Adaptive Filtering Via Particle Swarm Optimization

D. J. Krusienski and W. K. Jenkins
Department of Electrical Engineering
The Pennsylvania State University
University Park, PA

Abstract - **This paper introduces the application of particle swarm optimization techniques to generalized adaptive nonlinear and recursive filter structures. Particle swarm optimization (PSO) is a population based optimization algorithm, similar to the genetic algorithm (GA), that performs a structured randomized search of an unknown parameter space by manipulating a population of parameter estimates to converge on a suitable solution. These types of structured stochastic search techniques are independent of the adaptive filter structure and are capable of converging on the global solution for multimodal optimization problems, which makes them especially useful for optimizing nonlinear and infinite impulse response (IIR) adaptive filters. This paper outlines PSO for adaptive filtering and provides a comparison to the GA for various IIR and nonlinear filter structures.**

1. INTRODUCTION

It is well known that many systems employing adaptive signal processing, such those used in as communications, speech recognition, bio-systems, acoustics, etc. are recursive or nonlinear in nature and would greatly benefit from implementing IIR or nonlinear adaptive processing. The prime obstacle impeding the development of IIR and nonlinear adaptive processing is the lack of practical, efficient, and robust global optimization algorithms. This paper introduces a novel algorithm named particle swarm optimization (PSO) for nonlinear and IIR adaptive filtering. PSO is a population based search similar to the genetic algorithm (GA). But, unlike the GA, PSO has yet to emerge in adaptive filtering literature. Population based algorithms, such as PSO and the GA, are envisioned to receive increasing attention as parallel computing technology continues to progress to the forefront of information processing. As will be shown, PSO demonstrates several advantages over other global optimization algorithms for IIR and nonlinear adaptive filtering.

1.1 Preliminaries

For adaptive filtering problems such as system identification shown in Figure 1, the adaptive filter (AF) attempts to iteratively determine an optimal model for the unknown system (PLANT) based on some function of the error between the output of the AF and the output of the plant. The optimal model or solution is attained when this function of the error is minimized.

In cases where the unknown plant is nonlinear or contains feedback, a simple FIR adaptive filter of reasonable length may not be sufficient to provide an adequate model of the system. In these cases, it is only natural to model the unknown system using an IIR or nonlinear adaptive filter such as a neural network or polynomial filter.

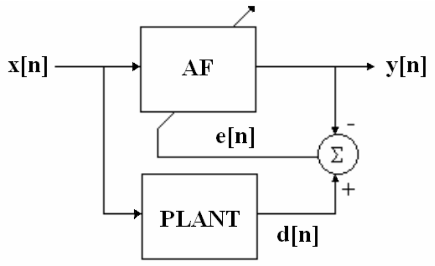


Fig. 1. Adaptive System Identification Configuration

The drawback to IIR and nonlinear adaptive filter structures is that they produce error surfaces that inherently tend to be multimodal. When the error surface is multimodal, local optimization techniques that work well for FIR adaptive filters, such as versions of gradient descent (GD) including the Least Mean Squares (LMS) algorithm and backpropagation for neural networks, are not suitable because they are likely to get trapped in the local minimum and never converge to the global optimum.

1.2 Global Optimization Approaches to Adaptive Filtering

Whereas the gradient approaches rely on the AF structure to update the filter parameters, a structure independent global optimization approach is a structured stochastic search of the error space. In structure independent optimization, a gradient is not calculated and the structure of the AF does not directly influence the parameter updates – aside from the error computation. Due to this property, these types of algorithms are capable of globally optimizing any class of adaptive filter structures or objective functions by assigning the parameter estimates to represent filter tap weights, neural network weights, and even exponents of polynomial terms in the model of an unknown system. There are several different structured stochastic search approaches in the adaptive filtering literature, most notably simulated annealing (SA) [4] and evolutionary algorithms such as the genetic algorithm (GA) [4,5,7,8,9].

