Chapter 8
A Simple Distributed Particle Swarm Optimization for Dynamic and Noisy Environments

Xiaohui Cui, Jesse St. Charles, and Thomas E. Potok

Abstract. In this paper, we present a Simple Distributed Particle Swarm Optimization (SDPSO) algorithm that can be used to track the optimal solution in a dynamic and noisy environment. The classic PSO algorithm lacks the ability to track changing optimum in a dynamic environment. Several approaches have been investigated to enhance the PSO algorithm's ability in dynamic environments. However, in dealing with dynamic environments, these approaches have lost PSO's original strengths of decentralized control and ease of implementation. The SDPSO algorithm proposed in this paper maintains these classic PSO features as well as provides the optimum result tracking capability in dynamic environments. In this research, the DF1 multimodal dynamic environment generator proposed by Morrison and De Jong is used to evaluate the classic PSO, SDPSO and other two adaptive PSOs.

8.1 Introduction

In the real world, we have to frequently deal with searching and tracking an optimal solution in a dynamic and noisy environment. This demands that the algorithm

Xiaohui Cui
Computational Sciences and Engineering Division,
Oak Ridge National Laboratory, Oak Ridge, TN 37831-6085
e-mail: cuix@ornl.gov

Jesse St. Charles
Department of Computer Science and Engineering,
University of Tennessee Chattanooga, TN 37403
e-mail: jesse-stcharles@utc.edu

Thomas E. Potok
Computational Sciences and Engineering Division,
Oak Ridge National Laboratory, Oak Ridge, TN 37831-6085
e-mail: potokte@ornl.gov

N. Krasnogor et al. (Eds.): Nature Inspired Cooperative Strate. for Optimization, SCI 236, pp. 89–102.
springerlink.com

not only find the optimal solution but also track the trajectory of the non-stationary optimal solution. Particle Swarm Optimization (PSO) [1] has been proven to be both effective and efficient in solving a diverse set of optimization problems [2, 3]. In the past several years, PSO has been successfully applied in many research and application areas. It has been demonstrated that PSO outperforms other optimization methods in static environments [4]. However, despite the successful application of Particle Swarm Optimization (PSO) techniques to many complex optimization problems, classic PSO solutions are often fragile and easily fail when the problem changes. The classic PSO algorithm lacks the ability to track non-stationary optimal solutions and does not have a mechanism to detect dynamic environment change [5].

There is a recent growing interest in designing PSO algorithms for dynamic environments[5, 6, 7, 8, 9, 10]. To deal with dynamic environments, these algorithms normally consider two main aspects [6]: (1) detecting the environment change and (2) reacting to the change so that the optimum can be tracked as closely as possible. Several approaches [7, 8] have been investigated to enhance the PSO algorithm's ability in dynamic environments. However, most approaches depend on adding sophisticated mechanisms for detecting environment change and/or maintaining the particle diversity for dealing with these changes. These approaches achieve the dynamic environment optimum result tracking capability by forfeiting PSO's original strengths of decentralized control and ease of implementation. The major reasons that PSO is attractive are that the algorithm is easy to implement, non-centralized, and require few parameters to adjust. In this paper, we propose a simple distributed PSO (SDPSO) for searching and tracking the non-stationary optimum solution in a dynamical environment. The SDPSO algorithm proposed in this paper retains the positive features of classic PSO while providing the optimum-solution tracking capability for dynamic environments.

The remainder of this paper is organized as follows: in section 8.2, a brief overview of classic PSO and a discussion of the shortcomings of PSO in a dynamic environment are presented. Various modified PSO approaches for dynamic environment are introduced in section 8.3. In section 8.4, the SDPSO approach is described in detail. In section 5, the DF1 multi-modal dynamic environment generator proposed by Morrison and De Jong is described and used as the test dynamic environment to evaluate the performance of the algorithms. Experiment setup and implementation in comparisons the performance of SDPSO and other modified PSO algorithms in the dynamical environment are presented in section 6. Experiment results are presented in section 8.7. Discussion and Conclusion are in section 8.8.

