A Particle Swarm Optimizer for Constrained Numerical Optimization

Leticia C. Cagnina[1], Susana C. Esquivel[1], and Carlos A. Coello Coello[2]

[1] LIDIC (Research Group). Universidad Nacional de San Luis - Ej. de Los Andes 950 - (5700) San Luis, Argentina.
{lcagnina,esquivel}@unsl.edu.ar

[2] CINVESTAV-IPN (Evolutionary Computation Group), Computer Science Section, Av. IPN No. 2508, Col. San Pedro Zacatenco, Mexico D.F. 07300, MEXICO.
ccoello@cs.cinvestav.mx

Abstract. This paper presents a particle swarm optimizer to solve constrained optimization problems. The proposed approach adopts a simple method to handle constraints of any type (linear, nonlinear, equality and inequality), and it also presents a novel mechanism to update the velocity and position of each particle. The approach is validated using standard test functions reported in the specialized literature and it's compared with respect to algorithms representative of the state-of-the-art in the area. Our results indicate that the proposed scheme is a promising alternative to solve constrained optimization problems using particle swarm optimization.

1 Introduction

Constraints are usually adopted in any sort of real-world optimization problems (e.g., in engineering, in cutting and packing problems, in VLSI design, etc.). The unconstrained nature of evolutionary algorithms (EA) makes it necessary to design schemes to incorporate the constraints of a problem into the fitness function [2]. Despite the popularity of penalty functions, they have certain limitations from which the main one has to do with the difficulties to define accurate penalty factors that allow an EA an efficient exploration of the search space (particularly when dealing with problems in which the global optimum lies on the boundary between the feasible and the infeasible regions).

Constrained optimization problems have been extensively studied in Mathematical Programming. However, despite the existence of a considerable number of deterministic optimization algorithms, there is no single approach that can guarantee convergence for the general nonlinear programming problem, which is the one of interest to us [8].

In the last few years, several metaheuristics have been adopted for numerical optimization. One of such metaheuristics which has become increasingly popular is particle swarm optimization (PSO) [6]. PSO is based on the metaphor of how some species share information and then use it for moving to those places where the food is located. The population is a set of individuals named *particles* which represent possible solutions within a multidimensional search space. The particles are characterized by a position, a velocity of exploration and a record of their past behavior. All of these are constantly updated in an iterative process. In this paper, we adopt PSO for solving constrained optimization problems.

The remainder of the paper is organized as follows. Section 2 provides the statement of the problem of our interest. In Section 3, we present a brief literature review. Section 4 describes our proposed approach. The experimental setup and the analysis of our results are presented in Section 5. Finally, our conclusions and some possible paths for future research are presented in Section 6.

2 Statement of the Problem

The problem of interest to us is the general *nonlinear programming problem* which is defined as the problem of finding $\boldsymbol{x}$ which optimizes the objective function:

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

where $f(\boldsymbol{x})$ is subject to:

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

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

$x_d \in [l_d, u_d]$ with $d \in [1..D]$. 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 $\boldsymbol{x}$ satisfying all inequality and equality constraint functions determine the feasible solution space $\mathcal{F}$.

3 Literature Review

Despite the popularity of PSO as a numerical optimizer, there is relatively little work regarding its use in constrained optimization problems. Next, we will review the most representative research within this area.

Zhang et al. [11] presented a PSO algorithm with a *periodic* mode of handling constraints. This technique makes periodic copies of the search space when the algorithm starts the run. In that way, it avoids the disorganization that may arise when the mutation operator is applied to those particles lying on the boundary between the feasible and infeasible regions. The authors tested their algorithm with a low number of evaluations (28,000 and 140,000) in eight test functions. They performed 100 runs for each test function and compared the performance of their approach with respect to the results provided by conventional constraint-handling methods (i.e., penalty functions). However, no comparisons are provided with respect to state-of-the-art constraint-handling techniques.