2. PARTICLE SWARM OPTIMIZATION

Particle swarm optimization was first developed in 1995 by Eberhart and Kennedy [3], rooted on the notion of swarm intelligence of insects, birds, etc. It has previously been presented as a batch processing type algorithm for numerical optimization, but, as will be shown, can be modified to operate iteratively on windowed input data for on-line adaptation.

Similar to the GA, PSO begins with a random population of individuals; here termed a swarm of particles. As with the GA, each particle in the swarm is a different possible set of the unknown parameters to be optimized. Each particle represents a point in the solution space that has a relative fitness determined by evaluating the parameters with respect to a predetermined fitness function that has an extremum at the desired optimal solution. As with the GA, the parameters can be real-valued or encoded depending on the particular circumstances. The goal is to efficiently search the solution space by swarming the particles toward the best fit solution encountered in previous iterations with the intent of encountering better solutions through the course of the process, eventually converging on a single minimum error solution. The difference between PSO and the GA is in the method that the population is manipulated and the space is searched.

The standard PSO algorithm begins by initializing a random swarm of M particles (an adequate M is dependent on the dimensionality of the problem), each having R unknown parameters to be optimized. Unless there is prior knowledge about the parameter space, the initial particles are typically distributed uniformly about the space to facilitate a global search. At each iteration, the fitness of each particle is evaluated according to the selected fitness function. The algorithm stores and progressively replaces the most fit parameters of each particle (*pbest*$_i$, *i=1,2,...,M*) as well as a single most fit particle (*gbest*) as better fit parameters are encountered. The parameters of each particle (p_i) in the swarm are updated at each iteration (n) according to the following equations:

$$\begin{aligned}\overline{vel}_i(n) &= w * \overline{vel}_i(n-1) \\ &+ acc_1 * diag[e_1, e_2, ..., e_R]_{i1} * (gbest - p_i(n-1)) \\ &+ acc_2 * diag[e_1, e_2, ..., e_R]_{i2} * (pbest_i - p_i(n-1))\end{aligned} \quad (1)$$

$$p_i(n) = p_i(n-1) + \overline{vel}_i(n) \quad (2)$$

$\overline{vel}_i(n)$ =velocity vector of particle i

e_r=random values $\in$ (0,1)

acc_1=acceleration coefficient toward *gbest*

acc_2=acceleration coefficient toward *pbest*$_i$

w=inertia weight

It can be gathered from the update equations that the trajectory of each particle is influenced in a direction determined by the previous velocity and the location of *gbest* and *pbest*$_i$. The acceleration constants are typically chosen in the range 0-1 and serve dual purposes in the algorithm. For one, they control the relative influence toward *gbest* and *pbest*$_i$ respectively by scaling each resulting distance vector, as illustrated for a 2-dimensional case in Figure 2. Secondly, the two acceleration coefficients combined form what is analogous to the step size of an adaptive algorithm. Selecting each coefficient to be relatively small will produce fine searches of a region, while larger coefficients will give a lesser search and faster convergence. The random e_i vectors have R different components, which are randomly chosen in the range 0-1. This allows the particle to take constrained randomly directed steps in a bounded region between *gbest* and *pbest*$_i$, as shown in Figure 2.

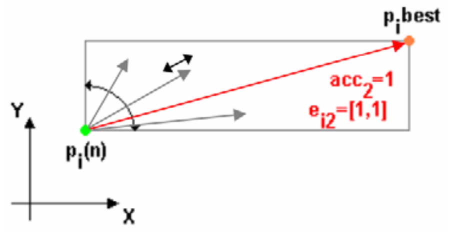


Fig. 2. Example of the scaling and random direction bounds for the vectors

The random e_i components are typically selected using a uniform distribution to provide a high degree randomness to the search, but can be assigned a distribution with a greater density near 1 for a faster, more direct search. The inertia weight controls the influence of the previous velocity. It is typically set to decay from 1 to 0 during some adequate interval in order to allow the algorithm to converge on *gbest*. A single particle update is graphically illustrated in two dimensions in Figure 3. The new particle coordinates can lie anywhere within the bounded region, depending upon the weights and random components associated with each vector. The particle update bounds in Figure 3 are basically composed of all of the bounded regions for each vector as shown in Figure 2, with the addition of the non-random velocity component.

