A New Optimizer Using Particle Swarm Theory

Russell Eberhart
Purdue School of Engineering and Technology
Indianapolis, IN 46202-5160
eberhart@engr.iupui.edu

James Kennedy
Bureau of Labor Statistics
Washington, DC 20212
kennedyj@po1.ocsp.bls.gov

ABSTRACT

The optimization of nonlinear functions using particle swarm methodology is described. Implementations of two paradigms are discussed and compared, including a recently developed locally oriented paradigm. Benchmark testing of both paradigms is described, and applications, including neural network training and robot task learning, are proposed. Relationships between particle swarm optimization and both artificial life and evolutionary computation are reviewed.

1. INTRODUCTION

A new method for optimization of continuous nonlinear functions was recently introduced [6]. This paper reviews the particle swarm optimization concept. Discussed next are two paradigms that implement the concept, one globally oriented (GBEST), and one locally oriented (LBEST), followed by results obtained from applications and tests upon which the paradigms have been shown to perform successfully.

Particle swarm optimization has roots in two main component methodologies. Perhaps more obvious are its ties to artificial life (A-life) in general, and to bird flocking, fish schooling, and swarming theory in particular. It is also related, however, to evolutionary computation, and has ties to both genetic algorithms and evolution strategies [1].

Particle swarm optimization comprises a very simple concept, and paradigms are implemented in a few lines of computer code. It requires only primitive mathematical operators, and is computationally inexpensive in terms of both memory requirements and speed. Early testing has found the implementation to be effective with several kinds of problems [6]. This paper discusses application of the algorithm to the training of artificial neural network weights. Particle swarm optimization has also been demonstrated to perform well on genetic algorithm test functions, and it appears to be a promising approach for robot task learning.

Particle swarm optimization can be used to solve many of the same kinds of problems as genetic algorithms (GAs) [6]. This optimization technique does not suffer, however, from some of GA's difficulties; interaction in the group enhances rather than detracts from progress toward the solution. Further, a particle swarm system has memory, which the genetic algorithm does not have. Change in genetic populations results in destruction of previous knowledge of the problem, except when elitism is employed, in which case usually one or a small number of individuals retain their "identities." In particle swarm optimization, individuals who fly past optima are tugged to return toward them; knowledge of good solutions is retained by all particles.

2. THE PARTICLE SWARM OPTIMIZATION CONCEPT

Particle swarm optimization is similar to a genetic algorithm [2] in that the system is initialized with a population of random solutions. It is unlike a genetic algorithm, however, in that each potential solution is also assigned a randomized velocity, and the potential solutions, called particles, are then "flown" through hyperspace.

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

Sixth International Symposium on
Micro Machine and Human Science
0-7803-2676-8/95 $4.00 ©1995 IEEE.

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

This paper introduces a "local" version of the optimizer in which, in addition to *pbest*, each particle keeps track of the best solution, called *lbest*, attained within a *local* topological neighborhood of particles. Both the global and local versions are described in more detail below.

The only variable that must be determined by the user is the maximum velocity to which the particles are limited. An acceleration constant is also specified, but in the experience of the authors, is not usually varied among applications.

3. TRAINING A MULTILAYER PERCEPTRON

The problem of finding a set of weights to minimize residuals in a feedforward neural network is not a trivial one. It is nonlinear and dynamic in that any change of one weight requires adjustment of many others. Gradient descent techniques, e.g., backpropagation of error, are usually used to find a matrix of weights that meets error criteria, though there is not widespread satisfaction with the effectiveness of these methods.

A number of researchers have attempted to use genetic algorithms (GAs) to find sets of weights, but the problem is not well suited to crossover. Because a large number of possible solutions exist, two chromosomes with high fitness evaluations are likely to be very different from one another; thus recombination may not result in improvement.

This discussion uses a three-layer network designed to solve the XOR problem, as a demonstration of the particle swarm optimization concept. The network has two inputs, three hidden processing elements (PEs), and one output PE. The output PE return a 1 if both inputs are the same, that is, input vector (1,1) or (0,0), and returns 0 if the inputs are different (1,0) or (0,1). Counting biases to the hidden and output PEs, solution of this problem requires estimation of 13 floating-point parameters. Note that, for the current presentation, the number of hidden units is arbitrary. A feedforward network with one or two hidden PEs can solve the XOR problem. Future research can test particle swarm optimization on a variety of architectures; the present paper necessarily, and arbitrarily, settled on one.