8.2 Classic Particle Swarm Models

PSO was originally developed by Eberhart and Kennedy in 1995 [1], inspired by the social behavior of bird flocks. In the PSO algorithm, birds in a flock are symbolically represented as particles. These particles can be considered as simple agents "flying" through a problem space. A problem space in PSO may have as many dimensions as are needed to model the problem space. A particle's location in the

multi-dimensional problem space represents one solution for the problem. When a particle moves to a new location, a different problem solution is generated. This solution is evaluated by a fitness function that provides a quantitative value of the solution's utility. The velocity and direction of each particle moving along each dimension of the problem space are altered at each generation of movement. It is the particle's personal experience combined with its neighbors' experience that influences the movement of each particle through a problem space. For every generation, the particle's new location is computed by adding the particle's current velocity $V-vector$ to its location $X-vector$. Mathematically, given a multi-dimensional problem space, the *ith* particle changes its velocity $v_{id}(t)$ and location $x_{id}(t)$ according to the following equations [4, 11]:

$$v_{id}(t+1) = w \times (v_{id}(t) + c_1 \times rand_1 \times (p_{id}(t) - x_{id}(t)) + c_2 \times rand_2 \times (p_{gd}(t) - x_{id}(t))) \tag{8.1}$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \tag{8.2}$$

where, $v_{id}(t)$ and $v_{id}(t+1)$ represent the *ith* particle's velocity in dimension d at time t and $t+1$; $x_{id}(t)$ and $x_{id}(t+1)$ represent the *ith* particle's location in dimension d at time t and $t+1$; p_{id} is the location where the best fitness value that *ith* particle has ever found since time t; p_{gd} is the location where the the highest fitness value that the whole particle population have ever found since time t; c_1 and c_2 are two positive acceleration constants; d is the number of dimensions of the problem space; $rand_1$, $rand_2$ are random values in the range of (0,1). w is called the constriction coefficient [4]. Equation 8.1 requires each particle to record its current position x_{id}, its velocity v_{id}, its personal best fitness value location vector p_{id} and the whole population's best fitness value location vector p_{gd}. The best fitness values are updated at each generation based on equation 8.3, where the symbol f denotes the fitness function; $P_i(t)$ denotes the best fitness coordination; and t denotes the generation.

$$P_i(t+1) = \begin{cases} P_i(t); & \text{if } f(X_i(t+1)) \le f(P_i(t)) \\ X_i(t+1); & \text{if } f(X_i(t+1)) > f(P_i(t)) \end{cases} \tag{8.3}$$

The P_{id} and P_{gd} and their fitness values $f(P_{id})$ and $f(P_{gd})$ can be considered as each individual particle's experience or knowledge and equation 8.3 is the particle's knowledge updating mechanism. In PSO, particles' knowledge will not be updated until the particle encounters a new vector location with a higher fitness value than the value currently stored in its memory. However, in a dynamic environment, the fitness value of each point in the problem space may change over time. The location vector with the highest fitness value ever found by a specific particle may not have the highest fitness value after several generations. It requires the particle to renew its memory whenever the real environment status does not match the particle's memorized knowledge. However, the traditional PSO lacks an updating mechanism to monitor the change of the environment and update each particle's memory when a change is detected. As a result, the particle continually uses the outdated experience/knowledge to direct its search, which inhibits the particle from following the

moving path of the current optimal solution and causes it to become trapped in the region of the former optimal solution.

8.3 Related Work in PSO for Dynamic Environment

To stop particles using the outdated knowledge in a dynamic environment, Carlisle [5] and Eberhart [7] proposed to periodically reset all particle memory and replace each particle's best fit value and location vector with its current location vector and fitness value to force the particle to "forget" its former experience. One major disadvantage of this reset mechanism is the difficulty of determining the reset frequency. Without prior-knowledge about the environment changing frequency, the particle's memory reset frequency needs to be set to a high value to capture the changing step of the environment.

However, a high resetting frequency reduces the efficiency of the convergence of the PSO. The essence of the PSO algorithm lies in each particle's learning from both its past search experience and its neighbor's past search experience and utilizing this knowledge to guide its next moving velocity. Periodic resetting will cause all particles to lose their knowledge gathered during the search and force them to restart learning. This decreases the search efficiency of the swarm. Frequently resetting the personal best vector may cause particles unable to quickly converge on the vicinity of the optimal solution, especially during the initial period of searching. Following each reset, the optimization algorithm needs extra time to re-evaluate each particle's current fitness value. In [12], Hu and Eberhart present a knowledge retaining PSO algorithm by only re-initializing a portion of the swarm when environment changes are detected. By resetting only a portion of the swarm, the old swarm particles can maintain memory of the previous environment.