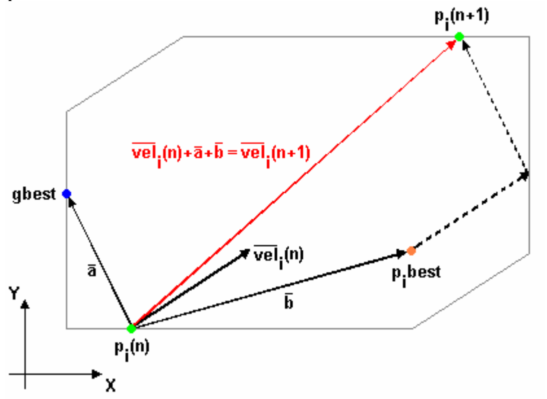


Fig. 3. Example of the possible search region for a single particle

When a new *gbest* is encountered during the update process, all other particles begin to swarm toward the new *gbest*, continuing the directed global search along the way. The search regions continue to decrease as new *pbest$_i$s* are found within the search regions. When all of the particles in the swarm have converged to *gbest* or a suitable minimum error condition is met, the *gbest* parameters characterize the minimum error solution determined by the algorithm.

2.1 PSO for Adaptive Filtering

Commonly in numerical optimization problems, the particle fitness is evaluated at each iteration using the entire input data. In typical on-line adaptive filtering problems, the entire input data is not available or is too lengthy to process in an efficient manner. Therefore, the input data must be processed and evaluated in blocks. In this case, at each iteration, the fitness or cost function will only provide an estimate of the actual error due to the entire input data. The quality of this estimate is dependent on the statistical nature of the input data and can be improved by averaging the error estimates over a window of previous input data.

In adaptive filtering, the mean squared error (MSE) between the output of the unknown system and the output of the AF is the typical cost function, and will hence be used for the fitness evaluation of each particle in the on-line form of PSO. For an adaptive system identification configuration as shown in Figure 1, the windowed MSE cost function is as follows:

$$J_i(n) = \frac{1}{N}\left[\sum_{k=0}^{N}(d(n-k) - y_{k,i}(n))^2\right] \tag{3}$$

$$y_{k,i}(n) = f[x(n-k-1), x(n-k-2), ..., x(n-k-L), p_i(n)] \tag{4}$$

where f is a nonlinear operator, N is the length of the window over which the error is averaged, and L is the amount of delay in the filter. The AF output $y_k(n)$ may also be a function of past values of itself if it contains feedback, or also a function of intermediate variables if the AF has a cascaded structure. When $J(n)$ is minimum, the AF parameters provide the best possible representation of the unknown system.

2.2 Modifications and Variations

PSO, in its most simplistic form described previously, can be made more efficient. In order to improve the efficiency of PSO, its weaknesses must be recognized. The following are a few of the concerns with standard PSO:

1) When a particle is found to be the new *gbest* of the swarm, all of the other particles begin to move toward it. If the new *gbest* particle is an outlying particle with respect to the swarm, the rest of the swarm can tend to move toward the new *gbest* from the same general direction. This may potentially leave some critical region around the new minimum excluded from the search.

2) Particles closer to *gbest* will tend to quickly converge on it and become stagnant while the other more distant particles continue to search. A stagnant particle is essentially useless because its fitness continues to be evaluated but it no longer aids the search.

3) The initial stages of the algorithm tend to give a broad search where key points can be missed because the individual search regions typically decrease in size rather quickly as new bests are found. If *gbest* is at a local minimum and continues to be after sufficient iterations, which would likely be caused by an inadequate swarm size or large initial acceleration, the swarm can undesirably converge on the local minimum.

