

Particle Swarm Optimization in Non-Stationary Environments

Susana C. Esquivel¹ and Carlos A. Coello Coello²

¹Laboratorio de Investigación y Desarrollo en Inteligencia Computacional
Universidad Nacional de San Luis, Ejército de los Andes 950
(5700) San Luis, ARGENTINA
esquivel@unsl.edu.ar

²CINVESTAV-IPN (Evolutionary Computation Group)
Electrical Engineering Department, Computer Science Section
Av. IPN No. 2508, Col. San Pedro Zacatenco
México D.F. 07300, MÉXICO
ccarloco@cs.cinvestav.mx

Abstract. In this paper, we study the use of particle swarm optimization (PSO) for a class of non-stationary environments. The dynamic problems studied in this work are restricted to one of the possible types of changes that can be produced over the fitness landscape. We propose a hybrid PSO approach (called HPSO-dyn), which uses a dynamic macromutation operator whose aim is to maintain diversity. In order to validate our proposed approach, we adopted the test case generator proposed by Morrison & De Jong [1], which allows the creation of different types of dynamic environments with a varying degree of complexity. The main goal of this research was to determine the advantages and disadvantages of using PSO in non-stationary environments. As part of our study, we were interested in analyzing the ability of PSO for tracking an optimum that changes its location over time, as well as the behavior of the algorithm in the presence of high dimensionality and multimodality.

1 Introduction

Many real-world optimization problems are non-stationary. Such problems arise when either the resources (constraints) or the optimization criteria change (or both), and they are common in engineering, production planning and economics [2]. In the last decade, heuristics that can adapt to changes have attracted a lot of interest, particularly those that operate on a population of solutions. From these heuristics, most of the research has concentrated on evolutionary algorithms [3, 4], and several promising results have been obtained within the last few years. Another population-based technique that has recently been adopted for dealing with non-stationary environments is particle swarm optimization [5, 6], which is precisely the approach adopted in the work reported in this paper. An algorithm capable of dealing with non-stationary environments normally considers two main aspects [4]: the detection of the change and the reaction of the algorithm to such change, such that the (moving) optimum can be tracked as closely as possible. There is some previous research on the use of particle swarm optimization

for dynamic optimization that involves these two aspects previously discussed. Carlisle & Dozier [7] used a *sentry particle* that is evaluated at each iteration, comparing the previous fitness value with the new one (if they differ, this means that the environment has changed). Hu & Eberhart [8] proposed the re-evaluation of two g_{best} particles in order to detect the movement of the location of the optimum. Several authors have also proposed approaches in which the main emphasis is the reaction of the algorithm to changes in the environment (see for example [9–11]). In this paper, we will focus on this second aspect. Despite the encouraging results reported in the papers previously indicated, all of these authors focus their work on unimodal fitness landscapes [10, 11]. This is rather unrealistic if we consider the complexity of real-world problems that frequently present high multimodality. This is precisely the issue that we address in this paper. Aiming to provide a framework for performing further comparative studies, we decided to adopt the test case generator originally developed by Morrison & De Jong [1], which shares several similarities with the proposal of Branke [4]. This test case generators allows us to define non-stationary environments of different degrees of complexity both regarding the morphology of the fitness landscapes as well as regarding the type of changes that can be produced and their severity.

2 Classical PSO Model

The PSO model (now considered “classical”) was originally proposed by James Kennedy and Russell Eberhart in 1995 [12], and has had several further extensions in the next few years [5]. The PSO model is inspired on the acting of communities that present both an individual and a social behavior, from which the main socio-cognitive ideas may be applied to the development of efficient algorithms that solve optimization problems (this is the so-called “swarm intelligence” [5]). Although PSO is considered by its authors as an evolutionary algorithm, it does not incorporate the traditional genetic operators (crossover and mutation) that evolutionary algorithms normally adopt. Instead, in PSO each particle adjusts its flight over the search space based on its own flight experience and that of its neighbors (i.e., the other particles of its swarm). Each particle represents a possible solution to the problem at hand and is treated as a point in a d -dimensional search space. A particle is characterized by its position and its current velocity. Furthermore, a particle has a memory where it stores the best position that it has found so far. Particles adjust their flight trajectories using the following equations [13]:

$$v_{i,j} = w \times v_{i,j} + c_1 \times r_1 \times (p_{i,j} - x_{i,j}) + c_2 \times r_2 \times (p_{g,j} - x_{i,j}) \quad (1)$$

$$x_{i,j} = x_{i,j} + v_{i,j} \quad (2)$$

where w is the inertia factor, whose goal is to control the impact of the history of the velocities of a particle over the current velocity, by influencing the local and global exploration abilities of the algorithm. $v_{i,j}$ is the velocity of the particle i in the j -th dimension, c_1 and c_2 are weights applied to the influence of the best position found so far by particle i and by the best particle in the swarm g . r_1 and r_2 are random values (with a uniform distribution) within the interval $[0, 1]$. After the velocity is updated, the new position of the particle i in its j th dimension is recomputed. This process is

repeated for each dimension of the particle i and for all the particles in the swarm. Kennedy and Eberhart [14] proposed different neighborhood techniques to be considered when updating the velocity of a particle. In this paper, we adopted the most simple neighborhood scheme available, which only includes the closest neighbors in terms of the indices of the particles. Under this scheme, the equation to update the velocity of a particle becomes:

$$v_{i,j} = w \times v_{i,j} + c_1 \times r_1 \times (p_{i,j} - x_{i,j}) + c_2 \times r_2 \times (p_{l,j} - x_{i,j}) \quad (3)$$

where $p_{l,j}$ represents to the best particle in the neighborhood. The formula used to update a particle remains the same as before.

3 Hybrid PSO Model

Recently, there have been several proposals of PSO approaches hybridized with evolutionary algorithms' concepts and operators [15, 16]. The proposal presented in this paper has as its basis the equations to update the particles and velocities defined by equations (2) and (3). Additionally, we use a dynamic macromutation operator that is described next.

Mutation: Each coordinate of the particle is independently mutated with a probability p_{mut} . The coordinate value is replaced by another value which is randomly generated within the allowable range. The mutation probability is dynamically adjusted taking values within the interval $[p_{min}, p_{max}]$ during the execution of the algorithm. Thus, p_{mut} is defined as follows:

$$p_{mut} = \frac{(p_{max} - p_{min}) \times (\text{interval} - \text{iter}_{current} \bmod \text{interval})}{\text{interval}} + p_{min} \quad (4)$$

where the values p_{min} and p_{max} are the lower and upper bounds of the variation interval of the probability, interval indicates the number of iterations between changes and $\text{iter}_{current}$ corresponds to the current flight cycle of the particles. Each time a change takes place in the environment, the value of p_{mut} is initialized to p_{max} and the end of the interval of p_{mut} is set to p_{min} .

Once the swarm and the velocities are initialized (lines 1–2), the swarm is evaluated with the base function F_0 . Then, the memory of the particles ($Swarm_bests$) is initialized (line 4). After that, the algorithm enters the flight loop (line 6) which is executed during a number of cycles that is determined in terms of the number of changes to be performed and in terms of the interval between such changes. Then, on line 7, we compute the value for p_{mut} (according to equation (4)) and the function `occurred_changes` determines if a change is required within the current flight cycle. In order to do this, we check if the current cycle number is a multiple of the number of cycles between changes. The changes in the environment are produced at constant intervals. If the environment must change, the dynamical statistics are reported, the function is modified and both the $Swarm$ and the $Swarm_bests$ are re-evaluated with the new fitness function (lines 9 to 12). These re-evaluations are necessary since the current fitnesses of the particles do not correspond to the new function. Once the two swarms

```

1. Initialize(Swarm)
2. Initialize(velocities)
3. Evaluate(Swarm,  $F_0$ )
4. Copy(Swarm, Swarm-bests)
5.  $t = 0$ 
6. do
7.   Calculate( $p_{mut}$ )
8.   if (occurred_change)
9.     Report_dynamic_statistics()
10.    Change_function()
11.    Evaluate(Swarm,  $F_t$ )
12.    Evaluate(Swarm-bests,  $F_t$ )
13.    Update(Swarm-bests) if appropriate
14.    Calculate( $p_{mut}$ )
15.   end if
16.   Mutate(Swarm)
17.   Update(velocities)
18.   Update(Swarm)
19.   Evaluate(Swarm,  $F_t$ )
20.   Update(Swarm-bests) if appropriate
21.    $t = t + 1$ 
22. while ( $\neg$ termination)

