A Modified Binary-PSO for Continuous Optimization

Alina Orellana - Gabriela Minetti
Laboratorio de Investigación en Sistemas Inteligentes
Universidad Nacional de La Pampa
República Argentina
orellanaalina@gmail.com - minettig@ing.unlpam.edu.ar

No Institute Given

Abstract. Metaheuristics based on swarm intelligence simulate the behavior of a biological social system like as a flock of birds or a swarm of bees, and they have achieved important advances for solving optimization problems. In this paper, we propose a variant for a particular kind of those metaheurisitcs: Particle Swarm Optimization (PSO). This modification arises after discovering a low rate of convergence produced by a high level of dispersal at the swarm. Finally, we analyzed and compared the results obtained by an original PSO algorithm and our proposal. From those, we can see the improvement obtained by our variant since it allows to explore more the search space.

1 Introduction

Metaheuristics are high level strategies that use different methods for exploring search spaces and solving problems. Some metaheuristics are known as trajectory methods, since they perform a trajectory in the search space and their main characteristic is they work with only one solution by iteration. Other metaheuristics are based on population (a set of solutions at time), they are inspired in different social or biological processes. Particularly we work with the Particle Swarm Optimization (PSO) metaheuristic which is an important area of the Swarm Intelligence, [1, 2].

Those methods reproduce the behavior of real swarms or insect colonies, that is they act as intelligent social organizations. In other words, the richness of these models is accentuated on the high level of social organization, the structural composition of individuals, the ability to adapt to environmental changes and provide operational continuity when one or more elements fail individually. Such social behavior defines movements of the decision variables in the search space and guiding to optimal solutions.

As the Genetic Algorithms (GAs) [3], PSO consists of an iterative and stochastic process that operates on a set of potential solutions to solve a given problem. Specifically, each solution or individual is considered a particle and a population is considered a swarm of particles. However, PSO uses an operator

movement, which alters the particle throughout the process, instead of evolutionary operators.

The particles need to maintain a minimum distance between them throughout the process. Besides, each individual should avoid an excessive distance to the rest of the group. In this way, the individual particle will be attracted to others, thus avoiding being alone and giving rise to different neighborhoods depending on the strength of attraction.

Like most metaheuristics based on population, PSO presents some drawbacks in their original version. That led to researchers to propose small changes to the configuration of its different parameters, giving rise to several alternative models of PSO [4]. Furthermore, noteworthy several hybrid versions of PSO have been developed. In order to do that different functions for updating position and velocity vectors, different procedure of particle selection to retrieve information for the social component, dynamic updates of parameters, incorporation of operators and different procedures of swarm initialization have been evaluated and used [5, 6, 7].

Since the mid '90s to the present, PSO has been used to solve a wide range of problems. For example: single-objective, multi-objective, engineering problems, among others [8, 9, 10].

In this paper, we show the behavior of an original binary PSO to optimize functions. After to study the strengths and weaknesses of PSO, we propose a new version which incorporates features from other metaheuristics. Particularly we use characteristics of GAs and CHC (*Cross generational elitist selection, Heterogeneous recombination, and Cataclysmic mutation*)a special kind of GAs [11]. Both PSO versions are evaluated considering four optimization functions, which represent many optimization problems. We try to provide empirical evidence for the practical usefulness of this metaheuristic.

The rest of this article is organized as follows. The next describes the original PSOs. Section 3 introduces and explains our proposal. The section 4 introduces the optimization functions used to test the PSO algorithms. In section 5 the experiments and discussions of their results are shown. Finally, the last section concludes and provides hints on further research.

2 Particle Swarm Optimization and binary codification

As we said above, for PSO, each individual is a particle (p_i) and the population is a swarm of particles (S). For each particle, the following attributes are identified: a current position (X_i), a current velocity (V_i), the best position achieved so far by each particle ($pBest_i$), and the best solution found so far by its neighbors ($gBest_i$).