To detect environment change, Carlisle [8] introduced a new notion, "sentry", in his APSO algorithm. The "sentry" is one or many special designed particles that are deployed in the problem space to monitor the environment changes. Different from other particles, the "sentry" does not move. Every time step, the "sentry" compares its current fitness value with the value in previous time step. If the value changed, it indicates the environment changed. When the "sentry" detects a change in the environment, it informs all others and forces other particles to recalculate the fitness value of the current personal best vector. Each particle then replaces $p_{id}(t)$ value in 8.1 with its current position's fitness value if the $p_{id}(t)$ value is less fit than its current position. However, the "sentry" can only detect the local changes where the "sentry" point resides. Some complex environments only exhibit local changes, which may not be detected by a "sentry". In most real world applications, the fitness value is not stable because of environmental noise interference. The "sentry" may be constantly triggered by the environment noise, requesting all other particles react to the false environment change alarm. In addition, this algorithm alters the classic PSO's decentralized processing model into an essentially centralized control model. All other particles have to depend on one or a limited number of sentries for detecting and reacting to the change of the environment. If the environment change pattern is

non-linear, it is possible that the fitness value of the "sentry" doesn't change when the environment changed. This will make the system unable to detect the environment change. At the same time, if the the "sentry" fitness value is polluted by noise, the "sentry" may generate fault alarm. Designing a particle that is capable of working as a sentry to monitor the environment will also increase the complexity of the entire system.

8.4 Simple Distributed PSO Approach

In [13], we presented a distributed adaptive PSO (DAPSO) algorithm for dynamic environments. In this algorithm, each particle monitors its personal performance at each generation. When the particle personal best value does not get updated for a pre-defined period, the particle will reset its personal best and global best value and change it with the current position fitness value. At the same time, the particle's velocity will be re-initialized. However, the performance monitoring mechanism increases the computation complexity of each particle, causing the algorithm to take longer to converge to the optimum. and can not maintain the optimum result tracking if the environment changes back to static environment.

In this research, we present a modified DAPSO, the simple distributed PSO approach (SDPSO). SDPSO has a simpler particle design for tracking and searching unstable optimum solution in dynamic environments, such as the DF1 test environment proposed by Morrison and De Jong [14]. In SDPSO, there is no specially designed particle to monitor changes in an environment, no additional fitness evaluation, and no performance monitoring. Like the classic PSO, each particle uses equation 8.1 to determine its next velocity. The only difference in the SDPSO algorithm is the particle's best fitness value update mechanism. Instead of using equation 8.3 to update the fitness value, we use equation equation 8.4 for the fitness value update. Equation 8.4 is slightly different compared to the classic PSO fitness value update function equation 8.3.

$$P_i(t+1) = \begin{cases} P_i(t) \times T; \text{ if } f(X_i(t+1)) \leq f(P_i(t)) \times T \\ X_i(t+1); \text{ if } f(X_i(t+1)) > f(P_i(t)) \times T \end{cases} \tag{8.4}$$

In equation 8.4, a notion, the evaporation constant T, is introduced. T has a value between 0 and 1. The personal fitness value that is stored in each particle's memory and the global fitness value of the particle swarm will gradually evaporate (decrease) at the rate of the evaporation constant T in each iteration. Eventually, the personal and global best fitness value will be lower than the fitness value of a particle's current location and the best fitness value will be replaced by the particle's current fitness value. Although all particles have the same evaporation constant T, each particle may update the best fitness value more frequently than other particles. The updating frequency depends on the particle's previous personal best fitness value $f(P)$ and the current fitness value $f(X)$ that the particle acquired. The particle will update its best fitness value more frequently by using the current fitness value when the

$f(P)$ is lower and the $f(X)$ is higher. However, when the $f(P)$ is higher and the $f(X)$ is lower in a changing environment, it indicates the particle's current location is far away from the current optimal solution compared to the distance between the optimal solution and the best fitness value's position stored in the particle's memory.

Usually the new environment (after changing) is closely related to the previous environment from which it evolves. It is beneficial to use knowledge/experience about the previous search space to help searching for the new optima. In this situation, the particle will keep the best fitness value in its memory until that value becomes obsolete. There is a concern that the swarm global best value may be updated too frequently because each particle in the swarm will decrease the existing global best value. However, according to our experiment, the quickly decreased global best will be quickly updated with the latest global best value. This quick global best value updating helps the particle swarm reacts quickly to the fast changing environment. The fitness value update equation enables the swarm to collectively self-adapt to the changing environment even without any mechanism to monitor environment change.

