Hindawi Publishing Corporation
Journal of Engineering
Volume 2013, Article ID 435104, 13 pages
http://dx.doi.org/10.1155/2013/435104

Research Article

Particle Swarm Algorithms to Solve Engineering Problems: A Comparison of Performance

Giordano Tomassetti[1] and Leticia Cagnina[2]

[1] *ENEA C.R. Frascati, Via E. Fermi 45, 00044 Frascati, Italy*
[2] *LIDIC Research Group, Universidad Nacional de San Luis, Ej. de los Andes 950, 5700 San Luis, Argentina*

Correspondence should be addressed to Leticia Cagnina; lcagnina@unsl.edu.ar

Received 31 December 2012; Accepted 10 February 2013

Academic Editor: Yangmin Li

In many disciplines, the use of evolutionary algorithms to perform optimizations is limited because of the extensive number of objective evaluations required. In fact, in real-world problems, each objective evaluation is frequently obtained by time-expensive numerical calculations. On the other hand, gradient-based algorithms are able to identify optima with a reduced number of objective evaluations, but they have limited exploration capabilities of the search domain and some restrictions when dealing with noncontinuous functions. In this paper, two PSO-based algorithms are compared to evaluate their pros and cons with respect to the effort required to find acceptable solutions. The algorithms implement two different methodologies to solve widely used engineering benchmark problems. Comparison is made both in terms of fixed iterations tests to judge the solution quality reached and fixed threshold to evaluate how quickly each algorithm reaches near-optimal solutions. The results indicate that one PSO algorithm achieves better solutions than the other one in fixed iterations tests, and the latter achieves acceptable results in less-function evaluations with respect to the first PSO in the case of fixed threshold tests.

1. Introduction

Optimization is an interesting and crucial aspect in design processes, particularly those related to real world issues. The reason for this interest in practical optimization problems has to be found in the intensive computational effort frequently needed to evaluate different solutions. The evaluation of a large number of candidate solutions is sometimes unaffordable when dealing with extensive CPU times for each calculation. Many methods have been proposed to solve this kind of problems such as mathematical programming [1–3] and nonlinear programming [4–6]. Mathematical programming algorithms are not always possible to apply because, many times, the real world problems cannot be transformed into linear models neither can the nonlinear variables be transformed into polynomial functions. The applicability of Nonlinear Programming algorithms is limited to the availability of the first- or second-order derivatives of the real-world problem to solve. However, both kinds of methods constitute an efficient gradient-based optimization set of algorithms. These algorithms are strongly influenced by the choice of the starting points, the number of local optima, and shape of the peaks that the functions have. They also have difficulties when dealing with discontinuities and they show a reduced exploration capability of the search domain with respect to evolutionary algorithms. The inefficiencies and instability of these methods have forced researchers to consider another kind of algorithms such as the recent swarm intelligence (SI) methods like particle swarm optimization [7] (PSO).

PSO is a metaheuristic based on the observation of movement rules followed by a swarm of birds. It has been observed that movements of each individual are influenced by both the swarm leader and the personal experience of the agent itself. This social behaviour has been translated into a mathematical recursive scheme that has proven to be successful in the search process.

In many engineering design optimizations, direct optimizers, such as PSOs, do not directly call the simulation but rather evaluate a surrogate of the objective and constraints. Such approaches considerably reduce the total number of simulations required. Although surrogate modeling is a powerful tool to deal with time-consuming optimization problems, a number of drawbacks arise when using

metamodels. The sampling techniques should limit the number of expensive simulations but be able, at the same time, to estimate the values of constraints and objectives of various shapes in a large portion of the domain. Many different techniques, algorithms, and interpolating functions have been proposed to reproduce the real functions. Neural networks [18], for example, have been widely used for function approximations implemented as meta-models in case of time-consuming simulations. Anyway, most metamodels include some sort of surrogate update procedure, thus increasing the computational effort. These reasons induce the scientific community to continue to search for effective optimization algorithms independently from the research going on in surrogate optimization.

In this paper two hybridized PSO algorithms are evaluated in terms of the computational effort required to solve real world engineering problems. The comparative study is made in terms of two stopping criteria based on a fixed iteration number and a fixed threshold to evaluate the quality of solutions and the convergence to near-optimal solutions, respectively.

The paper is organized as follows: Section 2 presents the basic concepts of the particle swarm optimization metaheuristic and the two PSO algorithms used in this study. Section 3 shows the performance comparison for both algorithms and the conclusions obtained from the study. General conclusions and future work are stated in Section 4.

2. The Approaches

The approaches used in this paper are described below. The first, named "simple constrained particle swarm optimization" (SiCPSO, for short, developed by the second author), is based on the classical PSO model [19], but it incorporates a mechanism to handle constraints and a different equation for updating the positions of particles. The second one, named "minimized computational effort particle swarm optimization" (MCEPSO, for short, developed by the first author), is also derived from the classical PSO model, but it includes a number of methodologies aimed to reduce the number of objective function calculations which often take most of the CPU time in real world optimization problems.

2.1. Classical PSO Model. The classic PSO algorithm operates on a population of individuals, named *particles*. Such particles consist of vectors of real numbers, and each vector position is named *dimension*. The algorithm iterates searching for solutions and saves the best position found so far for the particles ("global best", gbest model). The best value reached by each particle ("personal best", pbest) is also stored. The particles evolve using two update formulas, one for the velocity of particles and another for its position, in the following way:

$$ {}^{k}v_d^i = {}^{k-1}v_d^i + c_1 r_1 \left({}^{k}x_d^{\text{pbest}} - {}^{k}x_d^i \right) + c_2 r_2 \left({}^{k}x_d^{\text{gbest}} - {}^{k}x_d^i \right), \tag{1} $$

$$ {}^{k}x_d^i = {}^{k-1}x_d^i + {}^{k}v_d^i, \tag{2} $$

where k is the current iteration at the moment of updating, ${}^{k}v_d^i$ is the velocity of the particle i at the dimension d, c_1 is the personal learning factor, and c_2 is the social learning factor. r_1 and r_2 are two random numbers within the range $[0, 1]$, which are used to introduce stochastic values to determine how much of each factor is added. ${}^{k}x_d^{\text{pbest}}$ is the dimension d of the best position reached by the particle i at iteration k and ${}^{k}x_d^{\text{gbest}}$ is the best position reached by any particle in the entire swarm at iteration k. ${}^{k}x_d^i$ is the value of the particle i at the dimension d at iteration k.

Some modifications to (1) have been proposed with the goal to alleviate negative effects related to the parameters of such equation. The concept of the inertia weight w [7] was proposed to reduce strong attractions to the best positions previously reached, information that is included in the previous velocity. The following equation shows this modification:

$$ {}^{k}v_d^i = w^{k-1}v_d^i + c_1 r_1 \left({}^{k}x_d^{\text{pbest}} - {}^{k}x_d^i \right) + c_2 r_2 \left({}^{k}x_d^{\text{gbest}} - {}^{k}x_d^i \right), \tag{3} $$

where w is usually a value within the range $[0, 1]$ and it is preferably decreased over the time.

Another modification to (1) considers a constriction factor $\mathcal{X}$ [20] whose goal is to balance global exploration and local exploitation of the swarm. The following equation shows this modification:

$$ {}^{k}v_d^i = \mathcal{X} \left[{}^{k-1}v_d^i + c_1 r_1 \left({}^{k}x_d^{\text{pbest}} - {}^{k}x_d^i \right) + c_2 r_2 \left({}^{k}x_d^{\text{gbest}} - {}^{k}x_d^i \right) \right], \tag{4} $$

where $\mathcal{X}$ has a default value [20] of 0.729, but it could be set to a different value.