```

Fig. 1. General outline of the HPSO-dyn Algorithm

have been re-evaluated, we verify, for each particle, that *Swarm-best*s really contains the best particle positions for the new environment, as to maintain the consistency of the algorithm. By doing this, we do not lose the memory of all the particles, but only the memory of those for which their current positions (in *Swarm*) are the best [11]. We recompute again the transition function for the probability of mutation that will take us from p_{mut} to p_{max} . Next, we proceed with the normal PSO processing, which implies: mutate the particles, update their velocities, update the positions of the particles, evaluate the particles and, if applicable, modify their position in *Swarm-best*s (lines 16–20). This process is done asynchronously, since there is evidence of the efficiency of this type of processing when working with neighborhoods in PSO [16, 6].

4 DF1 Generator

In this section we briefly describe the Test Function Generator proposed by Morrison & De Jong [1]. This generator uses a morphology that the authors denominate “field of cones”. Such cones can have different heights and slopes and are randomly distributed along the landscape. The static base function (F_0), for the two-dimensional case, is defined as:

$$F_0(x, y) = \max_{i=1,n} [H_i - R_i \times \sqrt{(x - x_i)^2 + (y - y_i)^2}] \quad (5)$$

where n indicates the number of cones in the landscape and each cone is independently specified by its coordinates (x_i, y_i) that belong to the interval $[-1, 1]$, its height (H_i) and its slope (R_i). The cones independently defined are grouped using the function \max . Each time the generator is invoked, it randomly generates a morphology with the characteristics given. The function F can be defined for any number of dimensions. The user can create a wide variety of shapes for the fitness landscape by specifying the range of allowable values for the height, slope and location of the cones. Furthermore, the generator allows to re-write the values randomly generated in order to create landscapes with controlled characteristics. The severity of the changes is controlled through the *logistic function*:

$$Y_i = A \times Y_{i-1} \times (1 - Y_{i-1}) \quad (6)$$

where A is a constant defined within the interval $[1, 4]$ and Y_i is the value at the iteration i . Thus, the procedure to obtain the dynamic that the user wants is the following: 1) choose the number of cones and 2) determine what characteristics the user wishes to change (height, location, slope of one or all the cones). The severity of the change is controlled by the values that are assigned to the constant A , since from this depends that the movement is produced either at small, large or chaotic steps. For further details on this generator, the reader should refer to [1].

5 Experimental Design

The goal of this research was twofold: first, we wanted to determine if the proposed algorithm could track down the optimum once a change has occurred. Second, we wanted to analyze the algorithm's behavior upon scaling both the number of dimensions and the number of cones. Thus, we considered that the scenarios proposed by Morrison [17] provided the required conditions to perform our study. Such scenarios are described next: 1) by the shapes of the fitness landscape and 2) by the dynamics applied.

Description of the structure of the static landscape

1. E1: 2 dimensions, 5 cones (2d-5c).
2. E2: 2 dimensions, 14 cones (2d-14c).
3. E3: 5 dimensions, 5 cones (5d-5c).
4. E4: 10 dimensions, 14 cones (10d-14c).