The PSO algorithm starts generating random position and velocity vectors for each particle. After that, those vectors are updated according to the following functions:

$$V_i = W * V_i + \vartheta_1 * r_1 * (pBest_i - X_i) + \vartheta_2 * r_2 * (gBest_i - X_i) \quad (1)$$

$$X_i = X_i + V_i \quad (2)$$

Algorithm 1 PSOo

```
S ← initializeSwarm()
while not stop condition do
  for i = 0; i < size(S); i++ do
    if fitness(X_i) is better than fitness(pBest_i) then
      pBest_i ← X_i
      fitness(pBest_i) ← fitness(X_i)
    end if
    if fitness(pBest_i) is better than fitness(gBest_i) then
      gBest_i ← pBest_i
      fitness(gBest_i) ← fitness(pBest_i);
    end if
  end for
  //derivation 0
  for i = 0; i < size(S); i++ do
    V_i ← W * V_i + ϑ1 * r1 * (pBest_i − X_i) + ϑ2 * r2 * (gBest_i − X_i)
    X_i ← X_i + V_i
    X_i ← (4 + X_i) mod 2
    V_i ← (3 + V_i) mod 3 − 1
  end for
end while
```

Where W is the *inertia weight* [12] that regulates the impact of previous velocity vector on the new vector. ϑ_1 is a coefficient that controls the knowledge component ($\vartheta_1 * r_1 * (pBest_i - X_i)$. This component measures the influence of the knowledge obtained by a particle (its best position) for calculating its new velocity vector. While the ϑ_2 coefficient controls to the social component ($\vartheta_2 * r_2 * (gBest_i - X_i)$ which measures the influence of the best position obtained by all particles for calculating the new velocity vector of a given particle. r_1 y r_2 are random values in the range [0..1].

The origins of PSO are based on a continuous codification, due to its nature for using operations on gradients. However, there are adjustments to the binary encoding and permutations of integers [1, 13, 14]. In [14] the authors present a first adaptation for a binary PSO. Where the position vector is encoded as a binary string; meanwhile the velocity vector is real. This new version of PSO uses the same functions to update the velocity vector than the basic algorithm (see Eq. 1). In order to determine the probabilities for the computation of the position vector, a sigmoid function is used that maps a velocity value onto the interval [0,1]. With this adaptation a lot of information is lost and the binary PSO performance is low in many cases. Clerc in [13] achieves a better performance to incorporate a new method for computing the position vector: *derivation 0*. Where the normalization is done using the modular arithmetic; thus less information is lost. We use the Clerc's implementation to develop this work. In the Algorithm 1 a PSO pseudo code with binary codification and derivation 0 is shown, henceforth we call this version PSOo.

3 Our proposal

In this paper, we propose a PSO adaptation, which we call hybrid Particle Swarm Optimization (PSOh). This version incorporates to the PSOo, two different technics: a *crossover operator* that helps to exploit promising regions of search space,

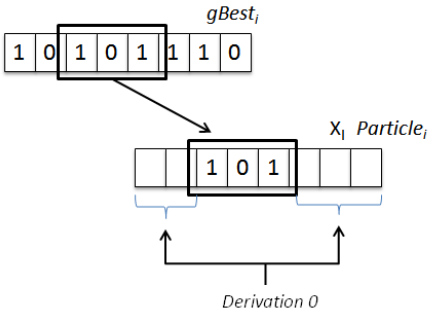

Fig. 1. Social Crossover Operator

and an *elitist restart operator* that allows to explore new regions of the search space. The idea behind this adaptation is to achieve a balance between exploration and exploitation of the search process since the PSOo shows a very disperse swarm at the end obtaining low quality results.

The crossover operator is one of the main operators of the Genetic Algorithms [3]. It combines genetic material from two or more individuals (called parents) to produce new individuals (called children). Particularly for PSOh, we propose a new crossover operator which is called Social Crossover (SX). SX copies a position vector fragment from the current global best solution ($gBest_i$) into the position vector of a given particle. Such fragment is copied in the same place from which is extracted and its size is random (see Fig. 1). For generating the rest of the particle, we use the *derivation 0* method. In this way significant information is transferred from the best particle during the search process. This allows: the exploitation of promising regions of the search space, an important social influence on a new particle and a faster convergence to the optimum.

While the restart procedure arises from CHC [11] where the population is restarted keeping the best individual when a population stagnation is detected. Note that, for CHC, this procedure replaced the mutation operator to provide a further exploration of the search space. Particularly for PSOh, we propose an elitist restart operator which, restarts a population proportion (20%) keeping the best particle. The idea behind this operator is to avoid: the particle swarm becomes homogeneous and convergence to be local optimum.