8.5 Dynamic Environment Generator

8.5.1 Environment Landscape

In [15], Angeline proposes three different types of dynamic, linear, circular and random trajectories for generating dynamic environment. However, real world dynamic problems are often much more complex, highly nonlinear, and frequently have high multimodality. In this research, we decided to use the dynamic environment generator originally proposed by Morrison and De Jong [14]. This dynamic environment generator allows us to create non-stationary environments with different degrees of complexity by controlling the morphology of the fitness landscapes as well as the types of changes that can be produced and the severity of those changes. In the DF1 test landscape, a "field of cones" of different heights and different slopes are randomly deployed across the field. The landscape can be specified for any number of dimensions. For a two dimensional case, the landscape fitness evaluation function in DF1 is defined as the following:

$$f(X,Y) = MAX[H_i - R_i \times \sqrt{(X-x_i)^2 + (Y-y_i)^2}];\ (i = 1, ...N) \qquad (8.5)$$

Where N denotes the number of peaks in the environment and (x_i, y_i) represents each cone's location. R_i and H_i represent the cone's height and slope.

8.5.2 Dynamics Generator

The movement of the problem solutions and consequently the change of the fitness value of each solution is simulated with the change in position and height of the

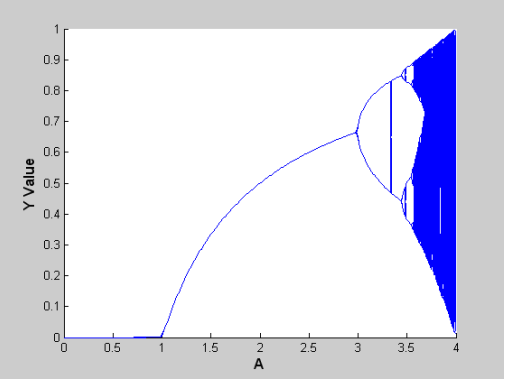


Fig. 8.1. Step size values generated by logic function with different A value [14]

cones in the DF1 generated landscape. Different movement functions generate different types of dynamic environments. In this research, the environment changing rate is controlled by using following discrete moving step size function [14] and the dynamic environment is simulated by the movement of the cone's location (x_i, y_i).

$$Y_i = A \times Y_{i-1} \times (1 - Y_{i-1}) \tag{8.6}$$

Where A is a constant and Y_i is the step size at the iteration i. The Y value produced each iteration will be used to control the changing step sizes of the dynamic environment. In this function, the constant A is the only parameter that can be adjusted for producing different movement behavior. As A is increased, movement will become more complex through the generation of variable moving step sizes. The step size value map generated by equation 8.5 with different A values is shown in figure 8.1. As shown in figure 8.1, when A's value is located in the range of 1 to 3, the function will generate a constant Y value on each iteration. When A's value reaches the range of 3 to 3.4, the function will generate two different values of Y for alternate iterations, which represents two step size of the moving cones. When the A's value increases above 3.5, the function will generate chaotic sequences of Y values.

8.5.3 Measurement for Tracking Optimum Result

The ability for the algorithm to track the optimum in a dynamic environment is measured by the offline error at each iteration. This measurement is the distance between the particle with the best fitness value and the cone centre with the highest peak. The distance value shows the tracking ability of the algorithm during the entire search procedure. If the algorithm can keep at least one particle located in a short distance from the optimal solution at each iteration, regardless of the solution's movement, this distance value will be kept in low value in the whole searching period. The offline error is defined as:

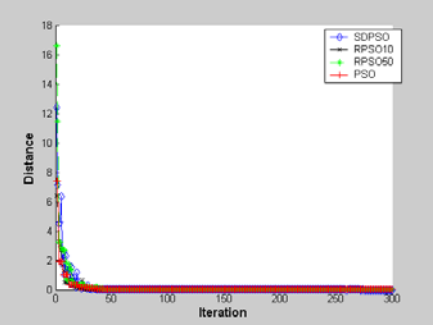


Fig. 8.2. The performance comparing of different PSO algorithms on static environment

$$e_{offline}(t) = \min(d_t(p_i, h)); \ \ (i = 1....N) \tag{8.7}$$

Where $d_t(p_i, h)$ is the distance between particles and the optimum at the iteration t. N denotes the number of particles in the environment.

8.6 Experiment Implementation

To evaluate the ability of the SDPSO algorithm in tracking the movement of the optimum in a dynamic environment, the performance of the SDPSO algorithm and the three other PSO algorithms are compared over the twelve different dynamic environments generated by DF1 function. Besides the proposed SDPSO algorithm and classic PSO algorithm, two modified PSO algorithms, RPSO10 and RPSO50 are included.

