Particle swarm optimization: Development, applications and resources

Conference Paper · February 2001

DOI: 10.1109/CEC.2001.934374 · Source: IEEE Xplore

Yuhui Shi
Southern University of Science and Technology

Particle Swarm Optimization:
Developments, Applications and Resources

Russell C. Eberhart
Purdue School of Engineering and Technology
799 West Michigan Street
Indianapolis, IN 46202 USA
Eberhart@engr.iupui.edu

Yuhui Shi
EDS Embedded Systems Group
1401 E. Hoffer Street
Kokomo, IN 46902 USA
Yuhui.Shi@EDS.com

Abstract- This paper focuses on the engineering and computer science aspects of developments, applications, and resources related to particle swarm optimization. Developments in the particle swarm algorithm since its origin in 1995 are reviewed. Included are brief discussions of constriction factors, inertia weights, and tracking dynamic systems. Applications, both those already developed, and promising future application areas, are reviewed. Finally, resources related to particle swarm optimization are listed, including books, web sites, and software. A particle swarm optimization bibliography is at the end of the paper.

1 Introduction

Particle swarm optimization (PSO) is an evolutionary computation technique developed by Kennedy and Eberhart in 1995 (Kennedy and Eberhart 1995; Eberhart and Kennedy, 1995; Eberhart, Simpson, and Dobbins 1996). Thus, at the time of the writing of this paper, PSO been around for just over five years. Already, it is being researched and utilized in over a dozen countries. It seems like an appropriate time to step back and look at where we are, how we got here, and where we think we may be going.

This paper is a review of developments, applications, and resources related to PSO since its origin in 1995. It is written from an engineering and computer science perspective, and is not meant to be comprehensive in areas such as the social sciences.

Following the introduction, major developments in the particle swarm algorithm since its origin in 1995 are reviewed. The original algorithm is presented first. Following are brief discussions of constriction factors, inertia weights, and tracking dynamic systems.

Applications, both those already developed, and promising future application areas, are reviewed. Those already developed include human tremor analysis, power system load stabilization, and product mix optimization.

Finally, particle swarm optimization resources are listed. Included are books, web sites and software. Sources for obtaining free software are included. At the end of the paper, a particle swarm optimization bibliography is presented.

2 Developments

2.1 The Original Version

The particle swarm concept originated as a simulation of a simplified social system. The original intent was to graphically simulate the graceful but unpredictable choreography of a bird flock. Initial simulations were modified to incorporate nearest-neighbor velocity matching, eliminate ancillary variables, and incorporate multidimensional search and acceleration by distance (Kennedy and Eberhart 1995, Eberhart and Kennedy 1995). At some point in the evolution of the algorithm, it was realized that the conceptual model was, in fact, an optimizer. Through a process of trial and error, a number of parameters extraneous to optimization were eliminated from the algorithm, resulting in the very simple original implementation (Eberhart, Simpson and Dobbins 1996).

PSO is similar to a genetic algorithm (GA) in that the system is initialized with a population of random solutions. It is unlike a GA, however, in that each potential solution is also assigned a randomized velocity, and the potential solutions, called *particles*, are then "flown" through the problem space.

Each particle keeps track of its coordinates in the problem space which are associated with the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called *pbest*. Another "best" value that is tracked by the *global* version of the particle swarm optimizer is the overall best value, and its location, obtained so far by any particle in the population. This location is called *gbest*.

The particle swarm optimization concept consists of, at each time step, changing the velocity (accelerating) each particle toward its *pbest* and *gbest* locations (global version of PSO). Acceleration is weighted by a random term, with separate random numbers being generated for acceleration toward *pbest* and *gbest* locations.

There is also a *local* version of PSO in which, in addition to *pbest*, each particle keeps track of the best solution, called *lbest*, attained within a *local* topological neighborhood of particles.

The (original) process for implementing the global version of PSO is as follows:

1) Initialize a population (array) of particles with random positions and velocities on *d* dimensions in the problem space.
2) For each particle, evaluate the desired optimization fitness function in *d* variables.
3) Compare particle's fitness evaluation with particle's *pbest*. If current value is better than *pbest*, then set *pbest* value equal to the current value, and the *pbest* location equal to the current location in *d*-dimensional space.
4) Compare fitness evaluation with the population's overall previous best. If current value is better than *gbest*, then reset *gbest* to the current particle's array index and value.
5) Change the velocity and position of the particle according to equations (1) and (2), respectively:

$$v_{id} = v_{id} + c_1 * rand() * (p_{id} - x_{id}) + c_2 * Rand() * (p_{gd} - x_{id}) \quad (1)$$

$$x_{id} = x_{id} + v_{id} \quad (2)$$

6) Loop to step 2) until a criterion is met, usually a sufficiently good fitness or a maximum number of iterations (generations).