The particle swarm optimization approach is to "fly" a population of particles through 13-dimensional hyperspace. Each particle is initialized with position and velocity vectors of 13 elements. For neural networks, it seems reasonable to initialize all positional coordinates (corresponding to connection weights) to within a range of (-4.0, +4.0), and velocities should not be so high as to fly particles out of the usable field. It is also necessary to clamp velocities to some maximum to prevent overflow. The test examples use a population of 20 particles. (The authors have used populations of 10-50 particles for other applications.) The XOR data are entered into the net and an error term to be minimized, usually squared error per output PE, is computed for each of the 20 particles.

As the system iterates, individual agents are drawn toward a global optimum based on the interaction of their individual searches and the group's public search. Error threshold and maximum iteration termination criteria have been specified: when these are met, or when a key is pressed, iterations cease and the best weight vector found is written to a file.

3.1 The GBEST Model

The standard "GBEST" particle swarm algorithm, which is the original form of particle swarm optimization developed, is very simple. The steps are:

1. Initialize an array of particles with random positions and velocities on *D* dimensions,

2. Evaluate the desired minimization function in *D* variables,

3. Compare evaluation with particle's previous best value (PBEST[]): If current value < PBEST[] then PBEST[] = current value and PBESTx[][d] = current position in *D*-dimensional hyperspace,

4. Compare evaluation with group's previous best (PBEST[GBEST]): If current value < PBEST[GBEST] then GBEST=particle's array index,

5. Change velocity by the following formula:

```
V[][d] = V[][d] +
ACC_CONST*rand()*(PBESTx[][d] - PresentX[][d])
+
ACC_CONST*rand()*(PBESTx[GBEST][d] -
PresentX[][d]), and,
```

6. Move to PresentX[][d] + v[][d]: Loop to step 2 and repeat until a criterion is met.

3.2 The LBEST Version

Based, among other things, on findings from social simulations, it was decided to design a "local" version (paradigm) of the particle swarm concept. In this paradigm, particles have information only of their own and their nearest array neighbors' bests, rather than that of the entire group. Instead of moving toward the stochastic average of *pbest* and *gbest* (the best evaluation in the entire *group*), particles move toward the points defined by *pbest* and "*lbest*," which is the index of the particle with the best evaluation in the *neighborhood.* In the neighborhood=2 model, for instance, particle(i) compares its error value with particle(i-1) and particle(i+1). The *lbest* version was tested with neighborhoods consisting of the immediately adjacent neighbors (neighborhood=2), and with the three neighbors on each side (neighborhood=6).

Table 1 shows results of performance on the XOR neural-net problem with neighborhood=2. Note that no trials fixated on local optima- nor have any in hundreds of unreported tests.

Cluster analysis of sets of weights from this version showed that blocks of neighbors, consisting of regions from 2 to 8 adjacent individuals, had settled into the same regions of the solution space. It appears that the invulnerability of this version to local optima might result from the fact that a number of "groups" of particles spontaneously separate and explore different regions. It is thus a more flexible approach to information processing than the GBEST model.

Nonetheless, though this version rarely if ever becomes entrapped in a local optimum, it clearly requires more iterations on average to find a criterion error level. Table 2 represents tests of a LBEST version with neighborhood=6, that is, with the three neighbors on each side of the agent taken into account (arrays wrapped, so the final element was considered to be beside the first one).

This version is prone to local optima, at least when VMAX is small, though less so than the GBEST version. Otherwise it seems, in most cases, to perform somewhat less well than the standard GBEST algorithm.

In sum, the neighborhood=2 model offered some intriguing possibilities, in that is seems immune to local optima. It is a highly decentralized model, which could be run with any number of particles. Expanding the neighborhood speeds up convergence, but introduces the frailties of the GBEST model.

Table 1. Local version, neighborhood=2. Median iterations required to meet a criterion of squared error per node < 0.02. Population=20 particles. There were no trials with iterations > 2000.