For all PSO algorithms, the factors c_1 and c_2 are set to 2.01 and these two factors produce an inertia weight w of 0.729844. The particle population is set to 30, which is considerably smaller than many dynamic PSO approaches. The particles are randomly distributed in a two dimensional environment with 100 units in each dimension. The initial moving velocity of each particle is set to equal to half of the environment width. This will guarantee that at least half of the particles will not run off the environment board in the first move. All particles can only acquire the noise added fitness value $f^n(x)$ instead of real fitness value $f(x)$, as discussed in section 5.3. The evaporation constant T_p for the personal best fitness value and the constant T_g for the global best fitness value in SDPSO are set as 0.85 and 0.9. The details of discovering the values of T_p and T_g are discussed in [13]. The RPSO10 and RPSO50 adopts the algorithm in [12]. RPSO10 re-initializes 10% of particles every 10 iterations and RPSO50 re-initializes 50% of particles every 10 iterations. There are no specially designed environment monitoring particles for monitoring the environment change.

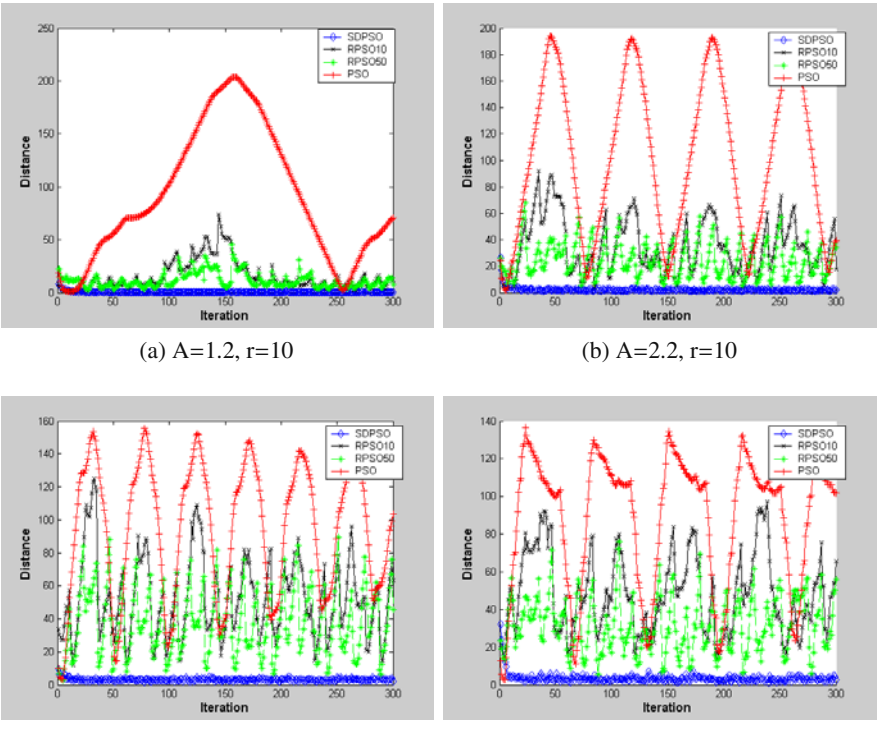
A landscape generated by the DF1 function is used as the test environment in all experiments. Fifty cone shape peaks are randomly deployed in the environment. The

height of all peaks are random values in the range of $[0,1]$. One peak height is intentionally set to 1.1 to make it as the highest peak. The severity of the environment dynamics is controlled by the moving step size of the landscape. In our simulation studies, the height and range of slope for each cone in the environment landscape are set to be constant. All cones in the environment are moved in a step size generated by equation 8.5. Four different A values $1.2, 2.2, 3.3, 3.9$ are used in equation 8.5 for generating four different kinds of movement behavior models. The $A = 1.2$ value generates 0.1667 constant step size. The $A = 2.2$ value generates 0.5455 constant step size. The $A = 3.3$ value generates pairs of step size 0.8236 and 0.4794. When the $A = 3.9$ value, the step size generated in each iteration will be varied in the range 0 to 1. Considering the size of the environment, the actual moving step size of all cones is ten times the Y value generated by equation 8.5. In our research, we chose three different change rates, $r = 1, 5, 10$, as the dynamic environment change frequency, which indicates all cones in the environment will move to a new location every r iterations. The whole iteration number for searching optimum are set as 300. Each algorithm's implementation will run 100 times, and the distance value between the swarm's global best location and the optimum location at every iteration is averaged over 100 runs. The algorithms are implemented with Matlab 6.5 and run on a $3.0GHz$ CPU and 2.0GB memory Windows XP platform.