Particles' velocities on each dimension are clamped to a maximum velocity *Vmax*. If the sum of accelerations would cause the velocity on that dimension to exceed *Vmax*, which is a parameter specified by the user, then the velocity on that dimension is limited to *Vmax*.

Vmax is therefore an important parameter. It determines the resolution, or fineness, with which regions between the present position and the target (best so far) position are searched. If *Vmax* is too high, particles might fly past good solutions. If *Vmax* is too small, on the other hand, particles may not explore sufficiently beyond locally good regions. In fact, they could become trapped in local optima, unable to move far enough to reach a better position in the problem space.

The acceleration constants c_1 and c_2 in equation (1) represent the weighting of the stochastic acceleration terms that pull each particle toward *pbest* and *gbest* positions. Thus, adjustment of these constants changes the amount of "tension" in the system. Low values allow particles to roam far from target regions before being tugged back, while high values result in abrupt movement toward, or past, target regions.

Early experience with particle swarm optimization (trial and error, mostly) led us to set the acceleration constants c_1 and c_2 each equal to 2.0 for almost all applications. *Vmax* was thus the only parameter we routinely adjusted, and we often set it at about 10-20% of the dynamic range of the variable on each dimension.

Based, among other things, on findings from social simulations, it was decided to design a "local" version of the particle swarm. In this version, particles have information only of their own and their neighbors' bests, rather than that of the entire group. Instead of moving toward a kind of stochastic average of *pbest* and *gbest* (the best location of the entire group), particles move toward points defined by *pbest* and "*lbest,*" which is the index of the particle with the best evaluation in the particle's *neighborhood*.

If the neighborhood size is defined as two, for instance, particle(i) compares its fitness value with particle(i-1) and particle(i+1). Neighbors are defined as topological neighbors; neighbors and neighborhoods do not change during a run. For the neighborhood version, the only change to the process defined in the six steps before is the substitution of p_{ld}, the location of the *neighborhood best*, for p_{gd}, the *global best*, in equation (1). Early experience (again, mainly trial and error) led to neighborhood sizes of about 15 percent of the population size being used for many applications. So, for a population of 40 particles, a neighborhood of six, or three topological neighbors on each side, was not unusual.

The population size selected was problem-dependent. Population sizes of 20-50 were probably most common. It was learned early on that smaller populations than were common for other evolutionary algorithms (such as genetic algorithms and evolutionary programming) were optimal for PSO in terms of minimizing the total number of evaluations (population size times the number of generations) needed to obtain a sufficient solution.

2.2 Inertia Weight

The maximum velocity *Vmax* serves as a constraint to control the global exploration ability of a particle swarm. As stated earlier, a larger *Vmax* facilitates global exploration, while a smaller *Vmax* encourages local exploitation. The concept of an inertia weight was developed to better control exploration and exploitation. The motivation was to be able to eliminate the need for *Vmax*. The inclusion of an inertia weight in the particle swarm optimization algorithm was first reported in the literature in 1998 (Shi and Eberhart 1998a, 1998b).

Equations (3) and (4) describe the velocity and position update equations with an inertia weight included. It can be seen that these equations are identical to equations (1) and (2) with the addition of the inertia weight w as a multiplying factor of v_{id} in equation (3).

The use of the inertia weight w has provided improved performance in a number of applications. As originally developed, w often is decreased linearly from about 0.9 to 0.4 during a run. Suitable selection of the inertia weight provides a balance between global and local exploration and exploitation, and results in fewer iterations on average to find a sufficiently optimal solution. [A different form of *w*, explained later, is currently being used by one of the authors (RE).]

$$v_{id} = w*v_{id} + c_1 * rand() * (p_{id} - x_{id}) + c_2 * Rand() * (p_{gd} - x_{id}) \quad (3)$$

$$x_{id} = x_{id} + v_{id} \quad (4)$$

After some experience with the inertia weight, it was found that although the maximum velocity factor *Vmax* couldn't always be eliminated, the particle swarm algorithm works well if *Vmax* is set to the value of the dynamic range of each variable (on each dimension). Thus, the need to think about how to set *Vmax* each time the particle swarm algorithm is used is eliminated.