Toscano Pulido and Coello Coello [10] 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 to improve the exploration of the search space. This operator changes the flight of particles to different zones. The algorithm was tested with a relatively large population size and a low number of iterations as to perform 340,000 evaluations of the objective function. Thirteen benchmark constrained functions from [9] were used to show the performance of this PSO. The authors concluded that their results were highly competitive.

Parsopoulos et al. [7] proposed a *Unified Particle Swarm Optimization* version and adapted it to handle constraints. They included a penalty function technique which uses the number of constraints that are violated and the degree of violation. The algorithm preserves the feasibility of the best solutions. They tested their version with four constrained engineering optimization problems with promising results.

4 Our Proposed Approach

In this section, we present our proposal of a Constrained Particle Swarm Optimizer (called **CPSO**). In CPSO, each particle consists of a n-dimensional real number vector (where n refers to the number of decision variables of the problem to be solved). Each dimension of a particle corresponds to a decision variable of the problem. The particles are evaluated using a fitness function which has some constraints. There are a several constraint-handling approaches that tend to add information about the distance from each individual to the feasible region into the fitness function in order to guide the search. One of the simplest methods (which was implemented in our algorithm) prefers to choose a feasible individual over an infeasible one. When the algorithm evaluates infeasible particles, it prefers the infeasible individuals that are closer to the feasible region. To determine the infeasibility degree, CPSO saves the largest violation obtained for each constraint. Then, when a particle is detected to be infeasible, the algorithm adds the amount of violation that corresponds to that particle (normalized with respect to the largest violation recorded so far). This approach was used in the PSO strategy to choose the best values: gbest, lbest and the best value reached by each particle. Thus, the equations to update velocity and position use the "best" feasible solution, or the infeasible solution which is closest to the feasible region (if there are feasible particles in the swarm).

Most constraint-handling techniques used in evolutionary algorithms tend to deal only with inequality constraints because equalities are very difficult to handle. To transform an equality constraint into an inequality we use:

$$|h_e(\boldsymbol{x})| - \epsilon \leq 0 \ . \tag{4}$$

where ϵ is the tolerance allowed. By adopting this transformation, our CPSO only deals with inequality constraints.

As in the basic PSO, our algorithm records the best position found so far for each particle (*gbest* approach) or in the neighborhood (*lbest* approach) if a neighborhood topology is implemented. These values are used to update the velocity and position of the particles. It is known that the *gbest* approach works well in many problems, but tends to converge to a local optimum in some cases [1]. For those cases, the *lbest* approach works better because it records the best value reached by a smaller group of particles, instead of considering the entire swarm.

We empirically found that a combination of the two approaches worked well in our CPSO. With *gbest*, the algorithm explores better and with *lbest*, we avoid stagnation. Thus, we modified the equation for computing the velocity (used to update the position of a particle) in the following way:

$$v_{id} = w(v_{id} + c_1 r_1 (p_{id} - part_{id}) + c_2 r_2 (p_{ld} - part_{id}) + c_3 r_3 (p_{gd} - part_{id})) \quad (5)$$

$$part_{id} = part_{id} + v_{id} \quad (6)$$

where v_{id} is the velocity of the particle i at the dimension d, w is the inertia factor [3] whose goal is to balance global exploration and local exploitation, c_1 is the personal learning factor, and c_2, c_3 are the social learning factors, r_1, r_2 and r_3 are three 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 and $part_{id}$ is the value of the particle i at the dimension d.

To compute the p_{ld} value, we used a circle topology [4], in which each particle is connected to k neighbors. The neighbors are determined by the position of the particles in the structure. Figure 1 illustrates this concept.

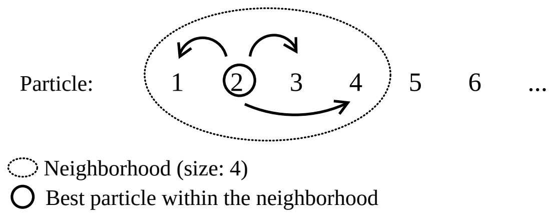

Fig. 1. Circle topology.