Many other variants have been proposed in order to increase the exploration step and obtain a fast convergence. For example, in [21] the authors used a decreasing proportional coefficient and a random exploration velocity, in [22] the velocity equation was modified with the addition of a term simulating the center of the mass used in the Big Bang-Big Crunch algorithm, and in [23] the authors presented a PSO with a learning strategy based on the simulation of the human social communication behavior. For the same purpose, SiCPSO and MCEPSO are proposed and described below.

2.2. The SiCPSO Approach. The modifications introduced in the SiCPSO approach with respect to the classical PSO model are described as follows.

2.2.1. Updating Particles. Previous works [24, 25] presented a combined equation to updating the positions of particles. Those used the update equation presented by Kennedy and Eberhart [26]. Here, a different version of that formula is employed (shown in (5)), as follows. Instead of using the common equation (2) (which uses (4) to update the velocity) in all the iterations, it is selected with a probability of 0.925. The rest of the time a Gaussian formula depicted in (5) is used. In this case, the position of each particle is randomly chosen from a Gaussian distribution with the mean selected as the average between the best position recorded for the

particle and the best in the swarm. The standard deviation is the difference between these two values:

$$ {}^{k}x^{i} = N\left(\frac{{}^{k}x^{\text{pbest}} + {}^{k}x^{\text{gbest}}}{2}, \left|{}^{k}x^{\text{pbest}} - {}^{k}x^{\text{gbest}}\right|\right), \quad (5) $$

Where ${}^{k}x_{d}^{i}$ is ith particle to be updated at iteration k, N is the Gaussian random generator, and ${}^{k}x^{\text{pbest}}$ and ${}^{k}x^{\text{gbest}}$ are, respectively, the best position reached by the ith particle at iteration k and the best position reached by any particle in the swarm. The probability of selection adopted (0.925), was selected after performing a Latin hypercube design [27] (LHD) study.

2.2.2. Handling Constraints. The constraint-handling method used in the proposed approach is one of the simplest. It is based on the following rule: "a feasible particle is preferred over an infeasible one." This constraint-handling scheme is used when the pbest and gbest particles must be chosen at every iteration of the algorithm. It is carried out after updating each particle in the swarm and just before selecting new values for pbest and gbest particles. The method works as is described below.

(i) If the particle is feasible but its corresponding pbest was infeasible, then the pbest is updated with the new value of the particle.

(ii) If the particle is infeasible but its pbest is feasible, then no change is made.

(iii) If both particle and pbest are infeasible, then the one closer to the feasible region is chosen. In order to do that, the algorithm stores the largest violation obtained for each constraint in each iteration. When an individual is found to be infeasible, the sum of its constraints violations (this value is normalized with respect to the largest violation stored so far) is the one considered as its distance to the feasible region.

The same process is applied to select the gbest particle at each iteration of the algorithm.

2.2.3. Keeping Mechanism. The keeping mechanism is applied to control that all the dimensions in all particles are within the allowable bounds. Those bounds are determined by the range (upper and lower limits) of each design variable corresponding to the problem that the algorithm is solving. If some particle dimension after the updating process exceeds the upper limit, that dimension will be reinitialized to the lower limit corresponding to the design variable that it represents. Using the lower limit the possibility that in the next iterations the same dimension exceeds again the upper limit is reduced.

2.2.4. SiCPSO Pseudocode. Pseudocode 1 shows the pseudocode of SiCPSO algorithm. At the beginning of the process, the vectors of position and velocity of each particle are initialized (lines 2 and 3). After evaluating the particles and obtaining the best values pbest and gbest (lines 4 and 5), the swarm begins to evolve. During the evolutionary process, new values of pbest and gbest are chosen and both the velocity and the position of each particle are updated (lines 6 to 21) until the stop condition is reached. At line 18, the keeping mechanism is applied. After that, the particles are evaluated and new "best" values are recorded (lines 19 and 20). Finally, the best value reached by the swarm is returned (lines 22 and 23).

```
(0)  SiCPSO:
(1)   Swarm Initialization
(2)     initialize positions
(3)     initialize velocities
(4)     Evaluate fitness of each particle
(5)     Record pbest for each particle and gbest
(6)   Swarm flights through the search space
(7)     DO
(8)       FOR i = 1 TO numberOfparticles DO
(9)         FOR j = 1 TO numberOfdimensions DO
(10)          Update velocities with (4)
(11)          IF flip(0.925)
(12)            Update particles with (2)
(13)          ELSE
(14)            Gaussian update with (5)
(15)          END
(16)        END
(17)      END
(18)      Keeping particles
(19)      Evaluate fitness of each particle
(20)      Record pbest and gbest
(21)    WHILE(not stop condition)
(22)    result=bestOfTheSwarm()
(23)    RETURN(result)
```

Pseudocode 1: Pseudocode of SiCPSO.

2.3. The MCEPSO Approach. MCEPSO is based on the classical PSO approach using (3), which uses the concept of inertia weight to update the velocity of the particles. In MCEPSO, constraints are classified into "side constraints" and "physical constraints". Side constraints (often referred to as "geometrical constraints") are limitations directly expressed in terms of upper and lower bounds for the design variables. On the other hand, physical constraints are restrictions expressed on quantities that are more complex functions of the design variables. Sometimes, physical constraints evaluation may only be achieved by running some external and time-consuming programs. Often, however, physical constraints are complex analytical equations limiting the design space but not requiring any external code to run. In this case, it is possible to order the different steps required by the optimization process as follows: firstly, side constraints are quickly calculated because they are a simple comparison between each particles position and the bounds; secondly, physical constraints are evaluated because they are functions of each particles position, but they do not require time-consuming programs for evaluation; and, finally, the objective function is evaluated because this very frequently requires a significant computational effort.

In the development of MCEPSO, the assumption that constraints are evaluated more quickly than the objective has been done, as it often happens. In case both constraints and

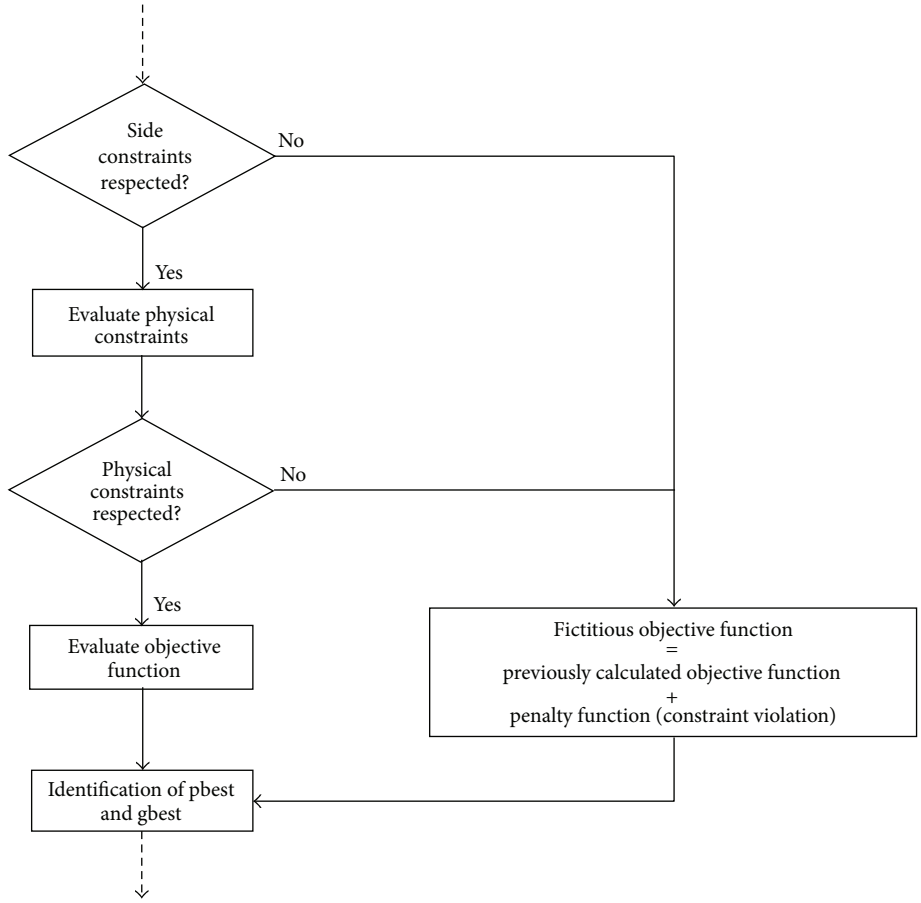

Figure 1: MCEPSO flowchart.

objectives are to be evaluated in one simulation, simulation is run one time only, saving constraints and objectives to be loaded in different moments of the algorithm execution.