VMAX	ACC_CONST		
	2.0	1.0	0.5
2.0	38.5	47	37.5
4.0	28.5	33	53.5
6.0	29.5	40.5	39.5

Table 2. Local version, neighborhood=6. Median iterations required to meet a criterion of squared error per node < 0.02. Population=20 particles.

VMAX	ACC_CONST		
	2.0	1.0	0.5
2.0	31.5(2)	38.5(1)	27(1)
4.0	36(1)	26	25
6.0	26.5	29	20

4. FLOCKS, SWARMS AND PARTICLES

A number of scientists have created computer simulations of various interpretations of the movement of organisms in a bird flock or fish school. Notably, Reynolds [10] and Heppner and Grenander [4] presented simulations of bird flocking.

It became obvious during the development of the particle swarm concept that the behavior of the population of agents is more like a swarm than a flock. The term *swarm* has a basis in the literature. In particular, the authors use the term in accordance with a paper by Millonas [7], who developed his models for applications in artificial life, and articulated five basic principles of swarm intelligence.

First is the proximity principle: the population should be able to carry out simple space and time computations. Second is the quality principle: the population should be able to respond to quality factors in the environment. Third is the principle of diverse response: the population should not commit its activities along excessively narrow channels. Fourth is the principle of stability: the population should not change its mode of behavior every time the environment changes. Fifth is the principle of adaptability: the population must be able to change behavior mode when it's worth the computational price. Note that principles four and five are the opposite sides of the same coin.

The particle swarm optimization concept and paradigm presented in this paper seem to adhere to all five principles. Basic to the paradigm are n-dimensional space calculations carried out over a series of time steps. The population is responding to the quality factors *pbest* and *gbest/lbest*. The allocation of responses between *pbest* and *gbest/lbest* ensures a diversity of response. The population changes its state (mode of behavior) only when *gbest/lbest* changes, thus adhering to the principle of stability. The population is adaptive because it *does* change when *gbest/lbest* changes.

The term *particle* was selected as a compromise. While it could be argued that the population members are mass-less and volume-less, and thus could be called "points," it is felt that velocities and accelerations are more appropriately applied to particles, even if each is defined to have arbitrarily small mass and volume. Further, Reeves [9] discusses *particle systems* consisting of clouds of primitive particles as models of diffuse objects such as clouds, fire and smoke. Thus the label the authors have chosen to represent the optimization concept is *particle swarm*.

5. TESTS AND EARLY APPLICATIONS OF THE OPTIMIZER

The paradigm has been tested using systematic benchmark tests as well as observing its performance on applications that are known to be difficult. The neural-net application described in *Section 3*, for instance, showed that the particle swarm optimizer could train NN weights as effectively as the usual error backpropagation method. The particle swarm optimizer has also been used to train a neural network to classify the Fisher Iris Data Set [3]. Again, the optimizer trained the weights as effectively as the backpropagation method. Over a series of ten training sessions, the particle swarm optimizer paradigm required an average of 284 epochs [6].

The particle swarm optimizer was compared to a benchmark for genetic algorithms in Davis [2]: the extremely nonlinear Schaffer f6 function. This function is very difficult to optimize, as the highly discontinuous data surface features many local optima. The particle swarm paradigm found the global optimum each run, and appears to approximate the results reported for elementary genetic algorithms in Chapter 2 of [2] in terms of the number of evaluations required to reach certain performance levels [6].

GAs have been used to learn complex behaviors characterized by sets of sequential decision rules. One approach uses *Cooperative Coevolutionary Genetic Algorithms* (CCGAs) to evolve sequential decision rules that control simulated robot behaviors [8]. The GA is used to evolve populations of rule sets, which are applied to problems involving multiple robots in competetive or cooperative tasks. Use of particle swarm optimization, currently being explored, instead of the GA, may enhance population evolution. For example, migration among sub-species of robots can be a problem due to GA crossover; this problem should not exist with particle swarms.