8.7 Experiment Results

In the first experiment, we evaluated the performance of the four algorithms in locating the optimal solution in static environment. The results are shown in figure 8.2. Figure 8.2 displays the shortest distance values between the optimal solution and the particle that has the highest fitness value. The smaller the distance value, the better the algorithm's solution in the environment. As shown in Figure 8.2, at the initial stage, particles are randomly deployed in the environment and the distance between the particle and the optimal solution are usually high. After 50 iterations all four algorithms can find the optimal solution in the static environment.

Figure 8.3, 8.4, 8.5 show the four different PSO algorithms' performances on a range of different dynamic environments. The A value control the environment changing speed through the 8.5. The r value represents how many time steps the environment will change once. The higher the r value, the slower the environment change. The $X-axis$ is the iteration number and $Y-axis$ is the distance between the particle swarm's global best location vector and the actual optimum location vector. The distance value represents the algorithm's optimum result tracking performance. The smaller the distance value, the better the algorithm's performance. As shown in Figure 8.3 8.4 8.5, the blue line (the bottom lines in monochrome figures) is the performance of SDPSO algorithm. It maintains near zero $Y-axis$ (distance) value in the whole 300 iterations. The red line (the zigzag lines located on top of the monochrome figures) is PSO tracking performance. The huge changing in the



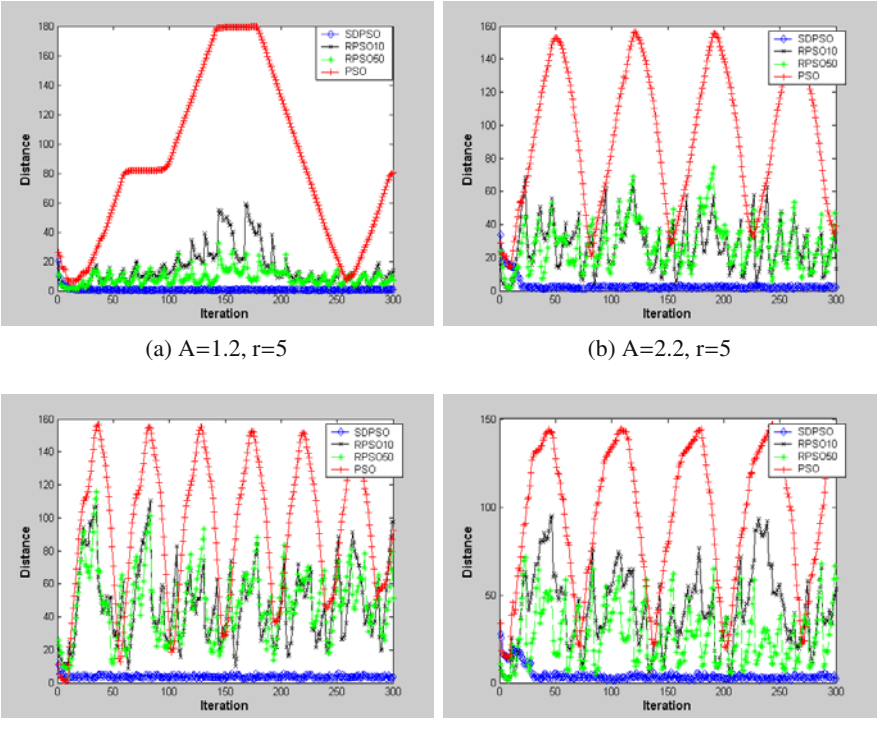
(a) A=1.2, r=10 (b) A=2.2, r=10

(c) A=3.3, r=10 (d) A=3.9, r=10

Fig. 8.3. The performance comparing of different PSO algorithms on various dynamic environments with changing frequency r = 10

$Y - acis$ value represents the PSO lack the capability in tracking the optimum result. The green and black lines (located in the middle part of monochrome figures) can track the optimum result when environment does change very fast. Compared to the other three algorithms, SDPSO performs efficiently in all dynamic environments. Although the optimum is continually changing in the entire searching period, the SDPSO algorithm can maintain the lowest distance (below 5) between the best fitness value particle and the optimal solution.

The RPSO10 and RPSO50 algorithms were originally designed for reinitializing a subset of the swarm particles when an environment change is detected. Without the mechanism for detecting the environment change, both algorithms' initializing frequencies are pre-set as 10 iterations. As shown in Figure 8.3 8.4 8.5, only when the moving step size of the dynamic environment is small (A = 1.2, step size = 1.667), the optimum result tracking performances of both algorithms are acceptable. The RPSO50's performance is better than RPSO10 because 50% of particles are reinitialized to avoid been trapped in local outdated area.