Another approach to using an inertia weight is to adapt it using a fuzzy system. The first paper published reporting this approach used the Rosenbrock function with asymmetric initialization as the benchmark function (Shi and Eberhart 2000). The fuzzy system comprised nine rules, with two inputs and one output. Each input and the output had three fuzzy sets defined. One input was the global best fitness for the current generation; the other was the current inertia weight. The output was the change in intertia weight. The results reported in the paper showed that by using a fuzzy adaptive inertia weight the performance of particle swarm optimization can be significantly improved in terms of the mean best fitness achieved in a given number of iterations.

2.3 Constriction Factor

Because particle swarm optimization originated from efforts to model social systems, a thorough mathematical foundation for the methodology was not developed at the same time as the algorithm. Within the last few years, a few attempts have been made to begin to build this foundation.

Recent work done by Clerc (1999) indicates that use of a *constriction factor* may be necessary to insure convergence of the particle swarm algorithm. A detailed discussion of the constriction factor is beyond the scope of this paper, but a simplified method of incorporating it appears in equation (5), where K is a function of c_1 and c_2 as reflected in equation (6).

$$v_{id} = K*[v_{id} + c_1 * rand() * (p_{id} - x_{id}) + c_2 * Rand() * (p_{gd} - x_{id})] \quad (5)$$

$$K = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|}, \text{ where } \varphi = c_1 + c_2, \quad \varphi > 4 \quad (6)$$

Typically, when Clerc's constriction method is used, φ is set to 4.1 and the constant multiplier K is thus 0.729. This results in the previous velocity being multiplied by 0.729 and each of the two *(p-x)* terms being multiplied by 0.729*2.05=1.49445 (times a random number between 0 and 1).

In initial experiments and applications, *Vmax* was set to 100,000, since it was believed that *Vmax* isn't necessary when Clerc's constriction approach is used. However, from subsequent experiments and applications (Eberhart and Shi 2000) it has been concluded that a better approach to use as a "rule of thumb" is to limit *Vmax* to *Xmax*, the dynamic range of each variable on each dimension, while selecting *w*, c_1, and c_2 according to equations (5) and (6).

2.4 Tracking and Optimizing Dynamic Systems

Most applications of evolutionary algorithms are to the solution of static problems. Many real-world systems, however, change state frequently (or continuously). These system state changes result in a requirement for frequent, sometimes almost continuous, re-optimization. It has been demonstrated that particle swarm optimization can be successfully applied to tracking and optimizing dynamic systems (Eberhart and Shi 2001a).

A slight adjustment was made to the intertia weight for this purpose. The inertia weight *w* in equation (3) was set equal to [0.5 + (Rnd/2.0)]. This produces a number randomly varying between 0.5 and 1.0, with a mean of 0.75. This was selected in the spirit of Clerc's constriction factor described above, which sets *w* to 0.729. Constants c_1 and c_2 in equation (3) were set to 1.494, also according to Clerc's constriction factor.

The random component of the inertia weight since, when tracking a dynamic system it can not be predicted whether exploration (a larger inertia weight) or exploitation (a smaller inertia weight) will be better at any given time. An inertia weight that varies roughtly within our previous range addresses this.

For the limited testing done (Eberhart and Shi 2001a) using the parabolic function, the performance of particle swarm optimization was shown to compare favorably (faster to converge, higher fitness) with other evolutionary algorithms for all conditions tested. The ability to track a 10-dimensional function was demonstrated.

3 Applications

One of the reasons that particle swarm optimization is attractive is that there are very few parameters to adjust. One version, with very slight variations (or none at all) works well in a wide variety of applications.

Particle swarm optimization has been used for approaches that can be used across a wide range of applications, as well as for specific applications focused on a specific requirement. In this brief section, we cannot describe all of particle swarm's applications, or describe any single application in detail. Rather, we summarize a small sample.

The first application represents an approach, or method, that can be used for many applications: evolving artificial neural networks. Particle swarm optimization is being used to evolve not only the network weights, but also the network structure (Eberhart and Shi 1998a, Kennedy, Eberhart and Shi 2001). The method is so simple and efficient that we have almost completely ceased using traditional neural network training paradigms such as backpropagation. Instead, we evolve our networks using particle swarms. The approach is effective for any network architecture.