4) To a lesser extent, the efficiency of the algorithm may be improved because particles closer to *gbest* may have already searched nearly the same region as more distant particles, which can be unnecessarily redundant.

These four concerns can be effectively combated with the following simple modifications:

1) When a new *gbest* is encountered, random particles can be re-randomized about the new *gbest*. This will act to ensure that the region around *gbest* is searched from all directions, while still keeping a portion of the swarm searching somewhat globally.

2) Stagnancy of particles can be eliminated by slightly varying the random parameters of each particle at every iteration, similar to mutation in the GA. This will have little effect on particles distant from *gbest* because this random influence should be relatively small compared to the random update of equation (1). However, this will eliminate any stagnant particles and generate a finer search about *gbest*.

3) The issue of premature convergence on a local minimum is occasionally inevitable, depending on the characteristics of the error surface or other constraints, but its likelihood can be decreased by continually re-randomizing a random portion of the particles over the entire parameter space and allowing them to converge. This will in effect continually generate unique search paths, which can increase the probability of finding the global optimum. This continuous probing of the space is also beneficial for tracking a non-stationary input or dynamic plant.

   Another alternative to better facilitate the search includes swarming particles toward centers of mass defined by groups of particles or previous bests rather than a single point, or using multiple sub-swarms that swarm toward separate centers. These two modifications can add more diversity and better distribute the swarm, decreasing the likelihood of converging on a local minimum.

4) The re-randomization procedures described previously can be planned such that the space is searched more efficiently and redundancy is kept to a minimum. One technique for accomplishing this is to re-randomize according to an

appropriate distribution. By the nature of the algorithm, the region close to the *gbest* is more efficiently searched because the density of the particles has been greatest in that region by the time of convergence. It may be unnecessarily redundant to uniformly redistribute some of the particles around *gbest* or the entire space, because all particles will traverse the inner perimeter eventually. Re-randomizing with a distribution that is sparse about *gbest* would be more effective in terms of reducing unnecessary redundancy.

In order to ensure convergence of the swarm, the variance of the mutation and selected re-randomization distributions must decrease according to some schedule. A reasonable variance decay curve, shown in Figure 4, is given in equation (5).

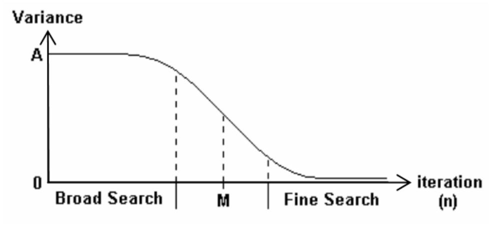


Fig. 4. Variance schedule

$$Variance(n) = \frac{-A}{\left(1 + e^{\frac{-n+M}{S}}\right)} + A \qquad (5)$$

M: transition midpoint
S: transition slope adjustment

This schedule specifies a wide search (large variance) initially, and then decays toward a finer search (small variance) at a suitable interval after which the space is presumed to be searched sufficiently. The slope of the transition region and search intervals can be tuned for the specific problem. This schedule may also be applied to the acceleration coefficients to further tune the search. The re-randomization and acceleration schedules can be coordinated to optimize the convergence speed and search efficiency.

If the dimensionality of the problem becomes an issue, another variation is to separate the filter parameters into multiple independent swarms that will each search a lower dimensional space. This can potentially lead to better convergence properties if the parameters designated to each swarm can be de-coupled.

3. Simulation Examples

In the following simulations, the properties of PSO and a modified version of PSO (MPSO) are compared to the GA for several system identification problems. All adaptive filters are matched in order and structure to the unknown plant. All algorithms were initialized with the same population of real-valued parameters and allowed to evolve. Each algorithm was tuned such that the population converged before the last iteration. The population sizes and algorithm parameters were chosen to experimentally provide the best results for each case. Each plot in Figures 5 and 6 shows the MSE averaged over 50 trials. The specifics of each algorithm are as follows:

PSO: The standard PSO algorithm is implemented with both acceleration constants weighted equally. The acceleration constant was chosen to give a reasonable balance between the search quality and convergence speed for each case.