For each iteration, PSOh executes the following actions (see Algorithm 2):

1. Update the components of position and velocity vectors using *derivation 0* method except the components which are between $pos1$ and $pos2$ positions. These components are updated using the Social Crossover. In this way, a new particle is created. This procedure is applied under an 80% of probability.
2. Apply the Elitist Restart Operator with a 20% of probability.

Algorithm 2 *PSOh*

```
S ← initializeSwarm()
while not stop condition do
  for i = 0; i < size(S); i++ do
    if fitness(X_i) is better than fitness(pBest_i) then
      pBest_i ← X_i
    end if
    if fitness(pBest_i) is better than fitness(gBest_i) then
      gBest_i ← pBest_i
      fitness(gBest_i) ← fitness(pBest_i)
    end if
  end for
  for i = 0; i < size(S); i++ do
    if rand(0,1) < 0.8 then
      pos1 ← rand(0, size(X_i) − 1)
      pos2 ← rand(0, size(X_i) − 1)
      if pos1 > pos2 then
        toExchange(pos1, pos2)
      end if
      j ← 0
      while (j >= 0 && j <= pos1) || (j >= pos2 && j < size(X_i)) do
        if j == pos1 then
          j ← pos2
        end if
        //Derivation0
        V_i ← W * V_i + ϑ1 * r1 * (pBest_i − X_i) + ϑ2 * r2 * (gBest_i − X_i)
        X_i ← X_i + V_i
        X_i ← (4 + X_i)mod_2
        V_i ← (3 + V_i)mod_3 − 1
        j ← j + 1
      end while
      //CrossoverOperator
      for j = pos1;pos1 < pos2; j++ do
        X_i ← gBest_i
      end for
    else
      //ElitistRestartOperator
      if X_i ≠ gBest then
        for j = 0; j < size(S); j++ do
          X_i ← rand(0,1)
        end for
      end if
    end if
  end for
end while
```

4 Test Functions

In this paper, we use four different optimization functions to analyze the above mentioned metaheuristic; they are: Sphere function (*f1*), Rosenbrock function (*f2*), Rastrigin function (*f3*), Easom function (*f4*). Those functions have been chosen since they represent real problems and belong to a function set which is used to study many optimization methods. All these functions are mapped in $R^n \rightarrow R$. The characteristics of each function are shown in Table 1.

5 Computational Experiments

In this section, we show the results obtained by applying PSOo and PSOh on the set of functions presented in section 2. This set of functions presents differ-

Table 1. Function characteristics

Functions	
Sphere (*f1*) $F(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^2$ *Optimum Value* 0.0	*Dimension Size* $n = 5(f1_5)$ *Search Space* $S = \{x \mid \forall i : -5.12 =< x_i =< 5.12 \in \mathbf{R}\}$ *Main Characteristics* unimodal
Rosenbrock (*f2*) $F(X) = \sum_{i=1}^{n-1} 100 * (x_{i+1}^2 - x_i^2)^2 + (x_i - 1)^2$ *Optimum Value* 0.0	*Dimension Size* $n = 5(f2_5)$ *Search Space* $S = \{x \mid \forall i : -5.12 =< x_i =< 5.12 \in \mathbf{R}\}$ *Main Characteristics* dependent variables
Rastrigin (*f3*) $F(X) = (\sum_{i=1}^{n} -10 * cos(2\pi x_i)) + 10 * n$ *Optimum Value* 0.0	*Dimension Size* $n = 5(f3_5)$ *Search Space* $S = \{x \mid \forall i : -5.12 =< x_i =< 5.12 \in \mathbf{R}\}$ *Main Characteristics* multimodal
Easom (*f4*) $F(x, y) = -cos(x) * cos(y) * exp(-(x - \pi)^2) - (y - \pi)^2$ *Optimum Value* -1	*Dimension Size* $n = 2$ *Search Space* $S = \{-100 =< x, y =< 100 \in \mathbf{R}\}$ *Main Characteristics* deceptive

Table 2. PSOo and PSOh parameters

Independent runs	30
Swarm Size	64
Neighborhood size	64
Inertia Weight (W)	0.732
Knowledge and Social Coefficients (ϑ_1, ϑ_2)	2
Stop Condition	600 generations

ent complexity degrees due to their characteristics (unimodal, multimodal and deceptive) and dimensions (2D and 5D). These functions are: $f1_5$, $f2_5$, $f3_5$, $f4_2$. PSOo and PSOh algorithms were executed using the MALLBA software [15], which was created by research group from Malaga, La Laguna and Barcelona Universities. For each algorithmic setting we have performed 30 independent runs per function and we use an Intel Centrino duo processor with 1.73 Ghz and 1 GB of RAM.