6 CONCLUSIONS

This paper introduces a new form of the particle swarm optimizer, examines how changes in the paradigm affect the number of iterations required to meet an error criterion, and the frequency with which models cycle interminably around a nonglobal optimum. Three versions were tested: the "GBEST" model, in which every agent has information about the group's best evaluation, and two variations of the "LBEST" version, one with a neighborhood of six, and one with a neighborhood of two. It appears that the original GBEST version performs best

in terms of median number of iterations to convergence, while the LBEST version with a neighborhood of two is most resistant to local minima.

Particle swarm optimization is an extremely simple algorithm that seems to be effective for optimizing a wide range of functions. We view it as a mid-level form of A-life or biologically derived algorithm, occupying the space in nature between evolutionary search, which requires eons, and neural processing, which occurs on the order of milliseconds. Social optimization occurs in the time frame of ordinary experience — in fact, it *is* ordinary experience. In addition to its ties with A-life, particle swarm optimization has obvious ties with evolutionary computation. Conceptually, it seems to lie somewhere between genetic algorithms and evolutionary programming. It is highly dependent on stochastic processes, like evolutionary programming. The adjustment toward *pbest* and *gbest* by the particle swarm optimizer is conceptually similar to the *crossover* operation utilized by genetic algorithms. It uses the concept of *fitness*, as do all evolutionary computation paradigms.

Unique to the concept of particle swarm optimization is flying potential solutions through hyperspace, accelerating toward "better" solutions. Other evolutionary computation schemes operate directly on potential solutions which are represented as locations in hyperspace. Much of the success of particle swarms seems to lie in the agents' tendency to hurtle past their target. Holland's chapter on the "optimum allocation of trials" [5] reveals the delicate balance between conservative testing of known regions versus risky exploration of the unknown. It appears that the current version of the paradigm allocates trials nearly optimally. The stochastic factors allow thorough search of spaces between regions that have been found to be relatively good, and the momentum effect caused by modifying the extant velocities rather than replacing them results in overshooting, or exploration of unknown regions of the problem domain.

Much further research remains to be conducted on this simple new concept and paradigm. The goals in developing it have been to keep it simple and robust, and we seem to have succeeded at that. The algorithm is written in a very few lines of code, and requires only specification of the problem and a few parameters in order to solve it.

ACKNOWLEDGMENT

Portions of this paper are adapted from a chapter on particle swarm optimization in a book entitled Computational Intelligence PC Tools, to be published in early 1996 by Academic Press Professional (APP). The permission of APP to include this material is gratefully acknowledged.

REFERENCES

[1] T. Baeck, "Generalized convergence models for tournament and (mu,lambda)-selection." *Proc. of the Sixth International Conf. on Genetic Algorithms*, pp. 2-7, Morgan Kaufmann Publishers, San Francisco, CA, 1995.

[2] L. Davis, Ed., *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, NY, 1991.

[3] R. A. Fisher, "The use of multiple measurements in taxonomic problems." *Annals of Eugenics*, 7:179–188, 1936.

[4] F. Heppner and U. Grenander, "A stochastic nonlinear model for coordinated bird flocks." In S. Krasner, Ed., *The Ubiquity of Chaos*, AAAS Publications, Washington, DC, 1990.

[5] J. H. Holland, *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge, MA., 1992.

[6] J. Kennedy and R. Eberhart, "Particle swarm optimization." Proc. IEEE International Conf. on Neural Networks (Perth, Australia), IEEE Service Center, Piscataway, NJ, 1995 (in press).

[7] M. Millonas, "Swarms, phase transitions, and collective intelligence." In C. G. Langton, Ed., *Artificial Life III*, Addison Wesley, Reading, MA, 1994.

[8] M. Potter, K. De Jong, and J. Grefenstette, "A coevolutionary approach to learning sequential decision rules." *Proc. of the Sixth International Conf. on Genetic Algorithms*, pp. 366-372, Morgan Kaufmann Publishers, San Francisco, CA, 1995.

[9] W. T. Reeves, "Particle systems - a technique for modeling a class of fuzzy objects." *ACM Transactions on Graphics*, 2(2):91–108, 1983.

[10] C. W. Reynolds, "Flocks, herds and schools: a distributed behavioral model." *Computer Graphics*, 21(4):25–34, 1987.