As an example of evolving neural networks, particle swarm optimization has been applied to the analysis of human tremor. The diagnosis of human tremor, including Parkinson's disease and essential tremor, is a very challenging area. PSO has been used to evolve a neural network that distinguishes between normal subjects and those with tremor. Inputs to the network are normalized movement amplitudes obtained from an actigraph system. The method is fast and accurate (Eberhart and Hu 1999).

As another example, end milling is a fundamental and commonly encountered metal removal operation in manufacturing environments. While development of computer numerically controlled machine tools has significantly improved productivity, the operation is far from optimized. None of the methods previously developed is sufficiently general to be applied in numerous situations with high accuracy. A new and successful approach involves using artificial neural networks for process simulation and PSO for multi-dimensional optimization. The application was implemented using computer-aided design and computer-aided manufacturing (CAD/CAM) and other standard engineering development tools as the platform (Tandon 2000).

Another application is the use of particle swarm optimization for reactive power and voltage control by a Japanese electric utility (Yoshida et al., 1999). Here, particle swarm optimization was used to determine a control strategy with continuous and discrete control variables, resulting in a sort of hybrid binary and real-valued version of the algorithm. Voltage stability in the system was achieved using a continuation power flow technique.

Particle swarm optimization has also been used in conjunction with a backpropagation algorithm to train a neural network as a state-of-charge estimator for a battery pack for electric vehicle use. Determination of the battery pack state of charge is an important issue in the development of electric and hybrid/electric vehicle technology. The state of charge is basically the fuel gauge of an electric vehicle. A strategy was developed to train the neural network based on a combination of particle swarm optimization and the backpropagation algorithm. One innovation was to use this combination to optimize the training data set. We can't say much more about this, since the application is proprietary, but the results are significantly more accurate than those provided by any other method (Eberhart, Simpson and Dobbins 1996).

Finally, one of the most exciting applications of PSO is that by a major American corporation to ingredient mix optimization. In this work, "ingredient mix" refers to the mixture of ingredients that are used to grow production strains of microorganisms that naturally secrete or manufacture something of interest. Here, PSO was used in parallel with traditional industrial optimization methods. PSO provided an optimized ingredient mix that provided over twice the fitness as the mix found using traditional methods, at a very different location in ingredient space. PSO was shown to be robust: the occurrence of an ingredient becoming contaminated hampered the search for a few iterations but in the end did not result in poor final results. PSO, by its nature, searched a much larger portion of the problem space than the traditional method.

Generally speaking, particle swarm optimization, like the other evolutionary computation algorithms, can be applied to solve most optimization problems and problems that can be converted to optimization problems. Among the application areas with the most potential are system design, multi-objective optimization, classification, pattern recognition, biological system modeling, scheduling (planning), signal processing, games, robotic applications, decision making, simulation and identification. Examples include fuzzy controller design, job shop scheduling, real time robot path planing, image segmentation, EEG signal simulation, speaker verification, time-frequency analysis, modeling of the spread of antibiotic resistance, burn diagnosing, gesture recognition and automatic target detection, to name a few.

3 Resources

The first book to include a section on particle swarm optimization was Eberhart, Simpson and Dobbins (1996). See Kennedy and Eberhart (1999) for a book chapter on PSO. An entire book is now available, however, on the subject of swarms: *Swarm Intelligence* (Kennedy, Eberhart and Shi 2001) discusses both the social and psychological as well as the engineering and computer science aspects of swarm intelligence. The web site for the book, www.engr.iupui.edu/~eberhart/web/PSObook.html, is a guide to a variety of resources related to particle swarm optimization. Included are Java applets that can be run online illustrating the optimization of a variety of benchmark functions. The user can select a variety of parameters. Also on the web site is PSO software written in C++, Visual BASIC and Java that can be downloaded. A variety of links to other web sites are also provided.

Acknowledgments

The cooperation and comments of Jim Kennedy are appreciated and acknowledged. Portions of this paper are adapted from *Swarm Intelligence* published in March 2001 by Morgan Kaufmann Publishers. Other portions are adapted from chapter six of *Computational Intelligence PC Tools* published in 1996 by Academic Press Professional.

Their permission to use this material is gratefully acknowledged.