As we said in previous sections, the position vector is a binary string where 24 bits encode a real value. Therefore, for a search space of 5 dimensions, the position vector of each particle has 120 bits and for 2 dimensions it has 56 bits.

Some parametric values (swarm size, neighborhood size, stop condition) have been optimized through hand-tuning comparing different values and others parameters (inertia weight, knowledge and social coefficients) are taken from the literature. In Table 2 are summarized those values.

We use the following relevant performance variables to analyze the behavior of each algorithm:

- *ABest (Average Best).* This value is the average of the best solutions found by each algorithmic version in each run.
- *Best (Best).* It is the best solution or particle that was found by each algorithmic version.
- *APop (Average Population).* This value is the average of the solutions in the final population.

Table 3. PSOo and PSOh Results

PSOo					
Functions	**Abest**	**Best**	**AbestI**	**Apop**	**AbestT**
$f1_5$	0.709 ± 0.23	0.116	280.067	42.906 ± 17.25	569.959
$f2_5$	48.572 ± 25.06	20.055	281.300	58687.08 ± 39657.66	551.941
$f3_5$	12.323 ± 2.32	7.809	337.733	92.862 ± 22.67	682.738
$f4_2$	-0.622 ± 0.25	-0.974	306.567	0 ± 0	135.459
PSOh					
$f1_5$	0.157 ± 0.08	0.039	304.80	43.274 ± 43	291.390
$f2_5$	22.471 ± 27.21	3.533	367.23	58941.776 ± 133857.79	293.817
$f3_5$	6.853 ± 4.43	1.905	285.23	93.149 ± 33.99	214.105
$f4_2$	-0.973 ± 0.04	-1.000	316.67	0 ± 0.01	64.613

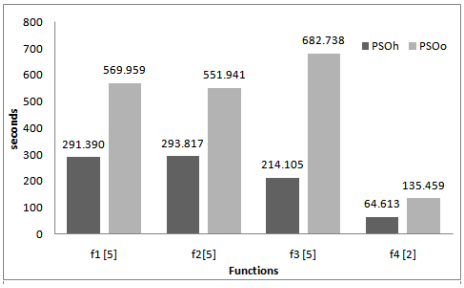

Fig. 2. PSOo and PSOh AbestT values

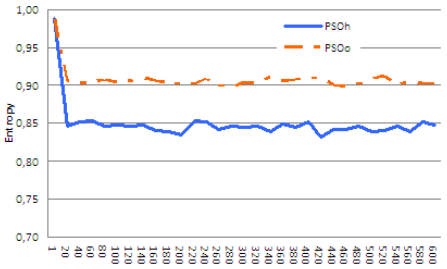

Fig. 3. PSOo and PSOh average entropy values every 20 iterations

Table 4. PSOo and PSOh average entropy values

	Average Entropy	
Función	**PSOo**	**PSOh**
$f1_5$	0.904	0.856
$f2_5$	0.900	0.850
$f3_5$	0.896	0.835
$f4_2$	0.909	0.851

- *ABestI (Average Best Iteration).* It is an average of the iterations which were necessary to find the Best value.
- *ABestT (Average Best Time).* It is the average time (in seconds) that the algorithm takes to find the best value.
- *Entropy.* The entropy measure is used to understand how the particles are different from each other. This measure is in the interval [0,1]. If the swarm entropy is close to 1, then the particles will be different. Otherwise if it is close to 0, particles will be similar, i.e., the genotypic diversity will be null or almost null.

In Table 3 the results obtained by PSOo and PSOh are detailed. In Figure 2, PSOo and PSOh performances are compared taking into account the four different search spaces that were used for testing. While in Table 4 the average entropy values corresponding to final populations are summarized.