MPSO: The modified PSO algorithm uses the standard PSO algorithm as a base, implementing the four modifications suggested earlier. As with standard PSO, both acceleration constants are weighted equally, using the same values as PSO. The mutations are randomly selected uniformly, with the variance decreasing according to Figure 4 to ensure convergence of the population. The variances of the re-randomization distributions also follow Figure 4. The re-randomization is sparsely distributed near *gbest*.

GA: The genetic algorithm uses the most fit half of the population to generate offspring, which replace the least fit half of the population. A crossover rate of 0.5 and a mutation rate of 0.25 are used for the evolution. The mutations are randomly selected uniformly, with the variance decreasing according to Figure 4 to aid the convergence of the population.

3.3.1 IIR Plant

For this example, the plant is a second order IIR filter taken from [9]. The algorithms are set to learn the five filter coefficients given in equation (7). A population of 50 was used and the acceleration constants of PSO and MPSO were selected to be 0.8.

$$PLANT = \frac{1.25z^{-1} - 0.25z^{-2}}{1 - 0.3z^{-1} + 0.4z^{-2}} \qquad (6)$$

$$AF = \frac{p_i^1 + p_i^2 z^{-1} + p_i^3 z^{-2}}{p_i^4 - p_i^5 z^{-1} + p_i^6 z^{-2}} \qquad (7)$$

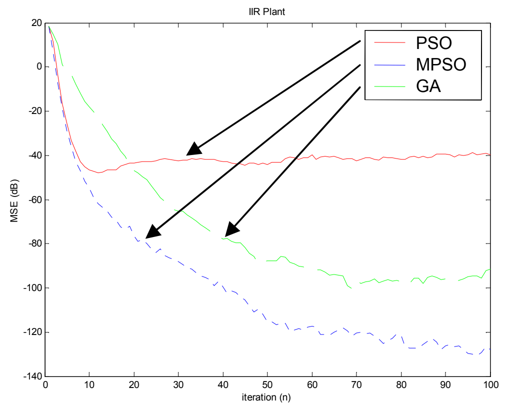
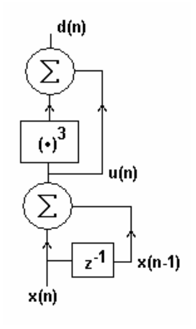


Fig. 5. Second-order IIR Plant

3.3.2 Polynomial Filter

This example illustrates how PSO can simultaneously update coefficients as well as exponential terms. The output of an FIR filter (equation (8)) is passed through a polynomial nonlinearity (equation (9)) to produce the desired training signal *d(n)*. The AF filter is set to learn the four weighting coefficients, as well as the two exponential terms (equations (10) & (11)). A population of 100 was used and the acceleration constants of PSO and MPSO were selected to be 0.3.



$$u(n) = -0.3x(n) + 0.8x(n-1) \quad (8)$$

$$d(n) = f[u(n)] = 0.5u^3(n) - 0.7u^1(n) \quad (9)$$

$$\hat{u}(n) = p_i^1(n)x(n) + p_i^2(n)x(n-1) \quad (10)$$

$$\hat{d}(n) = f[\hat{u}(n)] = p_i^3(n)u^{p_i^5(n)}(n) - p_i^4(n)u^{p_i^6(n)}(n) \quad (11)$$