Particle Swarm Optimization Bibliography

Carlisle, A., and Dozier, G. (2001). An off-the-shelf PSO. *Proceedings of the Workshop on Particle Swarm Optimization.* Indianapolis, IN: Purdue School of Engineering and Technology, IUPUI (in press).

Clerc, M. (1999). The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. *Proc. 1999 Congress on Evolutionary Computation*, Washington, DC, pp 1951-1957. Piscataway, NJ: IEEE Service Center.

Eberhart, R. C., and Hu, X. (1999). Human tremor analysis using particle swarm optimization. *Proc. Congress on Evolutionary Computation 1999*, Washington, DC, pp 1927–1930. Piscataway, NJ: IEEE Service Center.

Eberhart, R. C., and Kennedy, J. (1995). A new optimizer using particle swarm theory. *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, Nagoya, Japan, 39-43. Piscataway, NJ: IEEE Service Center.

Eberhart, R. C., Simpson, P. K., and Dobbins, R. W. (1996). *Computational Intelligence PC Tools*. Boston, MA: Academic Press Professional.

Eberhart, R. C., and Shi, Y. (1998)(a). Evolving artificial neural networks. *Proc. 1998 Int'l. Conf. on Neural Networks and Brain*, Beijing, P.R.C., PL5-PL13.

Eberhart, R. C. and Shi, Y. (1998)(b). Comparison between genetic algorithms and particle swarm optimization. In V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, Eds. *Evolutionary Programming VII: Proc. 7th Ann. Conf. on Evolutionary Programming Conf.*, San Diego, CA. Berlin: Springer-Verlag.

Eberhart, R. C., and Shi, Y. (2000). Comparing inertia weights and constriction factors in particle swarm optimization. *Proc. Congress on Evolutionary Computation 2000*, San Diego, CA, pp 84-88.

Eberhart, R. C., and Shi, Y. (2001)(a). Tracking and optimizing dynamic systems with particle swarms. *Proc. Congress on Evolutionary Computation 2001*, Seoul, Korea. Piscataway, NJ: IEEE Service Center. (in press)

Eberhart, R. C., and Shi, Y. (2001)(b). Particle swarm optimization: developments, applications and resources. *Proc. Congress on Evolutionary Computation 2001*, Seoul, Korea. Piscataway, NJ: IEEE Service Center. (in press)

Fan, H.-Y., and Shi, Y. (2001). Study of Vmax of the particle swarm optimization algorithm. *Proceedings of the Workshop on Particle Swarm Optimization.* Indianapolis, IN: Purdue School of Engineering and Technology, IUPUI (in press).

Fukuyama Y., Yoshida, H. (2001). A Particle Swarm Optimization for Reactive Power and Voltage Control in Electric Power Systems, *Proc. Congress on Evolutionary Computation 2001*, Seoul, Korea. Piscataway, NJ: IEEE Service Center. (in press)

He, Z.,Wei, C., Yang, L., Gao, X., Yao, S., Eberhart, R., and Shi, Y. (1998). Extracting rules from fuzzy neural network by particle swarm optimization, *Proc. IEEE International Conference on Evolutionary Computation,* Anchorage, Alaska, USA

Kennedy, J. (1997). The particle swarm: social adaptation of knowledge. *Proc. Intl. Conf. on Evolutionary Computation*, Indianapolis, IN, 303-308. Piscataway, NJ: IEEE Service Center.

Kennedy, J. (1998). Methods of agreement: inference among the eleMentals. *Proc. 1998 Intl. Symp. on Intelligent Control.* Piscataway, NJ: IEEE Service Center.

Kennedy, J. (1998). The behavior of particles. In V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, Eds. *Evolutionary Programming VII: Proc. 7th Ann. Conf. on Evolutionary Programming Conf.*, San Diego, CA, 581–589. Berlin: Springer-Verlag.

Kennedy, J. (1998). Thinking is social: experiments with the adaptive culture model. *Journal of Conflict Resolution.* 42(1), 56–76.

Kennedy, J. (1999). Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. *Proc. Congress on Evolutionary Computation 1999*, 1931–1938. Piscataway, NJ: IEEE Service Center.

Kennedy, J. (2000). Stereotyping: improving particle swarm performance with cluster analysis. *Proc. of the 2000 Congress on Evolutionary Computation*, San Diego, CA. Piscataway, NJ: IEEE Press.

Kennedy, J. (2001). Out of the computer, into the world: externalizing the particle swarm. *Proceedings of the Workshop on Particle Swarm Optimization.* Indianapolis, IN: Purdue School of Engineering and Technology, IUPUI (in press).