From an analysis of Tables 3 and Figure 2, we can observe:

- In average PSOh improves the quality of best found solution in a 75% with respect to PSOo. This difference is corroborated using a $t - student$ Test with $\alpha = 0.05$, which is applied to compare the fitness quality obtained by both algorithms in each function; where a $p - value = 0.000$ is lesser than α.

Noteworthy PSOo only is closed to the optimum when it solves $f4_2$. While PSOh is closed to the optimum when it solves $f1_5$, $f2_5$ and $f3_4$ functions and it reaches the optimum for $f4_2$.

- In average our proposed algorithm reduces the execution time to find the best solution in a 54%. Although both algorithmic approaches need approximately 310 iterations to find their best solution. This difference is also corroborated using a $t - student$ Test where a $p - value = 0.0003$ is lesser than $\alpha = 0.05$.
- The final populations obtained by PSOo and PSOh algorithms have diverse solutions when these algorithms solve $f1_5$, $f2_5$ and $f3_5$ functions. In Evolutionary Computation this kind of diversity is called *phenotypic diversity*.

From Table 4, we can analyze if the particle composition in the final population is diverse or not. That is called *genotypic diversity* and it is measured for the entropy variable. This diversity is almost total (0.9 in average) when PSOo solves every function. This means that the particles in the final swarm are very dispersing and PSOo algorithm has many difficulties to converge. While PSOh obtains final swarms with lesser genotypic diversity (0.8) converging to better solutions faster. This improvement is achieved by applying social crossover and elitist restart operators. This crossover operator guides the search towards better solutions i.e., it helps the exploitation. While the elitist restart operator allows to explore the search space keeping the diversity in a swarm whose size is relatively small. In this way, PSOh presents a better balance between exploitation and exploration than PSOo.

In order to analyze more exhaustively the behavior of PSOo and PSOh, we have carried out a survey every 20 iterations from each execution. In this survey, the considered variables are Best and Entropy. Figures 3 and 4 summarize these data.

In Figure 4, we can see the best solution found by PSOo and PSOh every 20 iterations during the search process. The PSOo and PSOh behaviors if we consider the result quality. In this regard, for each search space tested both algorithms present a similar behavior. They start with high fitness values, these values drop sharply at the first 80 iterations approximately. From this point, the improvements continue but with a fewer intensity and an important difference between these two algorithms appears. PSOo decreases the best fitness value keeping the same best fitness over long intervals of iterations; while PSOh shows a slow but steady decline of the best fitness for the rest of the iterations. Another important difference is that during all iterations, PSOh always obtains better fitness values than PSOo.

In Figure 3, we can see that both algorithms start with a very high genotypic diversity, which fall quickly at the first 40 iterations. But for PSOh the drop is more significant than for PSOo (0.8 vs. 0.9). In average those values are maintained on the rest of the run by each algorithm.

Once more, we can see the relation between the genotypic diversity achieved by each algorithm with the quality of the results obtained by each of them. In

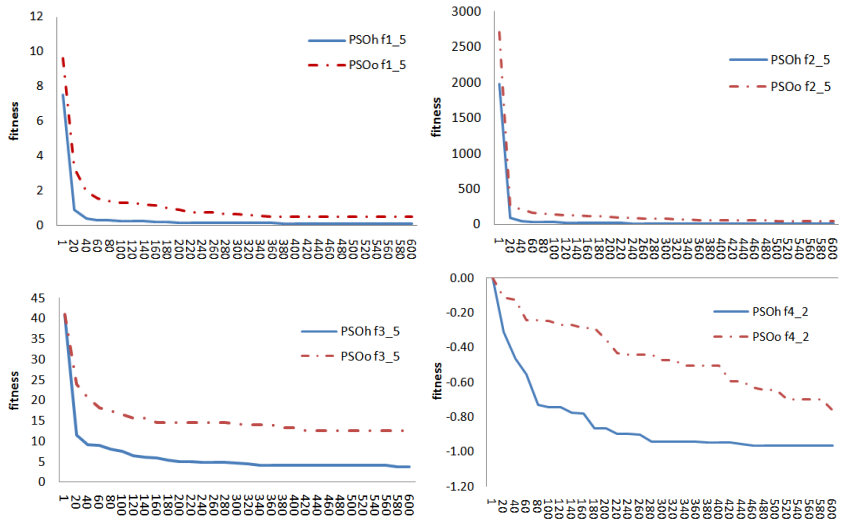

Fig. 4. PSOo and PSOh Abest values taken every 20 iterations for each tested function.

other words, PSOh reduces the swarm dispersion without losing the necessary exploration to avoid local optimum.