On the basis of these general considerations, MCEPSO has been structured in order to reduce, as much as possible, the computational effort of optimizations. As described in Figure 1, at each iteration and for each particle of the swarm, MCEPSO firstly evaluates if the particle is within side constraints. In case it is not, MCEPSO avoids calculating physical constraints and the objective function. Instead, it calculates a fictitious objective function which is the previously calculated objective function for the last feasible position occupied by the particle plus a penalty function proportional to the distance of the nonfeasible present particle's position from bounds. On the contrary, if the particle is within side constraints, the evaluation of physical constraints is performed. If physical constraints are infringed, MCEPSO calculates a fictitious objective function with a penalty function proportional, this time, to physical constraints infringement. Otherwise, just in case both side constraints and physical constraints are respected, the intensive task of evaluating the objective function is performed. After this, pbest and gbest are updated and the objective function values for feasible particles are stored to be used, in the following iterations, as fictitious objectives in case that the same particle becomes infeasible. If no values have yet been stored to be used for infeasible particles to calculate the fictitious objective, the constraint infringement term is added to a reference value for the objective function (an "order of magnitude" of the expected fitness) that the user should supply. A number of previous numerical tests have shown an increase of the algorithm efficiency if the penalty function weight is gradually increased when the same particle continues to be infeasible iteration after iteration.

It is important to highlight that, in a large number of practical engineering optimizations, the mentioned approach avoids impossible rather than intensive calculations. In fact, very often particles positions represent "parameters" of a numerical model. In structural optimizations, for example, some particular combinations of the design variables, even if respecting side constraints, could lead to nonphysical situations for which a finite element program crashes, interrupting then the optimization process. In structural design and in many other disciplines where optimization is implemented, physical constraints infringement usually cannot be tolerated even if the optimization process however requires a fitness value for each particle to continue. So, an alternative mechanism to supply the optimizer with fitness values for infeasible particles is needed. And the proposed penalty approach is thought to force the variables to reenter the design domain as soon as possible.

2.3.1. Handling Constraints. EAs have no implicit mechanism to deal with constraints. A large number of different methodologies have been proposed, but most of them are based on penalty functions. Penalty functions are added to the objective function in order to penalize infeasible design points with respect to feasible ones. An efficient way to express a penalty function is to calculate it in terms of distance from the feasible space. The penalty function is then added to the objective function to obtain a fictitious objective value. However, in a large number of practical optimization problems, troubles are encountered in using this approach because constraints are often required to be strictly satisfied. The classical application is structural optimization in which, for example, a negative value for a design variable representing an element thickness is meaningless and could cause the external structural program to crash. An obvious alternative to solve this problem is a penalty term $P({}^{\text{pres}}X)$ for the present candidate ${}^{\text{pres}}X$ to be added to the last feasible fitness value $f({}^{\text{last feas}}X)$ identified before the constraint violation, as follows:

$$f\left({}^{\text{pres}}X\right) = \begin{cases} f\left({}^{\text{pres}}X\right), & \text{if } {}^{\text{pres}}X \text{ feasible,} \\ f\left({}^{\text{last feas}}X\right) + r * P\left({}^{\text{pres}}X\right), & \text{if } {}^{\text{pres}}X \text{ infeasible,} \end{cases} \tag{6}$$

Where ${}^{\text{pres}}X$ is the particle position at the present iteration, $f({}^{\text{last feas}}X)$ is the previous design vector respecting all constraints, and r is a multiplying factor set to amplify constraint violation in the penalty evaluation. The penalty function $P({}^{\text{pres}}X)$ has the following expression equation:

$$P\left({}^{\text{pres}}X\right) = \begin{cases} \displaystyle\sum_{i=1}^{N}\left[\max\left(0, x_i^L - {}^{\text{pres}}x_i\right)^2 + \min\left(0, x_i^U - {}^{\text{pres}}x_i\right)^2\right], \\ \qquad\qquad \text{if } {}^{\text{pres}}X \notin \left[X^L, X^U\right], \\ \displaystyle\sum_{j=1}^{P}\min\left(0, g_j({}^{\text{pres}}X)\right)^2, \\ \quad \text{if } {}^{\text{pres}}X \in \left[X^L, X^U\right], \end{cases} \tag{7}$$

where ${}^{k}X^{j} = [{}^{k}x_1{}^{j}, {}^{k}x_2{}^{j}, \ldots, {}^{k}x_N{}^{j}]$ is the jth particle position at the kth iteration and X^L and X^U are lower and upper bounds, respectively. In this way, unnecessary (and sometimes meaningless) fitness calculations are avoided and an efficient mechanism to force particles to re-enter the design space is implemented.

3. Performance Comparison

A common choice to understand pros and cons of optimization algorithms is to test them using standard benchmark problems. The problems themselves and the respective numerical settings are selected to be the same used in a large number of previous studies, in order to concentrate the attention to the optimization algorithm itself. In this paper, four of the most widely used benchmark problems were selected. The authors established common numerical settings to be able to compare the results obtained by each one of the two proposed algorithms. Welded beam design (E01), pressure vessel design (E02), speed reducer design (E03), and tension/compression spring design (E04) are well-known test problems used in the evaluation of algorithm performances [10–12, 14, 22]. Mathematical formulations of the four problems used in this study are reported in the appendix. All the mentioned benchmark problems are characterized by nonlinear objective functions and/or nonlinear constraints. Particularly, the benchmark functions E01–E04 used in the present study are characterized by a dimensionality varying from 4 to 7. Optimization algorithms show a different behavior as the search space dimension increases and more efficient strategies are necessary in order to increase the exploration. Anyway, although the difficulty of an optimization problem generally increases with dimensionality, many real-world problems can be solved by decomposing them into a number of smaller subproblems involving a limited number of decision variables while considering the rest as constants [28]. This is why many engineering problems, as in the aerospace sector in which the presented algorithms were developed, are influenced by groups of parameters with limited interactions among the groups themselves. And this is reflected by the selected benchmark problems E01–E04 that are suitable to be applied as test functions to represent real-world optimization problems.

Two different kinds of tests were carried out in order to compare the algorithms' behaviour, using the same parameters for each test.

SiCPSO:

(i) learning factors: $c_1 = c_2 = 1.8$,

(ii) constriction factor: $\mathcal{X} = 0.8$ (velocity update based on (4)),

(iii) probability of Gaussian equation: 0.075.

MCEPSO:

(i) learning factors: $c_1 = 2.0$; c_2 linearly increased at each iteration, from 1.0 to 2.0,

(ii) inertia weight: w linearly decreased at each iteration, from 0.9 to 0.4 (velocity update based on (3)).

For both, SiCPSO and MCEPSO, the mentioned numerical settings were established after several empirical tests.

3.1. Fixed Iterations Tests. This test aims to evaluate the quality of the solutions obtained by the optimizers in terms of different values: best, mean, worst and standard deviation over 50 independent runs (executions) for each problem. The stopping condition for each run is based on the number of iterations performed. In other words, for each problem the optimizer run for a fixed number of iterations and the objective function final value are obtained at the end of each single run. Then, for statistics purposes, comparing the values obtained at the final iteration for each of the 50 runs, the best, the worst, the mean, and standard deviation are calculated.