In all the scenarios, we assigned the maximum value to a single peak. Such a value is greater than that of the other peaks (to make sure that there is a single global optimum).

Dynamics Applied

The type of change implemented consisted of modifying only the location of the cone that contains the optimum (CO) or changing the location of all the cones simultaneously (TC). In Table 1, we describe, for each problem studied, the type of change and its severity.

Table 1. Dynamic Behavior Applied

Scenarios	Dimension	Cones	Change_type	Step_size
E1	2	5	CO	small(s)
E2	2	14	TC	chaotic(c)
E3	5	5	TC	small
E3	5	5	TC	large(l)
E4	10	14	TC	chaotic

For the scenarios $E1$ to $E3$, the changes take place at every 10, 20, 30, 40 and 50 iterations. For scenario $E4$, given its complexity, changes take place at every 30, 60 and 90 iterations. Furthermore, at each run, 20 changes took place either on the location of the cone that contains the optimum value or in all the cones.

Parameters of the DF1 Generator

Table 2. DF1 Parameters

Par	2d-5c-CO-s	2d-14c-TC-c	5d-5c-TC-s	5d-5c-TC-l	10d-14c-TC-c
Hbase	60.0	1.0	60.0	60.0	1.0
Hrange	0.0	9.0	0.0	0.0	9.0
Rbase	70.0	8.0	70.0	70.0	8.0
Rrange	0.0	12.0	0.0	0.0	12.0
Ac	1.5	3.8	1.5	1.5	3.8
Scale	0.3	0.5	0.3	0.99	0.5
OptH	90.0	15.0	90.0	90.0	15.0
OptR	90.0	20.0	90.0	90.0	20.0

The parameters of DF1 that correspond to the dynamics implemented are defined in Table 2, according to the proposal by [17], where $Hbase$, $Hrange$, $Rbase$ and $Rrange$ correspond to the ranges of the heights and slopes of the cones that do not contain the global optimum. $OptH$ and $OptR$ correspond to the cone containing the global optimum.

Parameters of the HPSO-dyn algorithm

The parameters required by the algorithm are provided in Table 3 and were selected after a large series of experiments.

In all the experiments, we worked with a neighborhood radius of size 4. For each experiment, we performed 30 runs, all of them with the same base function and the same initial population.

Table 3. PSO Algorithm Parameter Settings

Scenarios	w	$c1$	$c2$	p_{min}	p_{max}	$swarmsize$
E1	0.5	1.5	1.5	0.1	0.4	50
E2	0.5	1.5	1.5	0.5	0.8	50
E3	0.5	1.5	1.5	0.3	0.6	200
E4	0.5	1.5	1.5	0.5	0.8	500

Table 4. Two-dimensional functions

Int_{chgs}	2d-5c-CO-s	2d-14c-TC-s
10	600 (100%)	600 (100%)
\bar{f}	89.695084	14.848885
s	0.042112	0.027802
20	600 (100%)	600 (100%)
\bar{f}	89.966666	14.983450
s	0.274450	0.003067
30	600 (100%)	600 (100%)
\bar{f}	89.997041	14.998491
s	0.02776	0.002023
40	600 (100%)	600 (100%)
\bar{f}	89.999650	14.999882
s	0.000252	0.000354
50	600 (100%)	600 (100%)
\bar{f}	89.999973	14.999358
s	0.000032	0.000177

6 Analysis of Results

In Table 4 we show the results obtained for the 2D (i.e., two-dimensional) functions with 5 and 14 cones. Here, \bar{f} refers to the mean of the best average fitness values found in the 30 runs and s is the variance. The optimum value for all the functions with 5 cones is 90.00 and all the other cones have a maximum value of 60.00. For the functions with 14 cones, the maximum value is 15.00 and all the other cones have a maximum of 10.00. The performance of our HPSO-dyn for the two-dimensional functions with 5 and 14 cones is very satisfactory, since in all cases, for all the changes performed, the algorithm was able to track down the global optimum with an acceptable error.

