index_files/CEC-06/CEC06.html.

[9] X. Hu, R. Eberhart, and Y. Shi. Swarm intelligence for permutation optimization: a case study on n-queens problem. In *Proceeding of the IEEE Swarm Intelligence Symposium*, pages 243–246, Indianapolis, Indiana, USA, 2003.

[10] J. Kennedy. Small world and mega-minds: effects of neighborhood topologies on particle swarm performance. In *1999 Congress on Evolutionary Computation*, pages 1931–1938, Piscataway, NJ, 1999. IEEE Service Center.

[11] J. Kennedy. Bores bones particle swarm. In *IEEE Swarm Intelligence Symposium*, pages 80–87, 2003.

[12] J. Kennedy and R. Eberhart. The particle swarm: social adaptation in information-processing systems. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Organization*, 1999.

[13] J. Kennedy and R. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, California, USA, 2001.

[14] J. Kennedy and W. Spears. Matching algorithm to problems: an experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator. In *IEEE Conference on Evolutionary Computation*, Anchorage, Alaska, 1998.

[15] R. Krohling, H. Knidel, and Y. Shi. Solving numerical equations of hydraulic problems using particle swarm optimization. In *Congress on Computational Intelligence*, pages 1968–1961, Honolulu, Haway, 2002.

[16] J. J. Liang and P. N. Suganthan. Dynamic multi-swarm particle swarm optimizer with a novel constraint-handling mechanism. In *IEEE Congress on Evolutionary Computation*, pages 316–323, Vancouver, Canada, 2006. IEEE Press.

[17] Efrén Mezura-Montes and Carlos A. Coello Coello. A Simple Multimembered Evolution Strategy to Solve Constrained Optimization Problems. *IEEE Transactions on Evolutionary Computation*, 9(1):1–17, February 2005.

[18] Angel E. Muñoz et al. PESO+ for Constrained Optimization. In *IEEE Congress on Evolutionary Computation*, pages 936–942, Vancouver, Canada, 2006.

[19] G. Toscano Pulido and C. A. Coello Coello. A constraint-handling mechanism for particle swarm optimization. In *IEEE Congress on Evolutionary Computation*, pages 1396–1403, Portland, Oregon, USA, 2004.

[20] T. P. Runarsson. Approximate evolution strategy using stochastic ranking. In *2006 IEEE World Congress on Computation Intelligence*, volume 3, pages 2760–2767, British Columbia, Canada, 2006.

[21] T. P. Runarsson and X. Yao. Stochastic ranking for constrained evolutionary optimization. In *IEEE Transactions on Evolutionary Computation*, volume 3, pages 284–294, 2000.

[22] Hans-Paul Schwefel. *Evolution and Optimum Seeking*. John Wiley & Sons, New York, 1995.

[23] W. Zang, X. Xie, and D. Bi. Handling boundary constraints for numerical optimization by particle swarm flying in periodic search spaces. In *IEEE Congress on Evolutionary Computation*, pages 2307–2311, Portland, Oregon, USA, 2004.