In order to understand the influence of the swarm size on the performance of each algorithm, the tests were carried out keeping constant the total number of function evaluations executed. For real optimization problem, 30000 is a considerable number of function evaluations and for this reason was adopted in this experimental study.

The results obtained for fixed iterations tests (FIT) are summarized in Tables 1 and 2. Table 1 shows the best values obtained for each algorithm after 3000 iterations, considering a population of 10 particles. Table 2 shows the best values obtained for each algorithm after 1500 iterations, considering a population of 20 particles.

From the observations of Tables 1 and 2, it is possible to note that both algorithms find a "best" solution, over the 50 performed runs, which is very close, or event coincident, to the optimal values for the benchmark problems, both for 10 and 20 particles. The behaviour of the two algorithms is different for the "mean" values. MCEPSO performs better for E01 both for 10 and 20 particles, but SiCPSO gives better results than MCEPSO for E02. For E03 and E04, both the algorithms show, more or less, the same results. About the standard deviation, MCEPSO performs better for E01 for both 10 and 20 particles. But SiCPSO works better than MCEPSO for E02. For E03 and E04, the behaviour of the two algorithms is quite similar both for 10 and 20 particles. About the "worst" values, MCEPSO performs better than SiCPSO for E01. On the contrary, SiCPSO gets lower values than MCEPSO for E02. For E03 and E04, performances are quite similar. In general, it is possible to conclude that MCEPSO obtains better results for E01 while SiCPSO works better for E02. Results obtained for E03 and E04 are very close for the two algorithms.

It is important to highlight that the initialization techniques used for the algorithms represent an important issue affecting the values obtained in the very first iterations of each run. This aspect influences the behaviour of each algorithm in the first iterations, promoting large differences between MCEPSO and SiCPSO.

For MCEPSO, the swarm is randomly initialized within side constraints, but there is no guarantee that the initial swarm respects physical constraints. Since the constraint-handling technique adopted for MCEPSO is basically a penalty function approach, in the very first iterations the objective value is probably inflated due to physical constraints infringement (see (6) and (7) and Figure 1). That is, at the beginning of the search process, MCEPSO has a fitness enlarged by the penalty term because (probably) the solutions are not feasible.

The initialization technique used for SiCPSO is similar to MCEPSO's; that is, the swarm is randomly initialized within side constraints. The difference in the very first iterations could be caused, then, by the constraint-handling technique adopted for SiCPSO, because if at least one solution is feasible (or close to a feasible one), the whole swarm is guided quickly to a feasible zone. In that manner, SiCPSO is able to obtain feasible solutions at the first search stages.

Tables 3, 4, 5, and 6 show a summary of results of each engineering problem obtained with the algorithms more representative of the state of the art in optimization. In the tables, the first column shows the name of the algorithm and the second one the number of fitness evaluations (FEs) used to obtain the best results. The third column shows the number of different executions (runs) performed by each algorithm. This value is used to calculate the statistical values: mean and standard deviation which are shown in columns five (mean) and six (St. Dev.). The best value and the worst value obtained by each algorithm considering all the executions are shown in columns four (best) and seven (worst).

Observing the tables it is possible to conclude that both SiCPSO and MCEPSO achieve the known best values with mean and standard deviation values comparable to the state-of-the-art algorithms for the same engineering problems.

3.2. Fixed Threshold Tests. The second kind of tests carried out within the present work has been named "fixed threshold tests" (FTT). As mentioned before, in these tests, the attention is focused on the *convergence speed* which means the quickness, expressed in terms of number of function evaluations, needed to reduce the objective value below a fixed threshold. Computing time is considered here less significant than the function evaluations number because it is machine dependent. These tests have fundamental importance in practical optimization. The possibility of reaching a quasi-optimal solution in an affordable number of function evaluations is crucial when dealing with time-consuming problems. It is sufficient to highlight to the reader the huge number of design problems in which optimization cannot be faced because performance evaluations are obtained by solving complex physical problems, needing long calculations to get a solution for each design parameters set. Many studies are continuously done in the field of numerical solution techniques in order to reduce the computational costs; besides that, a challenge for evolutionary algorithms is needed to supply the designer with optimization algorithms requiring only the number of evaluations strictly needed to get an acceptable approximation to the optimal solution. In order to measure the convergence speed of an algorithm, an intuitive choice could be to fix a quasi-optimal solution and then counting the number of evaluations needed for reaching a value respecting that threshold. Repeating such a test for a significant number of times gives significant statistics to evaluate that previously defined as *convergence speed* of the algorithm. In this study, the threshold was arbitrarily fixed 20% higher than the best known value for each of the four benchmark problems. Of course, the choice of a quasi-optimal solution which is 20% far from the best known optimum has no particular numerical meaning. It simply represents a reasonable approximation of the solution that is used, within the present study, to compare the algorithms behaviour.

It is interesting to note than both algorithms work in different ways due the mechanisms they have to evaluate the particles; that is, SiCPSO evaluates fitness and constraints at the same time for all particles, in each iteration. Instead, MCEPSO, in an attempt to reduce the number of unnecessary calculations, evaluates the fitness only if all constraints are satisfied; otherwise, it avoids the calculation for the current particle, replacing its true uncalculated fitness with a fictitious one. This is the reason why for the MCEPSO algorithm the number of fitness evaluations are reported separately with

Table 1: FIT: results with 10 particles: 3000 iterations. Statistics over 50 runs.

Pr.	Best MCEPSO	Best SiCPSO	Mean MCEPSO	Mean SiCPSO	St. Dv. MCEPSO	St. Dv. SiCPSO	Worst MCEPSO	Worst SiCPSO
E01	1.724852	1.724852	1.7343	2.1871	0.0243	0.4762	1.8811	3.4460
E02	6059.714335	6059.714335	6286.2411	6194.1437	265.6660	143.5273	6820.4147	6424.0662
E03	2996.348166	2996.348165	2996.3495	2996.3481	0.0027	0.0000	2996.3560	2996.3481
E04	0.012665	0.012665	0.0134	0.0136	0.0007	0.0010	0.0156	0.0170

Table 2: FIT: results with 20 particles: 1500 iterations. Statistics over 50 runs.

Pr.	Best MCEPSO	Best SiCPSO	Mean MCEPSO	Mean SiCPSO	St. Dv. MCEPSO	St. Dv. SiCPSO	Worst MCEPSO	Worst SiCPSO
E01	1.724852	1.724852	1.7333	1.9590	0.0178	0.2557	1.8043	2.6175
E02	6059.714335	6059.714335	6274.2285	6172.3441	202.7861	140.6302	6820.4101	6410.0867
E03	2996.348165	2996.348165	2996.3482	2996.3481	0.0000	0.0000	2996.3482	2996.3481
E04	0.012666	0.012665	0.0132	0.0133	0.0005	0.0005	0.0146	0.0146

respect to the number of constraints evaluations, while for the SiCPSO only one value corresponding to fitness and constraints evaluations is reported in Tables 8 and 9 for each problem. This performance study is described in two ways: SiCPSO (FCE: fitness and constraints) evaluations against MCEPSO fitness evaluations (FEs) and SiCPSO (FCE: fitness and constraints) evaluations against MCEPSO constraints evaluations (CEs).

As it was explained before for FIT study, FTT have also been performed using 10 and 20 particles in order to understand if there is any influence of the swarm size on the convergence speed. About the experiments of the algorithms considering 10 particles, each run stops after executing 3000 iterations or after reaching the stop condition $f(\mathbf{x}) < B$ with $\mathbf{x}$ a feasible solution and B a threshold determined by the values in Table 7. The statistics are calculated over 50 complete runs, with reinitializations of runs if the 3000 iterations were reached but not the stop condition.