Table 5 shows the results obtained by our algorithm for the 5D functions with 5 cones, in the case where all the cones change their location in the landscape, with step sizes which are, in one case small, and in the other one, large. In this case, our algorithm is able to track down the optimum upon performing all the changes. However, note that the error becomes higher when the changes are produced at every 10 iterations. Finally, Table 6 presents the results obtained for the most complex environment studied in this paper: a 10D function with 14 cones, with changes produced with a chaotic step size. As can be observed, the performance of the algorithm was degraded since

Table 5. Functions of 5 dimensions with 5 cones

Int_{chgs}	5d-5c-TC-l	5d-5c-TC-s
10	600 (100%)	600 (100%)
\bar{f}	79.618352	78.821386
s	0.913599	0.934210
20	600 (100%)	600 (100%)
\bar{f}	86.373647	85.169944
s	0.997890	1.003454
30	600 (100%)	600 (100%)
\bar{f}	88.554862	88.960822
s	1.137792	1.20767
40	600 (100%)	600 (100%)
\bar{f}	89.606563	89.673193
s	0.061213	0.077778
50	600 (100%)	600 (100%)
\bar{f}	89.865952	89.849669
s	0.029567	0.013788

Table 6. 10d-14c Function with chaotic changes for all cones

Int_{chgs}	10d-14c-TC-c
30	527 (87,8%)
\bar{f}	11.813951
s	0.181945
60	550 (91,6%)
\bar{f}	13.74900
s	0.111303
90	560 (93,3%)
\bar{f}	13.983545
s	0.201526

we could not succeed 100% of the time at tracking down the optimum in any of our experiments. Nevertheless, the success rate oscillates between 87% and 93%, which are values reasonably good. It is also worth noticing that the difference between the fitness values found by the algorithm with respect to the global optimum increased as well. This is because we included in the average fitness values the unsuccessful cases. Table 7 provides the success rate of our algorithm for the case of 10D functions with 14 cones. This table indicates (as a percentage) the number of runs in which the algorithm was able to successfully track down the global optimum (inputs in the tables). Note that 20 changes took place per run. As we can see in the table even in the most unfavorable case, when the interval between changes is small (30 iterations) the algorithm is able to adapt between the 85% and 90% of the changes. As we expected, these percentages variate between 90% and 95% when the range of the interval is increased.

Table 7. Success Rates for the Function 10d-14c

%	30	60	90
100	-	-	-
95	4	10	20
90	13	20	10
85	10	-	-
80	3	-	-

7 Conclusions and Future Work

We have presented an optimization algorithm for non-stationary environments. Our algorithm is based on a particle swarm optimizer which was hybridized with a dynamic macromutation operator. Although one could think at first sight that the mutation probabilities adopted in our work are too high, this is done to maintain the required diversity in the swarm and does not disrupt the search process, because of the use of the memory *Swarm-best*, which is not destroyed when the changes take place and is only updated when a particle from the *Swarm* obtains a better fitness than the one stored in such memory. In our experiments, we adopted functions with 2, 5 and 10 dimensions and with 5 and 14 cones. All of these functions were created with the DF1 generator, which allowed us to study the behavior of the algorithm with more complex fitness landscape structures than those normally adopted in the specialized literature (i.e., the sphere model). The changes produced consisted either on changing the location of the cone that contained the global optimum or on changing the location of all the cones, with step sizes that were from small to chaotic. The results obtained show that the performance of the algorithm over morphologies with several suboptima, is highly satisfactory when using either 2 or 5 dimensions and a number of cones between 5 and 14. However, the performance degrades as we move to problems with 10 dimensions. Nevertheless, the success rate of the algorithm in these cases remains reasonably good, since it only fails to track down an average of 2 (out of 20) changes performed. Although the scenarios presented in this paper were non-trivial, our future work addresses two aspects: 1) to extend these scenarios to include other types of landscape morphologies and 2) to study the other change types provided by the DF1 generator.