Kennedy, J. and Eberhart, R. C. (1995). Particle swarm optimization. *Proc. IEEE Int'l. Conf. on Neural Networks, IV*, 1942–1948. Piscataway, NJ: IEEE Service Center.

Kennedy, J. and Eberhart, R. C. (1997). A discrete binary version of the particle swarm algorithm. *Proc. 1997 Conf. on Systems, Man, and Cybernetics*, 4104–4109. Piscataway, NJ: IEEE Service Center.

Kennedy, J., and Eberhart, R. C. (1999). The particle swarm: social adaptation in information processing systems. In Corne, D., Dorigo, M., and Glover, F., Eds., *New Ideas in Optimization*. London: McGraw-Hill.

Kennedy, J., Eberhart, R. C., and Shi, Y. (2001). *Swarm Intelligence*, San Francisco: Morgan Kaufmann Publishers.

Kennedy, J. and Spears, W. M. (1998). Matching algorithms to problems: an experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator. *Proc. Intl. Conf. on Evolutionary Computation*, 78–83. Piscataway, NJ: IEEE Service Center.

Mohan, C. K., and Al-kazemi, B. (2001). Discrete particle swarm optimization. *Proceedings of the Workshop* *on Particle Swarm Optimization.* Indianapolis, IN: Purdue School of Engineering and Technology, IUPUI (in press).

Naka, S., Grenji, T., Yura, T., Fukuyama, Y. (2001). Practical Distribution State Estimation Using Hybrid Particle Swarm Optimization, *Proc. of IEEE PES Winter Meeting*, Columbus, Ohio, USA.

Ozcan, E., and Mohan, C. (1999). Particle swarm optimization: surfing the waves. *Proc. 1999 Congress on Evolutionary Computation*, 1939–1944. Piscataway, NJ: IEEE Service Center.

Parsopoulos, K. E., Plagianakos, V. P., Magoulas, G. D. and Vrahatis, M. N. (2001). Stretching technique for obtaining global minimizers through particle swarm optimization. *Proceedings of the Workshop on Particle Swarm Optimization.* Indianapolis, IN: Purdue School of Engineering and Technology, IUPUI (in press).

Secrest, B. R., and Lamont, G. B. (2001). Communication in particle swarm optimization illustrated by the traveling salesman problem. *Proceedings of the Workshop on Particle Swarm Optimization.* Indianapolis, IN: Purdue School of Engineering and Technology, IUPUI (in press).

Shi, Y. and Eberhart, R. C. (1998a). Parameter selection in particle swarm optimization. In *Evolutionary Programming VII: Proc. EP98*, New York: Springer-Verlag, pp. 591-600.

Shi, Y. and Eberhart, R. C. (1998b). A modified particle swarm optimizer. *Proceedings of the IEEE International Conference on Evolutionary Computation*, 69-73. Piscataway, NJ: IEEE Press.

Shi, Y. and Eberhart, R. C. (1999). Empirical study of particle swarm optimization. *Proceedings of the 1999 Congress on Evolutionary Computation*, 1945–1950. Piscataway, NJ: IEEE Service Center.

Shi, Y. and Eberhart, R., (2000). Experimental study of particle swarm optimization. *Proc. SCI2000 Conference*, Orlando, FL.

Shi, Y. and Eberhart, R., (2001). Fuzzy Adaptive Particle Swarm Optimization, *Proc. Congress on Evolutionary Computation 2001*, Seoul, Korea. Piscataway, NJ: IEEE Service Center. (in press)

Suganthan, P. N. (1999). Particle swarm optimiser with neighbourhood operator. *Proceedings of the 1999 Congress on Evolutionary Computation*, 1958–1962. Piscataway, NJ: IEEE Service Center.

Tandon, V. (2000). Closing the gap between CAD/CAM and optimized CNC end milling. Master's thesis, Purdue School of Engineering and Technology, Indiana University Purdue University Indianapolis.

Yoshida, H., Kawata, K., Fukuyama, Y., and Nakanishi, Y. (1999). A particle swarm optimization for reactive power and voltage control considering voltage stability. In G. L. Torres and A. P. Alves da Silva, Eds., *Proc. Intl. Conf. on Intelligent System Application to Power Systems*, Rio de Janeiro, Brazil, 117–121.