Table 8 shows the values obtained for Problem E01 with 10 particles. SiCPSO obtained a minimum best FCE although the best value corresponds to MCEPSO which is composed by a low number of fitness evaluations combined with a higher amount of constraints evaluations. In fact, SiCPSO FCE is lower than MCEPSO CE (that shows the difficulty of E01 for MCEPSO) compared with the FE which is quite lower. The variability of mean values between MCEPSO and SiCPSO over the 50 runs are similar to the bests; that is, SiCPSO FCE is higher than FE but lower than CE (the last ones of MCEPSO). The standard deviation value of SiCPSO is slightly higher compared with those of MCEPSO; the same occurs with the worst values. That could indicate that SiCPSO presents more variability; that is, many iterations are needed for finding a solution that satisfies the stop condition. It is important to note that SiCPSO stalled in 20 runs (over the 50) so 20 reinitializations were necessary to obtain a solution. MCEPSO did not have to repeat any run, possibly because this problem is more difficult for SiCPSO than for MCEPSO.

The results obtained for E02 with 10 particles (Table 8) indicate that SiCPSO quickly reached a good solution obtaining the minimum FCE. MCEPSO needed some more FE to obtain the best results (in average) and many CEs. That fact is observed also in the low mean, standard deviation and worst FCE values obtained by SiCPSO compared with those (higher) FE values of MCEPSO. It is important to note that the values corresponding to the constraints evaluations are higher than those corresponding to the fitness evaluations of MCEPSO, specially the one corresponding to the worst values. The last could indicate that for MCEPSO the evaluations of E02's constraints are the most difficult part (compared with the evaluation of the function, FE).

Problem E03 with 10 particles (Table 8) seems to be easy to solve for both algorithms because they could reach the solution with few FCE. Nevertheless SiCPSO used lower FCE compared with the CE of MCEPSO although that obtained the solution with only one evaluation of the objective function and found that value in the first particle evaluated. SiCPSO obtained the solution in the first iteration so the minimum number of FCE is 10 (because all particles were evaluated).

The minimum FE for E04 with 10 particles (Table 8) was obtained by MCEPSO (FE and CE) although SiCPSO obtained lower FCE mean values compared with the CE mean value of MCEPSO. The higher standard deviation was obtained by SiCPSO. The worst value of FCE (over the 50 runs) was obtained by SiCPSO which states that the algorithm needed many evaluations in some runs to reach a good solution compared with the lower values of MCEPSO (FE and CE). Both algorithms had to repeat a stalled run.

Table 9 shows the results obtained by both algorithms using 20 particles with 1500 iterations. For Problem E01, SiCPSO found a solution with a lower number of FCE compared with the higher value of CE of MCEPSO, although the last algorithm needed a lower number of fitness evaluations to obtain a solution (FE). The FCE mean of SiCPSO is a higher value when it is compared with the MCEPSO FE but not too much different to that of MCEPSO CE value. SiCPSO obtained a higher standard deviation which states that the variability of FCE is bigger than that of MCEPSO.

Table 3: Results for E01: welded beam design.

Algorithm	FEs	Runs	Best	Mean	St. Dev.	Worst
$(\mu + \lambda)$-ES[1]	30000	30	1.724852	1.777692	$8.8E-2$	2.074562
ϵPSO[2]	5000	NA	1.7258	1.8073	$1.2E-1$	2.1427
HPSACO[3]	NA	30	1.724849	1.727564	$8.3E-3$	1.759522
NM-PSO[4]	80000	30	1.724717	1.726373	$3.5E-3$	1.733393
HEA-ACT[5]	200000	30	2.380957	2.380971	$1.3E-5$	2.381021
PSOLVER[6]	297	30	1.724717	1.724717	$1.6E-11$	1.724717
CPSOSA[7]	2450	30	1.724852	1.724853	$1.7E-5$	1.724861
GABC[8]	30000	30	1.724	1.763	$3.3E-2$	NA
PSO[9]	30000	30	1.724897	1.730813	$1.0281E-2$	1.767000
GDA[10]	20000	20	1.724852	1.724852	0	1.724852
SiCPSO	30000	50	1.724852	1.9590	$2.557E-1$	2.6175
MCEPSO	30000	50	1.724852	1.7333	$1.78E-2$	1.8043

NA stands for not available.
[1][8], [2][9], [3][10], [4][11], [5][12], [6][13], [7][14], [8][15], [9][16], and [10][17].

Table 4: Results for E02: pressure vessel design.

Algorithm	FEs	Runs	Best	Mean	St. Dev.	Worst
$(\mu + \lambda)$-ES[1]	30000	30	6059.701610	6379.938037	$2.1E+2$	6820.397461
ϵPSO[2]	50000	NA	6059.7143	6136.7744	$1.123306E+2$	6410.0868
HPSACO[3]	NA	NA	NA	NA	NA	NA
NM-PSO[4]	NA	NA	NA	NA	NA	NA
HEA-ACT[5]	200000	30	NA	NA	NA	NA
PSOLVER[6]	310	30	6059.7143	6059.7143	$4.63E-12$	6059.7143
CPSOSA[7]	2450	30	6059.7143	6059.7143	$2.26E-06$	6059.7146
GABC[8]	30000	30	6059.714	6218.515	$1.9E+02$	NA
PSO[9]	NA	NA	NA	NA	NA	NA
GDA[10]	20000	20	6059.83905683	6149.72760669	$2.1077E+2$	6823.60245024
SiCPSO	30000	50	6059.714335	6172.3441	$1.406302E+2$	6410.0867
MCEPSO	30000	50	6059.714335	6274.2285	$2.027861E+2$	6820.4101

NA stands for not available.
[1][8], [2][9], [3][10], [4][11], [5][12], [6][13], [7][14], [8][15], [9][16], and [10][17].

That variability is observed in the high number of SiCPSO FCE needed to reach a solution in some runs SiCPSO had to reinitialize 9 runs while MCEPSO only 2 runs. Again, for Problem E01, SiCPSO presents more difficulties than MCEPSO in the process to find a solution respecting the established stop condition established.

For the Problem E02 with 20 particles, SiCPSO obtained the minimum FCE indicating that it quickly reaches a solution (see Table 9). MCEPSO needed just some more FEs and many CEs to obtain the solution. That fact is observed also in the low mean, standard deviation and worst FCE values obtained by SiCPSO compared with those (higher) of MCEPSO. Note that the values corresponding to the MCEPSO constraints evaluations are higher than those corresponding to the fitness evaluations, specially those of the worst values. The last affirmation could indicate that for MCEPSO the evaluations of constraints of E02 are the most difficult part to optimize. Also MCEPSO had to re-initialize 3 runs while SiCPSO did not have any.

Table 9 shows the results for Problem E03 with 20 particles. Both algorithms had similar behaviour for that of the 10-particles case. That is, E03 seems to be easy to solve for MCEPSO and SiCPSO. They could reach the solution with few FEs. Nevertheless SiCPSO used lower FCE compared with those CEs of MCEPSO. In fact, note that MCEPSO obtained the solution with only one evaluation of the objective function (FE) and found that value in the first particle evaluated. Also it is important to observe that 50 evaluations of constraints were needed to find the best result. SiCPSO obtained the solution in the first iteration so the minimum number of FCE is 20; that is, all particles were evaluated.

For Problem E04 with 20 particles, Table 9 shows that the best values of FE and CE were obtained by MCEPSO compared with the FCE needed by SiCPSO. The worst value of FCE (over the 50 runs) was obtained by SiCPSO which states that the algorithm needed many evaluations in some run to reach a good solution (compared with the lower values of MCEPSO) although no run had to be repeated. MCEPSO had to re-initialize 5 runs.

Concluding this experimental study it is interesting to observe that SiCPSO obtained better FCE mean values for all problems with 10 and 20 particles (except for E01 and

Table 5: Results for E03: speed reducer design.