6 Conclusions

Two different algorithmic versions for Particle swarm optimization have been presented, studied and analyzed in this job. One of this version is a classical binary PSO which use the *derivation 0* method to compute the values of the position vector. The other version is proposed for us, it is a *hybrid* binary PSO which use the *derivation 0* method.

This hybridization arises because the first algorithm (the classical binary PSO) cannot converge to the optimum since the high degree of dispersion in the swarm during the search process. In order for solving this problem a social crossover and an elitist restart operators are incorporated into the binary PSO. In this way, we can exploit the search space using this crossover operator and explore it using the elitist restart operator. The idea behind this hybridization is to balance the exploitation and exploration in the search process. That means, the particles should be sufficiently separated from one another without being too dispersed.

Both versions of the Particle Swarm Optimization were tested for a set of selected functions which have different characteristics and complexity degrees. At the light of these results we can conclude that:

– Regarding the quality of results (best and average individuals) and the time to find near optimal solutions the hybrid approach outperforms the classical binary PSO.

– Concerning the genotypic diversity, the hybrid approach losses a minimum of diversity on the search process to achieve a better balance between exploration and exploitation.

Due to these promissory results, this research will be continued solving complex combinatorial optimization problems. For example, problems in the area of antenna designs will be studied. Furthermore, an analysis about parameters will be performed.

References

[1] Kennedy, J., Eberhart, R., Shi, Y.: Swarm intelligence. (2001)
[2] Shi, Y.: Particle swarm optimization. IEEE Neural Networks Society ,Electronic Data Systems, Kokomo, IN 46902, USA (2004)
[3] Holland, J.H.: Adaptation in Natural and Artificial Systems. First edn. The MIT Press, Cambridge, Massachusetts (1975)
[4] Kennedy, J.: The particle swarm: Social adaptation of knowledge. International Conference on Evolutionary Computation IEEE (1997) 303 – 308
[5] Liang, J.J., Suganthan, P.N.: Dynamic multiswarm particle swarm optimizer (dms-pso). Proceedings of the IEEE swarm intelligence symposium (SIS).Piscataway: IEE (2005) 124129
[6] Clerc, M.: Stagnation analysis in particle swarm optimization or what happens when nothing happens. Technical Report CSM-460, Department of Computer Science, University of Essex (august 2006)
[7] K.E. PARSOPOULOS, M.V.: Initializing the particle swarm optimizer using the nonlinear simplex method. Technical report, Department of Mathematics University of Patras (2002)
[8] Ai, T.J., Kachitvichyanukul, V.: Particle swarm optimization and two solution representations for solving the capacitated vehicle routing problem. Computers & Operations Research **36**(5) (May 2009) 1693–1702
[9] Moore, J., Chapman, R.: Application of particle swarm to multiobjective optimization. Technical report, Department of Computer Science and Software Engineering, Auburn University (1999)
[10] Millie, P., Radha, T., Ajith, A.: Particle swarm based meta-heuristics for function optimization and engineering applications. 7th Computer Information Systems and Industrial Management Applications (2008)
[11] Eshelman, L.J.: The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In: Foundations of Genetic Algorithms, Morgan Kaufmann (1991) 265– 283
[12] Eberhart, R., Shi, Y.: Comparing inertia weights and constriction factors in particle swarm optimization. In: International Congress on Evolutionary Computation. Volume 1. (2000) 84–88
[13] Clerc, M.: Binary particle swarm optimisers: Toolbox, derivations, and mathematical insights. (febrero 2005)
[14] Kennedy, J., Eberhart, R.C.: A discrete bynary version of the particle swarm algorithm. In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics **5**(0-78034053-1) (1997) 4104 – 4109
[15] Alba, E.: Mallba: a library of skeletons for combinatorial optimisation. In Eighth International Euro-Par Conference (2002) 927–932