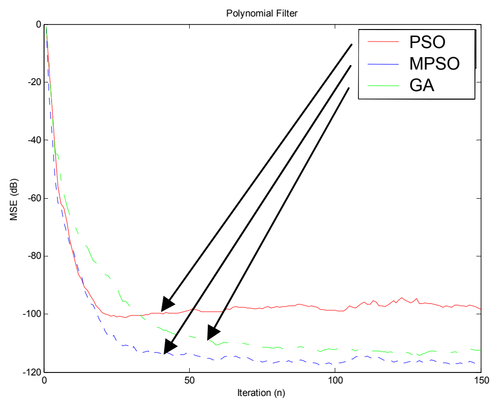


Fig. 6. Polynomial Filter

4. Discussion

From Figures 5 and 6, standard PSO exhibits a fast convergence initially, but fails to improve because the swarm quickly becomes stagnant and converges to a suboptimal solution. The MPSO particles are not allowed to stagnate, which enables the algorithm to surpass both PSO and the GA. MPSO displays a much faster convergence compared to the GA because the GA doesn't have an explicit step size and must evolve at its own rate.

Because the computational complexity of these algorithms increases proportionally with the population size, it is desirable to work with smaller populations when possible. For the GA, when the population size is relatively small, parameters that exhibit less sensitivity with respect to the error surface can be purged from the population (through selection and crossover) in the initial stages of the algorithm. This can greatly reduce the population diversity, leading to an inferior solution. Similar purging is evident in the IIR case because the numerator coefficients will to have a greater influence on the error in the initial stages of the updates. This does not occur in PSO due to the prescribed particle memories, which is one of the reasons that MPSO results in a lower minimum MSE in Figure 5. Similarly, for the polynomial plant results in Figure 6, the relative parameter sensitivities are increased by the introduction of the exponential terms. This creates an error surface fraught with local minima within regions exhibiting comparatively low error values. Again, the PSO swarm remains more resilient, motivating the hypothesis that PSO will continue to be more effective as the population size decreases.

By examining the most basic form of each algorithm, PSO seems to demonstrate a more objective direction, or controlled randomness than the GA. PSO particles take directed steps within well defined bounded regions, whereas the GA updates are less directed and possibly more redundant, which can adversely effect the convergence rate. The PSO algorithm eliminates some potential search redundancy by retaining the previous particle bests and velocities.

The aforementioned properties of PSO make it easier to visualize and predict the search process. This, in turn, provides a more objective plan to assigning the search parameters such as accelerations and the inertia weight. Conversely, the search parameter assignments of the GA (mutation, crossover, etc.) are more heuristic. Another advantage of PSO is that the convergence of the search can be more readily controlled via the acceleration coefficients, which are not present in the GA. Because of this, the convergence rate of PSO can be tuned to be significantly faster than the GA, especially as the dimensionality of the space increases. This property of PSO makes it better suited for on-line adaptive filtering problems.

5. References

[1] Goldberg, D.E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989
[2] Haykin, S., *Adaptive Filter Theory,* 4th ed. Prentice Hall, 2001.
[3] Kennedy, J., Eberhart, R. C., and Shi, Y., *Swarm intelligence* San Francisco: Morgan Kaufmann Publishers, 2001.
[4] Nambiar, R. and Mars, P., "Genetic and Anealing Approaches to Adaptive Digital Filtering," *Proc. 26th Asilomar Conf. on Signals, Systems, and Computers*, vol. 2, pp. 871-875, Oct. 1992.
[5] Ng, S.C., Leung, S.H., Chung, C.Y., Luk, A., and Lau, W.H., "The Genetic Search Approach," *IEEE Signal Processing Magazine*, pp. 28-46, November 1996.
[6] Shynk, J.J., "Adaptive IIR Filtering," *IEEE ASSP Magazine*, pp. 4-21, April 1989.
[7] Tang, K.S., Man, K.F., He, Q. "Genetic Algorithms and their Applications," *IEEE Signal Processing Magazine*, pp. 22-37, November 1996.
[8] White, M.S. and Flockton, S.J., Chapter in *Evolutionary Algorithms in Engineering Applications*, Editors: D. Dasgupta and Z. Michalewicz, Springer Verlag, 1997.
[9] Yao L., Sethares W.A., "Nonlinear Parameter Estimation via the Genetic Algorithm," *IEEE Transactions on Signal Processing*, vol.42, April 1994.