Algorithm	FEs	Runs	Best	Mean	St. Dev.	Worst
$(\mu + \lambda)$-ES[1]	30000	30	2996.348094	2996.348094	0	2996.348094
ϵPSO[2]	NA	NA	NA	NA	NA	NA
HPSACO[3]	NA	NA	NA	NA	NA	NA
NM-PSO[4]	NA	NA	NA	NA	NA	NA
HEA-ACT[5]	200000	30	2994.4991	2994.6134	$7.0E-2$	2994.7523
PSOLVER[6]	NA	NA	NA	NA	NA	NA
CPSOSA[7]	NA	NA	NA	NA	NA	NA
GABC[8]	30000	30	2996.783	2996.783	0	2996.783
PSO[9]	5000	30	3000.8	NA	NA	NA
GDA[10]	20000	20	2996.348072	3094.556809	$2.448E-1$	3016.492651
SiCPSO	30000	50	2996.348165	2996.3481	0	2996.3481
MCEPSO	30000	50	2996.348165	2996.3482	0	2996.3482

NA stands for not available.
[1][8], [2][9], [3][10], [4][11], [5][12], [6][13], [7][14], [8][15], [9][16], and [10][17].

Table 6: Results for E04: tension/compression spring design.

Algorithm	FEs	Runs	Best	Mean	St. Dev.	Worst
$(\mu + \lambda)$-ES[1]	30000	30	0.012689	0.013165	$3.9E-4$	0.014078
ϵPSO[2]	NA	NA	NA	NA	NA	NA
HPSACO[3]	NA	NA	NA	NA	NA	NA
NM-PSO[4]	80000	30	0.00126302	0.0126314	$1.5824E-05$	0.012719
HEA-ACT[5]	200000	30	0.012665233	0.012665234	$1.49E-09$	0.01266524
PSOLVER[6]	253	30	0.0126652	0.0126652	$2.46E-9$	0.0126652
CPSOSA[7]	NA	NA	NA	NA	NA	NA
GABC[8]	30000	30	0.126	0.127	$2.8E-4$	NA
PSO[9]	30000	30	1.724897	1.730813	$1.0281E-2$	1.767000
GDA[10]	20000	20	0.0126652296	0.0140793687	$2.966889E-4$	0.0128750789
SiCPSO	30000	50	0.012665	0.0133	$5.0E-4$	0.0146
MCEPSO	30000	50	0.012666	0.0132	$5.0E-4$	0.0146

NA stands for not available.
[1][8], [2][9], [3][10], [4][11], [5][12], [6][13], [7][14], [8][15], [9][16], and [10][17].

Table 7: Stop condition used in FTT.

Problem	Best known $f(\mathbf{x}^*)$	Stop condition $f(\mathbf{x}) \leq f(\mathbf{x}^*) + 20\%$
E01	1.724852	$f(\mathbf{x}) \leq 2.0698$
E02	6059.714335	$f(\mathbf{x}) \leq 7,271.7$
E03	2996.348165	$f(\mathbf{x}) \leq 3595.6$
E04	0.012665	$f(\mathbf{x}) \leq 0.0152$

E04 with 20 particles) if these values are compared with MCEPSO CE. On the other hand, MCEPSO obtained better mean FE values for E01, E03, and E04 with 10 and 20 particles if these values are compared with FCE values of SiCPSO. These results show a fair bahavior of both algorithms for the engineering problems studied.

4. Conclusions and Future Work

In this paper, a performance study is presented using two different PSO-based approaches to solve engineering optimization problems. The comparison between the approaches is mainly based on the behaviour observed in finding acceptable solutions in a reasonable amount of iterations rather than achieving the best possible optimum in an unlimited number of calculations. The possibility of determining an approximation of the optimal solution in an affordable calculation time is, in fact, crucial in many disciplines in which optimization is used.

Both algorithms are based on the classical PSO approach but implementing different methodologies to improve its performance. In SiCPSO, the position of each particle is determined with values obtained from Gaussian and uniform distributions in order to improve the search space exploration capability. SiCPSO implements a mechanism to handle constraints based on the rule that "a feasible particle is preferred to an unfeasible one." In MCEPSO, fitness calculation is performed if and only if all constraints are satisfied; otherwise a fictitious fitness is determined and used. MCEPSO also implements a penalty function approach to deal with constraints. The penalty function is thought to deal with

Table 8: FTT: number of evaluations obtained with 10 particles: 3000 iterations. Statistics over 50 complete runs.

Problem	Mean MCEPSO	Mean SiCPSO	St. Dv. MCEPSO	St. Dv. SiCPSO	Best MCEPSO	Best SiCPSO	Worst MCEPSO	Worst SiCPSO	Rep. Runs MCEPSO	Rep. Runs SiCPSO
FE E01	300.7	1061.4	177.9	1070.2	48	130	855	5620	0	20
CE E01	1376.1		814.3		221		3753			
FE E02	572.9	193.8	1151.9	62.4	99	30	8337	370	0	0
CE E02	1769.2		3082.1		284		22103			
FE E03	13.4	98.4	16.8	50.9	1	10	77	210	0	0
CE E03	537.9		557.2		27		2716			
FE E04	166.5	1223.3	247.4	3514.7	11	190	1383	22480	1	1
CE E04	1488.2		1495.5		150		7251			

Table 9: FTT: number of evaluations obtained with 20 particles: 1500 iterations. Statistics over 50 complete runs.

Problem	Mean MCEPSO	Mean SiCPSO	St. Dv. MCEPSO	St. Dv. SiCPSO	Best MCEPSO	Best SiCPSO	Worst MCEPSO	Worst SiCPSO	Rep. Runs MCEPSO	Rep. Runs SiCPSO
FE E01	413.9	2292.8	266.3	3904.6	33	140	1340	18320	2	9
CE E01	1804.7		1153.5		176		5294			
FE E02	453.8	255.2	166	73.6	83	80	901	400	3	0
CE E02	1421.3		662.1		226		3603			
FE E03	13.8	134.4	11.1	71.68	1	20	44	760	0	0
CE E03	287		248.7		50		1730			
FE E04	132	3231.6	125.9	5968.6	2	240	746	27720	5	0
CE E04	1147.5		846		40		5083			

a large number of engineering problems where constraint infringement is not tolerable because it represents nonphysical situations.

On the basis of the experiments performed, it can be concluded that SiCPSO takes less iterations (in FIT) to initially decrease the fitness function because MCEPSO is more influenced by the penalty function when dealing with nonfeasible particles in the first iterations. In the same tests, SiCPSO shows higher values of the standard deviation with respect to MCEPSO. When the algorithms are compared in terms of number of calculations to converge below a fixed threshold (in FTT), it is difficult to come to a general conclusion, because MCEPSO incorporates a mechanism to avoid unneeded fitness calculations. Generally speaking, MCEPSO achieves acceptable approximations of the optimum in less FEs but it takes more CEs than SiCPSO. Then, from this experimental study it is possible to state that both algorithms represent good alternatives to solve engineering optimization problems.

Further experiments with other real optimization problems could be interesting to test the algorithms in order to determine if the performance of both of them is still acceptable and comparable. Also, for future research it could be considered to try to merge the positive aspects of the two algorithms compared here. An attractive idea could be to implement in a single algorithm the distinction between fitness and constraint calculations avoiding sometimes impossible evaluations in practical engineering problems, with the constraint-handling technique used in SiCPSO in order to avoid the negative drawbacks of the penalty function approach.

Appendix

Engineering design problems used to test the algorithm proposed.