It is well known that it is important to maintain the population's diversity to avoid stagnation (i.e., convergence to a local optimum). In order to meet this goal, we adopted a dynamic mutation operator, which was applied to each particle with a probability *pm*. This probability uses the total number of cycles and the current cycle number in the following equation:

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

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

We empirically found that for some difficult functions, our CPSO could not find good values. The reason was its diversification of solutions which kept the approach from converging. In order to overcome this problem, we changed the common update equation (eq. (6)) of particles for the update equation presented by Kennedy [5] in the so-called *Gaussian Bare Bones PSO*. In that algorithm, the new 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 its neighborhood. The standard deviation is the difference between these two values. Then, the position was updated using the following equation:

$$part_i = N\left(\frac{p_i + p_l}{2}, |p_i - p_l|\right) \qquad (8)$$

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 $part_i$ and the best position reached by any particle in the neighborhood of $part_i$. CPSO used this equation to update particles with a certain probability (a 50% probability was adopted to select between equation (6) and equation (8)). We choose those probabilities ($\langle 0.5, 0.5\rangle$) because we determined it was the best combination to be used to select between equations (6) and equation (8) for updating the particles. We performed a series of previous experiments (which are shown in Table 1) using the 3 functions in which CPSO had more difficulties to obtain good values: functions 2, 6 and 13. The notation $\langle r, s\rangle$ means that we selected equations (6) with a probability r and equation (8) with a probability s. Figure 2 shows the pseudo-code of our CPSO.

Table 1. Best Values obtained with *CPSO*, performing 340,000 evaluations with different probabilities of selection for the updating equations.

Function	Best Known Value	$\langle 0.1, 0.9\rangle$	$\langle 0.5, 0.5\rangle$	$\langle 0.9, 0.1\rangle$
2	-0.803619	0.801825	-0.801388	-0.757889
6	-6961.814	-6962.046	-6961.825	-6827.984
13	0.053950	0.157094	0.054237	0.316460

5 Parameter Settings and Analysis of Results

The CPSO algorithm was tested using the thirteen constrained test functions adopted in [9]. We performed 30 independent runs for each function. Our results are compared with respect to the PSO-based approach which currently is the most competitive reported in the specialized literature for constrained optimization (i.e., the approach by Toscano Pulido and Coello Coello [10]). Additionally, we also compared our results with respect to Stochastic Ranking [9], which is a constraint-handling technique representative of the state-of-the-art in the area.

```
0. CPSO:
1. Swarm Initialization
2.    FOR i=1 TO number of particles DO
3.        FOR j=1 TO number of dimensions DO
4.            Initialize $part_{ij}$
5.            Initialize $vel_{ij}$
7.        END
8.    END
9.    Evaluate fitness
10.   Record pbest
11.   Record gbest
12. Swarm flights through the search space
13.    DO
14.       FOR i=1 TO number of particles DO
15.           Search the best leader in the
                 neighborhood of $part_i$
                 and record in $lbest_i$
16.           IF flip$(0.5)$
17.              FOR j=1 TO number of dimensions DO
18.                  Update $vel_{ij}$
19.                  Update $part_{ij}$ using eq. $(6)$
20.              END
21.           ELSE
22.              gaussian update using eq. $(8)$
23.           END
24.       END
25.       Keeping particles
26.       Update $pm$
27.       Mutate every particle depending on $pm$
28.       Evaluate fitness$(part_i)$
29.       Record pbest
30.       Record gbest
31.    WHILE$(current\_cycle < max\_cycle)$
```

Fig. 2. Pseudo-code of our proposed CPSO

Stochastic Ranking was validated performing 350,000 objective function evaluations per run. However, the approach from [10] performed 340,000 objective function evaluations per run. Thus, in order to allow a fair comparison, we performed experiments with only 340,000 objective function evaluations. The parameters of our approach are the following: swarm size = 10 particles, *pm_min* = 0.1, *pm_max* = 0.4, neighborhood size = 4, the inertia factor w was set randomly with a value within the range [0.8,0.9], and learning factors c_1, c_2, and c_3 were randomly chosen within the range [1.8,1.9]. The parameter settings such as the probability of mutation, neighborhood size, inertia and learning factors were empirically derived after numerous experiments. As we stated in Section 3, we transformed the equality constraints into inequality constrains, using $\epsilon = 0.0001$. This tolerance causes that the algorithm identifies as feasible some constraints which are being slightly violated. That is the reason why some results reported in the present work are better than the reference solutions previously reported.