Acknowledgments

The first author acknowledges the continuous support received from the Universidad Nacional de San Luis and the ANPCYT. The second author acknowledges support from CONACyT project number 42435-Y.

References

1. Morrison, R.W., De Jong, K.A.: A Test Problem Generator for Non-Stationary Environments. In: Conference on Evolutionary Computation (CEC'99). Volume 3., Piscataway, NJ., IEEE Service Center (1999) 2047–2053

2. Michalewicz, Z., Fogel, D.B.: How to Solve it: Modern Heuristics. Springer, Berlin (2000)
3. Angeline, P.J.: Tracking Extrema in Dynamic Environments. In et al., P.J.A., ed.: Evolutionary Programming VI, 6th International Conference EP'97, Springer-Verlag. Lecture Notes in Computer Science No. 1213 (1997) 335–345
4. Branke, J.: Evolutionary Optimization in Dynamic Environments. Kluwer Academic Publishers, Boston/Dordrecht/London (2002)
5. Kennedy, J., Eberhart, R.C.: Swarm Intelligence. Morgan Kaufmann Publishers, California, USA (2001)
6. Carlisle, A.: Applying The Particle Swarm Optimization to Non-Stationary Environments. PhD thesis, Auburn University, USA (2002)
7. Carlisle, A., Dozier, G.: Adapting Particle Swarm Optimization to Dynamic Environments. In: International Conference on Artificial Intelligence, Las Vegas, Nevada (2000) 429–434
8. Hu, X., Eberhart, R.C.: Adaptive particle swarm optimization: Detection and response to dynamic systems. In: International Conference on Evolutionary Algorithms, Piscataway, NJ., IEEE Press (2002)
9. Hu, X., Eberhart, R.: Tracking Dynamic Systems with PSO: Where's the Cheese? In: Workshop on Particle Swarm Optimization, Purdue school of engineering and technology (2001)
10. Blackwell, T.: Swarms in Dynamic Environments. In et al., E.C.P., ed.: Genetic and Evolutionary Computation—GECCO 2003. Proceedings, Part I, Springer. Lecture Notes in Computer Science Vol. 2723 (2003) 1–12
11. Carlisle, A., Dozier, G.: Tracking Changing Extrema with Adaptive Particle Swarm Optimizer. In: World Automation Congress, Orlando, USA (2002)
12. Kennedy, J., Eberhart, R.C.: Particle Swarm Optimization. In: Proceedings of the 1995 IEEE International Conference on Neural Networks, Piscataway, NJ., IEEE Service Center (1995) 1942–1948
13. Shi, Y., Eberhart, R.: A Modified Particle Swarm Optimizer. In: Proceedings of the 1998 IEEE International Conference on Evolutionary Computation, Piscataway, NY., IEEE Service Center (1998) 69–73
14. Kennedy, J., Mendes, R.: Population Structure and Particle Swarm Performance. In: Congress on Evolutionary Computation (CEC'2002). Volume 2., Piscataway, New Jersey, IEEE Service Center (2002) 1671–1676
15. Angeline, P.J.: Using selection to improve particle swarm optimization. In: Proceedings of the 1998 IEEE International Conference on Evolutionary Computation, Piscataway, NJ., IEEE Service Center (1998) 84–89
16. Esquivel, S., Coello Coello, C.A.: On the Use of Particle Swarm Optimization with Multi-modal Functions. In: 2003 Congress on Evolutionary Computation (CEC'2003). Volume 2., IEEE Press (2003) 1130–1136
17. Morrison, R.W.: Designing Evolutionary Algorithms for Dynamic Environments. PhD thesis, George Mason University, Fairfax, Virginia, USA (2002)