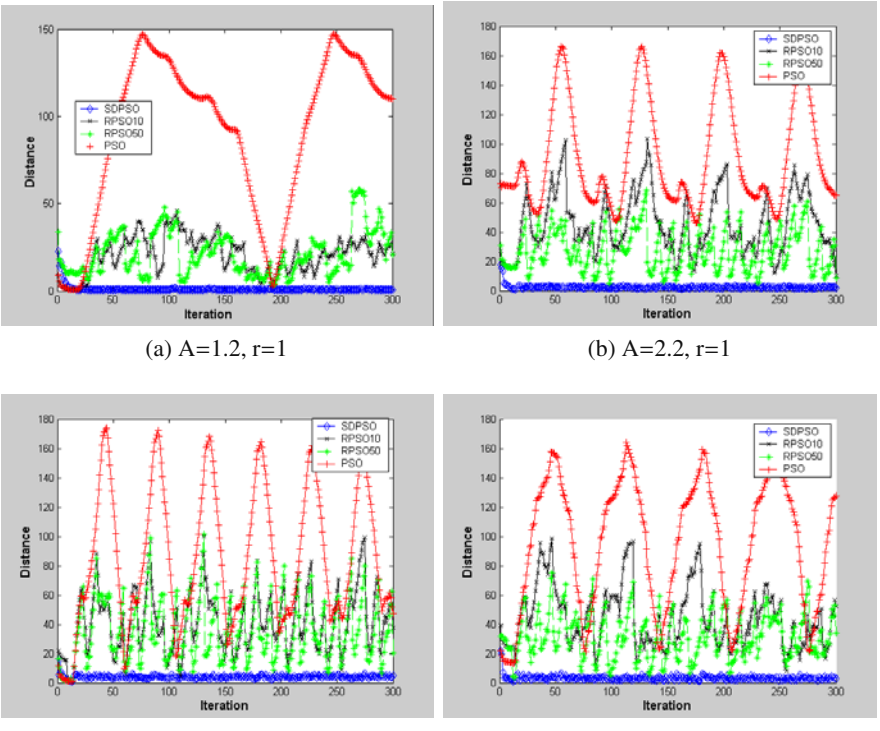
(a) A=1.2, r=5 (b) A=2.2, r=5

(c) A=3.3, r=5 (d) A=3.9, r=5

Fig. 8.4. The performance comparing of different PSO algorithms on various dynamic environments with changing frequency r =5

The distance values for the classic PSO model are very high for most iterations; this indicates that the classic PSO totally failed in the tracking of the movement of the optimum. The saw shape patterns are caused by the cone movement design in the implementation. In our implementation program, if the peak cone approaches the board of the environment, it will automatically reverse its movement direction to avoid moving out of the boundary. This causes the optimum to move back to its original location where the classic particles are trapped. The higher the cone's moving step size value, the quicker the optimum cone move back to its original location. The distance patterns of classic PSO in figure Figure 8.3 8.4 8.5, indicate these features.

The quantitative comparison of the tracking performances between different PSO models is shown in Table 8.1. The average of the distance values of each PSO model in dynamic environment ($A = 2.2, r = 5$) are listed in the table. To avoid the impact of the initial particles' location, only distance values between the 50th and 300th iteration are averaged. The average value in the table shows that SDPSO has the lowest distance value; this indicates the SDPSO can maintain the shortest distance between the swarm global best vector and the actual optimum location.



(a) A=1.2, r=1 (b) A=2.2, r=1

(c) A=3.3, r=1 (d) A=3.9, r=1

Fig. 8.5. The performance comparing of different PSO algorithms on various dynamic environments with changing frequency r = 1

Table 8.1. The average distance value between the optimal solution and the particle that has the highest fitness evaluation value after 50 Iterations for environment ($A = 3.9, r = 10$)

Algorithms	Average distance after 50 iteration
$SDPSO$	3.8493
$RPSO10$	41.1411
$RPSO50$	33.7604
PSO	89.5743