Table 2 displays the results obtained with 3 different algorithms: our version of $CPSO$ with 340,000 evaluations, the algorithm presented by Toscano Pulido and Coello Coello [10] (PSO_{Tos}) and Stochastic Ranking (SR) [9]. The best result found for each function is marked with *italics*.

Table 2. Best results obtained by our $CPSO$, PSO_{Tos} (with 340,000 objective function evaluations), and SR (with 350,000 objective function evaluations).

Function	Type	Best Known Value	$CPSO$	PSO_{Tos}	SR
1	Min	-15.000	*-15.000*	*-15.000*	*-15.000*
2	Max	-0.803619	-0.801388	-0.803432	*-0.803515*
3	Max	1.000	*1.000*	1.004	*1.000*
4	Min	-30665.539	-30665.659	-30665.500	*-30665.539*
5	Min	5126.498	*5126.497*	5126.640	*5126.497*
6	Min	-6961.814	-6961.825	-6961.810	*-6961.814*
7	Min	24.306	24.400	24.351	*24.307*
8	Max	0.095825	*0.095825*	*0.095825*	*0.095825*
9	Min	680.630	680.636	680.638	*680.630*
10	Min	7049.3307	*7052,8523*	7057.5900	7054.316
11	Min	0.750	0.749	0.749	*0.750*
12	Max	1.000	*1.000*	*1.000*	*1.000*
13	Min	0.053950	0.054237	0.068665	*0.053957*

Comparing our best results with respect to PSO_{Tos} (Table 2), our approach was able to improve its best results in five test functions: 3, 5, 9, 10 and 13 (it is worth remarking that functions 10 and 13 are among the most difficult from the benchmark considered). PSO_{Tos} outperforms $CPSO$ in test functions 2 and 7. Additionally, in functions 4 and 6, our $CPSO$ did not reach the optimum values while PSO_{Tos} obtained values lower than the best reported values due to rounding errors on the constraints. Comparing our $CPSO$ with respect to SR, we can observe that $CPSO$ obtained better values for function 10, equal values for five test functions (1, 3, 5, 8, 12) and SR found slightly better results for the rest of the problems (2, 4, 6, 7, 9, 11 y 13). However, it is important to note that both PSO algorithms obtained their results with a lower computational cost (measured in terms of the number of evaluations of the objective functions), since they performed 340,000 objective function evaluations, whereas Stochastic Ranking performed 350,000 objective function evaluations.

Table 3. Best, Mean and Worst Values Obtained with $CPSO$, performing 340,000 objective function evaluations.

Function	Best	Mean	Worst
1	-15.000	-15.0001	-134.2191
2	-0.801388	0.7653	0.0917
3	1.000	1.0000	1.0000
4	-30665.659	-30665.6564	-25555.6267
5	5126.497	5327.9569	2300.5443
6	-6961.825	-6859.0759	64827.5545
7	24.400	31.4854	4063.5252
8	0.095825	0.0958	-0.0006
9	680.636	682.3973	18484.7591
10	7052,8523	8533.6999	13123.4656
11	0.749	0.7505	0.4466
12	1.000	1.000	9386
13	0.054237	1.4139	0.9675

The mean and worst values obtained by PSO_{Tos} and SR (Tables 4 and 5) are both better that those of $CPSO$ (Table 3). We believe that this fact is due to the mechanism implemented to maintain the swarm's diversity. However, this mechanism provided a trade-off that we considered acceptable, since the best values found remained competitive despite the larger variability of results obtained. Note that in Table 3 some mean values (functions 2, 4, 5, 11 and 13) do not fall within the best and the worst values. This is because the worst values reached by $CPSO$ are not feasible while the best values are feasible. We believe the same occurs with PSO_{Tos} and SR (Tables 4 and 5) in some cases.