E01: Welded Beam Design Optimization Problem. The problem aims to design a welded beam with minimum cost, subject to some constraints [29]. Welded beam structure consists of a beam A and the weld required to hold it to member B. The objective is to find the minimum fabrication cost, considering four design variables: x_1, x_2, x_3, x_4 and constraints of shear stress τ, bending stress in the beam σ, buckling load on the bar P_c, and end deflection on the beam δ. The optimization model is summarized in the next equation:

$$\begin{aligned} &\text{Minimize: } f(\mathbf{x}) \\ &\quad = 1.10471 x_1^2 x_2 + 0.04811 x_3 x_4 \left(14.0 + x_2\right) \end{aligned} \tag{A.1}$$

subject to the following physical constraints:

$$g_1(\mathbf{x}) = \tau(\mathbf{x}) - 13600 \leq 0,$$

$$g_2(\mathbf{x}) = \sigma(\mathbf{x}) - 30000 \leq 0,$$

$$g_3(\mathbf{x}) = x_1 - x_4 \leq 0,$$

$$g_4(\mathbf{x}) = 0.10471\left(x_1^2\right) + 0.04811x_3x_4\left(14 + x_2\right) - 5.0 \le 0,$$
$$g_5(\mathbf{x}) = 0.125 - x_1 \le 0,$$
$$g_6(\mathbf{x}) = \delta(\mathbf{x}) - 0.25 \le 0,$$
$$g_7(\mathbf{x}) = 6000 - Pc(\mathbf{x}) \le 0, \tag{A.2}$$

with

$$\tau(\mathbf{x}) = \sqrt{(\tau')^2 + \left(2\tau'\tau''\right)\frac{x_2}{2R} + (\tau'')^2},$$
$$\tau' = \frac{6000}{\sqrt{2}x_1x_2},$$
$$\tau'' = \frac{MR}{J},$$
$$M = 6000\left(14 + \frac{x_2}{2}\right),$$
$$R = \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2},$$
$$J = 2\left\{x_1x_2\sqrt{2}\left[\frac{x_2^2}{12} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\},$$
$$\sigma(\mathbf{x}) = \frac{504000}{x_4x_3^2},$$
$$\delta(\mathbf{x}) = \frac{65856000}{\left(30 \times 10^6\right)x_4x_3^3},$$
$$Pc(\mathbf{x}) = \frac{4.013\left(30 \times 10^6\right)\sqrt{x_3^2x_4^6/36}}{196} \times\left(1 - \frac{x_3\sqrt{\left(30 \times 10^6\right)/4\left(12 \times 10^6\right)}}{28}\right), \tag{A.3}$$

with side constraint: $0.1 \le x_1, x_4 \le 2.0$, and $0.1 \le x_2, x_3 \le 10.0$.

Best known solution: $x^* = (0.205730, 3.470489, 9.036624, 0.205729)$, where $f(x^*) = 1.724852$.

E02: Pressure Vessel Design Optimization Problem. The problem considers a compressed air storage tank with a working pressure of 3000 psi and a minimum volume of 750 ft^3. A cylindrical vessel is capped at both ends by hemispherical heads. Using rolled steel plate, the shell is made in two halves that are joined by two longitudinal welds to form a cylinder. The objective is minimize the total cost, including the cost of the materials forming the welding [30]. The design variables are thickness x_1, thickness of the head x_2, the inner radius x_3, and the length of the cylindrical section of the vessel x_4. The variables x_1 and x_2 are discrete values which are integer multiples of 0.0625 inch. Then, the formal statement is

$$\text{Minimize: } f(\mathbf{x}) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3, \tag{A.4}$$

subject to

$$g_1(\mathbf{x}) = -x_1 + 0.0193x_3 \le 0,$$
$$g_2(\mathbf{x}) = -x_2 + 0.00954x_3 \le 0,$$
$$g_3(\mathbf{x}) = -\pi x_3^2x_4^2 - \frac{4}{3}\pi x_3^3 + 1296000 \le 0,$$
$$g_4(\mathbf{x}) = x_4 - 240 \le 0, \tag{A.5}$$

with side constraints $1 \times 0.0625 \le x_1, x_2 \le 99 \times 0.0625$, $10.0 \le x_3$, and $x_4 \le 200.0$. As x_1 and x_2 deal with integer values, they are truncated to the nearest integer.

Best known solution: $x^* = (0.8125, 0.4375, 42.098446, 176.636596)$, where $f(x^*) = 6059.714335$.

E03: Speed Reducer Design Optimization Problem. The design of the speed reducer [31] is considered with the face width x_1, module of teeth x_2, number of teeth on pinion x_3, length of the first shaft between bearings x_4, length of the second shaft between bearings x_5, diameter of the first shaft x_6, and diameter of the first shaft x_7 (all variables are continuous except x_3 that is an integer). The weight of the speed reducer has to be minimized subject to constraints on bending stress of the gear teeth, surface stress, transverse deflections of the shafts, and stresses in the shaft. The problem is

$$\text{Minimize: } f(\mathbf{x}) = 0.7854x_1x_2^2\left(3.3333x_3^2 + 14.9334x_3 - 43.0934\right) - 1.508x_1\left(x_6^2 + x_7^2\right) + 7.4777\left(x_6^3 + x_7^3\right) + 0.7854\left(x_4x_6^2 + x_5x_7^2\right), \tag{A.6}$$

subject to

$$g_1(\mathbf{x}) = \frac{27}{x_1x_2^2x_3} - 1 \le 0,$$
$$g_2(\mathbf{x}) = \frac{397.5}{x_1x_2^2x_3^2} - 1 \le 0,$$
$$g_3(\mathbf{x}) = \frac{1.93x_4^3}{x_2x_3x_6^4} - 1 \le 0,$$
$$g_4(\mathbf{x}) = \frac{1.93x_5^3}{x_2x_3x_7^4} - 1 \le 0,$$
$$g_5(\mathbf{x}) = \frac{1.0}{110x_6^3}\sqrt{\left(\frac{745.0x_4}{x_2x_3}\right)^2 + 16.9 \times 10^6} - 1 \le 0,$$

$$g_6(\mathbf{x}) = \frac{1.0}{85x_7^3}\sqrt{\left(\frac{745.0x_5}{x_2x_3}\right)^2 + 157.5 \times 10^6} - 1 \le 0,$$

$$g_7(\mathbf{x}) = \frac{x_2x_3}{40} - 1 \le 0,$$

$$g_8(\mathbf{x}) = \frac{5x_2}{x_1} - 1 \le 0,$$

$$g_9(\mathbf{x}) = \frac{x_1}{12x_2} - 1 \le 0,$$

$$g_{10}(\mathbf{x}) = \frac{1.5x_6 + 1.9}{x_4} - 1 \le 0,$$

$$g_{11}(\mathbf{x}) = \frac{1.1x_7 + 1.9}{x_5} - 1 \le 0, \tag{A.7}$$

with side constraints $2.6 \le x_1 \le 3.6$, $0.7 \le x_2 \le 0.8$, $17 \le x_3 \le 28$, $7.3 \le x_4 \le 8.3$, $7.8 \le x_5 \le 8.3$, $2.9 \le x_6 \le 3.9$, and $5.0 \le x_7 \le 5.5$.

Best known solution: $x^* = (3.500000, 0.7, 17, 7.300000, 7.800000, 3.350214, 5.286683)$, where $f(x^*) = 2996.348165$.

E04: Tension/Compression Spring Design Optimization Problem. This problem [32, 33] minimizes the weight of a tension/compression spring subject to constraints of minimum deflection, shear stress, surge frequency, and limits on outside diameter and on design variables. There are three design variables: the wire diameter x_1, the mean coil diameter x_2, and the number of active coils x_3. The mathematical formulation of this problem is

$$\text{Minimize: } f(\mathbf{x}) = (x_3 + 2)\, x_2x_1^2, \tag{A.8}$$

subject to

$$g_1(\mathbf{x}) = 1 - \frac{x_2^3x_3}{7178x_1^4} \le 0,$$

$$g_2(\mathbf{x}) = \frac{4x_2^2 - x_1x_2}{12566\left(x_2x_1^3\right) - x_1^4} + \frac{1}{5108x_1^2} - 1 \le 0,$$

$$g_3(\mathbf{x}) = 1 - \frac{140.45x_1}{x_2^2x_3} \le 0, \tag{A.9}$$

$$g_4(\mathbf{x}) = \frac{x_2 + x_1}{1.5} - 1 \le 0,$$

with side constraints $0.05 \le x_1 \le 2.0$, $0.25 \le x_2 \le 1.3$, and $2.0 \le x_3 \le 15.0$.