8.8 Discussion and Conclusion

Most papers discussing applications of optimization algorithms only discuss the scenario of the static environment. The performance evaluation of various approaches is mainly based on how fast an approach can find the optimal point in the benchmark problems. It has been proven that PSO is very effective in applications with a static

environment. However, the real world is rarely static, and a frequently changing solution space may cause the optimal solutions to change over time. The optimal solution found at time T_1 may no longer be valid at time T_2. When the problem space is changing, the goal of optimization is not only to acquire the optimal solution but also to track the solution's trajectory as closely as possible. Several approaches have been investigated to enhance the PSO algorithm's ability in dynamic environments. However, most approaches depended on adding sophisticated mechanisms for detecting the environment change and maintaining the particle diversity for reacting to the environment change. These approaches achieved high performance at the cost of exchanging the classic PSO's ideally decentralized processing model with a more centralized and harder to implement algorithm. The major reasons that Particle Swarm Optimization (PSO) is attractive are that the algorithm is easy to implement, non-centralized, and requires few parameters to adjust. There are only ten lines of code to implement the original PSO algorithm [1]. The complexity in many PSO approaches makes them difficult to compete with modified EA algorithm for optimization in dynamic environment.

In this paper, we present a new approach, SDPSO, a modified PSO, for tracking the optimization solution in a dynamic environment. Unlike other adaptive PSO algorithms designed for dynamic environments, which need one or more specially designed "sentry" particles to detect change in the environment and to control other particles' actions, each particle in SDPSO individually updates its knowledge based on the local environment status that the particle perceives. All particles in the SDPSO system are homogenous. The SDPSO algorithm maintains these classic PSO features as well as provides the optimum result tracking capability in dynamic environments. Our simulation experiment results indicate that SDPSO can efficiently track the movement of an optimal solution in different kinds of dynamic environments generated by the DF1 function. Because each particle updates its memory only based on its perception and its knowledge evaporation rate, this SDPSO algorithm can avoid failing to tracking the optimal solution as happens in other modified PSO approaches that are based on resetting the memory periodically.

References

[1] Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, Nagoya, Japan, pp. 39–43 (1995)

[2] Cui, X., Potok, T.E.: Document Clustering Analysis Based on Hybrid PSO+K-means Algorithm. Journal of Computer Sciences Special Issue, 27–33 (2005)

[3] van der Merwe, D.W., Engelbrecht, A.P.: Data clustering using particle swarm optimization. In: 2003 Congress on Evolutionary Computation, Canberra, ACT, Australia, December 8-12, pp. 215–220 (2003)

[4] Clerc, M., Kennedy, J.: The particle swarm-explosion, stability, and convergence in a multidimensional complex space. IEEE Transactions on Evolutionary Computation 6, 58–73 (2002)

[5] Carlisle, A., Dozier, G.: Adapting particle swarm optimization to dynamic environments. In: Proceedings of the International Conference on Artificial Intelligence, Las Vegas, NV, USA, pp. 429–433 (2000)
[6] Li, X., Khanh Hoa, D.: Comparing particle swarms for tracking extrema in dynamic environments, Canberra, ACT, Australia, pp. 1772–1779 (2003)
[7] Eberhart, R.C., Yuhui, S.: Tracking and optimizing dynamic systems with particle swarms, Seoul, South Korea, pp. 94–100 (2001)
[8] Carlisle, A., Dozler, G.: Tracking changing extrema with adaptive particle swarm optimizer. In: Soft Computing, Multimedia Biomedicine, Image Processing and Financial Engineering, Orlando, FL, USA, pp. 265–270 (2002)
[9] Blackwell, T., Branke, J.: Multi-swarm optimization in dynamic environments, Coimbra, Portugal, pp. 489–500 (2004)
[10] Parrott, D., Li, X.: Locating and tracking multiple dynamic optima by a particle swarm model using speciation. IEEE Transactions on Evolutionary Computation 10, 440–458 (2006)
[11] Clerc, M.: The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In: Proceedings of the 1999 Congress on Evolutionary Computation, Washington, DC, USA, pp. 1951–1957 (1999)
[12] Hu, X., Eberhart, R.C.: Adaptive particle swarm optimization: detection and response to dynamic systems, Honolulu, HI, USA, pp. 1666–1670 (2002)
[13] Cui, X., Potok, E.T.: Distributed Adaptive Particle Swarm Optimizer in Dynamic Environment. In: 21st IEEE International Parallel & Distributed Processing Symposium. IEEE Computer Society, Long Beach (2007)
[14] Morrison, R.W., De Jong, K.A.: A test problem generator for non-stationary environments, Washington, DC, USA, pp. 2047–2053 (1999)
[15] Angeline, P.J.: Tracking extrema in dynamic environments, Indianapolis, IN, USA, pp. 335–345 (1997)
[16] Parsopoulos, K.E., Vrahatis, M.N.: Recent approaches to global optimization problems through particle swarm optimization. Natural Computing 1, 235–306 (2002)