6 Conclusions and Future Work

We have introduced a new proposal to solve constrained optimization problems using particle swarm optimization. Our approach uses simple selection rules for handling the constraints of a problem, and adopts both the local and the global best models to update the particles of the swarm. Our best results are very competitive in most cases, even with respect to Stochastic Ranking (which is the best constraint-handling technique known to date) although they present

Table 4. Best, Mean and Worst Values Obtained with PSO_{Tos}, performing 340,000 objective function evaluations.

Function	Best	Mean	Worst
1	-15.000	-15.0000	-15.0000
2	-0.803432	0.790406	0.750393
3	1.004	1.0038	1.0024
4	-30665.500	-30665.5000	-30665.5000
5	5126.640	5461.0813	6104.7500
6	-6961.810	-6961.8100	-6961.8100
7	24.351	25.3557	27.3168
8	0.095825	0.0958	0.0958
9	680.636	680.8523	680.5530
10	7057.5900	7560.0478	8104.3100
11	0.749	0.7501	0.7528
12	1.000	1.0000	1.0000
13	0.068665	1.7164	13.6695

Table 5. Best, Mean and Worst Values Obtained with SR, performing 350,000 evaluations.

Function	Best	Mean	Worst
1	-15.000	-15.0000	-15.0000
2	-0.803515	0.781975	0.726288
3	1.000	1.0000	1.0000
4	-30665.500	-30665.5000	-30665.5000
5	5126.539	5128.8810	5142.4720
6	-6961.814	-6875.9400	-6350.2620
7	24.307	24.3740	24.6420
8	0.095825	0.0958	0.0958
9	680.630	680.6560	680.7630
10	7054.3160	7559.1920	8835.6550
11	0.750	0.7500	0.7500
12	1.000	1.0000	1.0000
13	0.053957	0.0570	0.2169

a high variability in some cases. Additionally, in several cases our approach outperformed a previous PSO-based constraint-handling scheme.

As part of our future work, we aim to study alternative schemes to maintain diversity. Another goal is to improve the robustness of our approach, so that the variability of results significantly decreases, without degrading the quality of the best solutions currently found.

Acknowledgements

The third author acknowledges support from CONACyT through project number 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. Carlos A. Coello Coello. Theoretical and Numerical Constraint Handling Techniques used with Evolutionary Algorithms: A Survey of the State of the Art. *Computer Methods in Applied Mechanics and Engineering*, 191(11-12):1245–1287, January 2002.
3. R. Eberhart and Y. Shi. A modified particle swarm optimizer. In *International Conference on Evolutionary Computation, IEEE Service Center*, Anchorage, AK, Piscataway, NJ, 1998.
4. 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.
5. J. Kennedy. Bare bones particle swarms. In *IEEE Swarm Intelligence Symposium*, pages 80–87, 2003.
6. J. Kennedy and R. C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, California, USA, 2001.
7. K. E. Parsopoulos and M. N. Vrahatis. Unified particle swarm optimization for solving constrained engineering optimization problems. In *Lecture Notes of Computer Science*, pages 582–591, Springer-Verlag Berlin Heidelberg, 2005.
8. Singiresu S. Rao. *Engineering Optimization*. John Wiley & Sons, 3rd edition, 1996.
9. T. P. Runarsson and X. Yao. Stochastic ranking for constrained evolutionary optimization. In *IEEE Transactions on Evolutionary Computation*, volume 3, pages 284–294, 2000.
10. G. Toscano Pulido and C. A. Coello Coello. A constrained-handling mechanism for particle swarm optimization. In *Congress on Evolutionary Computation*, pages 1396–1403, Portland, Oregon, USA, 2004.
11. W. Zhang, X. Xie, and D. Bi. Handling boundary constraints for numerical optimization by particle swarm flying in periodic search space. In *Congress on Evolutionary Computation*, pages 2307–2311, Portland, Oregon, USA, 2004.