Best known solution: $x^* = (0.051690, 0.356750, 11.287126)$, where $f(x^*) = 0.012665$.

Acknowledgment

The research work of the second author is partially funded by CONICET (Argentina).

References

[1] A. Ben-Tal and A. Nemirovski, "Robust optimization—methodology and applications," *Mathematical Programming*, vol. 92, no. 3, pp. 453–480, 2001.

[2] D. Bertsimas and M. Sim, "Robust discrete optimization under ellipsoidal uncertainty sets," Technical Report Working Paper, MIT, 2004.

[3] Y. Kanno and I. Takewaki, "Evaluation and maximization of robustness of trusses by using semidefinite programming," in *Proceedings of the 6th World Congress on Structural and Multidisciplinary Optimization (WCSMO '05)*, J. Herskovits, S. Mazorche, and A. Canelas, Eds., Rio de Janeiro, Brazil, 2005.

[4] X. Du and W. Chen, "Towards a better understanding of modeling feasibility robustness in engineering design," *Journal of Mechanical Design, Transactions of the ASME*, vol. 122, no. 4, pp. 385–394, 2000.

[5] D. Indraneel, "Robustness optimization for constrained nonlinear programming problems," *Engineering Optimization*, vol. 32, no. 5, pp. 585–618, 2000.

[6] I. Doltsinis and Z. Kang, "Robust design of structures using optimization methods," *Computer Methods in Applied Mechanics and Engineering*, vol. 193, no. 23–26, pp. 2221–2237, 2004.

[7] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Evolutionary Computation Proceedings*, IEEE World Congress on Computational Intelligence, pp. 69–73, 1998.

[8] E. Mezura-Montes and C. A. Coello Coello, "Useful infeasible solutions in engineering optimization with evolutionary algorithms," in *Proceedings of the 4th Mexican international Conference on Advances in Artificial Intelligence (MICAI '05)*, pp. 652–662, Springer, 2005.

[9] T. Takahama and S. Sakai, "Solving constrained optimization problems by the ε constrained particle swarm optimizer with adaptive velocity limit control," in *Proceedings of the IEEE Conference on Cybernetics and Intelligent Systems*, pp. 1–7, June 2006.

[10] A. Kaveh and S. Talatahari, "Engineering optimization with hybrid particle swarm and ant colony optimization," *Asian Journal of Civil Engineering*, vol. 10, pp. 611–628, 2009.

[11] E. Zahara and Y. T. Kao, "Hybrid Nelder-Mead simplex search and particle swarm optimization for constrained engineering design problems," *Expert Systems with Applications*, vol. 36, no. 2, pp. 3880–3886, 2009.

[12] Y. Wang, Z. Cai, Y. Zhou, and Z. Fan, "Constrained optimization based on hybrid evolutionary algorithm and adaptive constraint-handling technique," *Structural and Multidisciplinary Optimization*, vol. 37, no. 4, pp. 395–413, 2009.

[13] A. H. Kayhan, H. Ceylan, M. T. Ayvaz, and G. Gurarslan, "Psolver: a new hybrid particle swarm optimization algorithm for solving continuous optimization problems," *Expert Systems with Applications*, vol. 37, pp. 6798–6808, 2010.

[14] Y. Zhou and S. Pei, "A hybrid co-evolutionary particle swarm optimization algorithm for solving constrained engineering design problems," *Journal of Computers*, vol. 5, no. 6, pp. 965–972, 2010.

[15] M. Tuba, N. Bacanin, and N. Stanarevic, "Guided artificial bee colony algorithm," in *Proceedings of the 5th European Conference on European Computing Conference (ECC '11)*, pp. 398–403, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisc, USA, 2011.

[16] H. Nahvi and I. Mohagheghian, "A Particle Swarm Optimization algorithm for mixed variable nonlinear problems,"

International Journal of Engineering, Transactions A: Basics, vol. 24, no. 1, pp. 65–78, 2011.

[17] A. Baykasoglu, "Design optimization with chaos embedded great deluge algorithm," *Applied Soft Computing*, vol. 12, no. 3, pp. 1055–1067, 2012.

[18] X. Chen and Y. Li, "Enhance computational efficiency of neural network predictive control using pso with controllable random exploration velocity," in *Proceedings of the 4th International Symposium on Neural Networks: Advances in Neural Networks (ISNN '07)*, pp. 813–823, Springer, 2007.

[19] R. Eberhart and J. Kennedy, "New optimizer using particle swarm theory," in *Proceedings of the 6th International Symposium on Micro Machine and Human Science (MHS '95)*, pp. 39–43, October 1995.

[20] M. Clerc and J. Kennedy, "The particle swarm-explosion, stability, and convergence in a multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, 2002.

[21] X. Chen and Y. Li, "A modified PSO structure resulting in high exploration ability with convergence guaranteed," *IEEE Transactions on Systems, Man, and Cybernetics B*, vol. 37, no. 5, pp. 1271–1289, 2007.

[22] A. Kaveh, S. Talatahari, and B. Farahmand Azar, "An improved hpsaco for engineering optimum design problems," *Asian Journal of Civil Engineering*, vol. 12, no. 2, pp. 133–141, 2010.

[23] Y. Liu and B. Niu, "A novel pso model based on simulating human social communication behavior," *Discrete Dynamics in Nature and Society*, vol. 2012, Article ID 791373, 21 pages, 2012.

[24] L. Cagnina, S. Esquivel, and C. Coello-Coello, "A particle swarm optimizer for constrained numerical optimization," in *Parallel Problem Solving from Nature—PPSN IX*, T. Runarsson, H. G. Beyer, E. Burke, J. Merelo-Guervs, L. Whitley, and X. Yao, Eds., vol. 4193 of *Lecture Notes in Computer Science*, pp. 910–919, Springer, Berlin, Germany, 2006.

[25] L. Cagnina and S. Esquivel, "Global numerical optimization with a bi-population particle swarm optimizer," in *Proceedings of the 13rd Congreso Argentino en Ciencias de la Computación (CACIC '07)*, pp. 1452–1463, Corrientes, Argentina, 2007.

[26] J. Kennedy and R. Eberhart, "Bare bones particle swarms," in *Proceedings of the IEEE Swarm Intelligence Symposium*, pp. 80–89, 2003.

[27] M. D. Mckay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 42, no. 1, pp. 55–61, 2000.

[28] 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., University of Science and Technology of China (USTC), School of Computer Science and Technology, Nature Inspired Computation and Applications Laboratory (NICAL), Anhui, China, 2010.

[29] K. M. Ragsdell and D. T. Phillips, "Optimal design of a class of welded structures using geometric programming," *ASME Journal of Engineering for Industries*, vol. 98, no. 3, pp. 1021–1025, 1976.

[30] E. Sandgren, "Nonlinear integer and discrete programming in mechnical design optimization," *Journal of Mechanisms, Transmissions, and Automation in Design*, vol. 112, no. 2, pp. 223–229, 1990.

[31] J. Golinski, "An adaptive optimization system applied to machine synthesis," *Mechanism and Machine Theory*, vol. 8, no. 4, pp. 419–436, 1973.

[32] J. Arora, *Introduction to Optimum Design*, McGraw-Hill, New York, NY, USA, 1989.

[33] A. Belegundu, *A study of mathematical programming methods for structural optimization [Ph.D. thesis]*, Department of Civil Environmental Engineering, University of Iowa, Iowa City, Iowa, USA, 1982.