Particle Swarm Optimization

James Kennedy[1] and Russell Eberhart[2]

[1]Washington, DC 20212
kennedy_jim@bls.gov

[2]Purdue School of Engineering and Technology
Indianapolis, IN 46202-5160
eberhart@engr.iupui.edu

ABSTRACT
A concept for the optimization of nonlinear functions using particle swarm methodology is introduced. The evolution of several paradigms is outlined, and an implementation of one of the paradigms is discussed. Benchmark testing of the paradigm is described, and applications, including nonlinear function optimization and neural network training, are proposed. The relationships between particle swarm optimization and both artificial life and genetic algorithms are described.

1 INTRODUCTION

This paper introduces a method for optimization of continuous nonlinear functions. The method was discovered through simulation of a simplified social model; thus the social metaphor is discussed, though the algorithm stands without metaphorical support. This paper describes the particle swarm optimization concept in terms of its precursors, briefly reviewing the stages of its development from social simulation to optimizer. Discussed next are a few paradigms that implement the concept. Finally, the implementation of one paradigm is discussed in more detail, followed by results obtained from applications and tests upon which the paradigm has 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 evolutionary programming. These relationships are briefly reviewed in the paper.

Particle swarm optimization as developed by the authors comprises a very simple concept, and paradigms can be 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. 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. This paper discusses the performance on Schaffer's f6 function, as described in Davis [1].

2 SIMULATING SOCIAL BEHAVIOR

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 [8] and Heppner and Grenander [4] presented simulations of bird flocking. Reynolds was intrigued by the aesthetics of bird flocking choreography, and Heppner, a zoologist, was interested in discovering the underlying rules that enabled large numbers of birds to flock synchronously, often changing direction suddenly, scattering and regrouping, etc. Both of these scientists had the insight that local processes, such as those modeled by cellular automata, might underlie the unpredictable group dynamics of bird social behavior. Both models relied heavily on manipulation of inter-individual distances; that is, the synchrony of flocking behavior was thought to be a function of birds' efforts to maintain an optimum distance between themselves and their neighbors.

It does not seem a too-large leap of logic to suppose that some same rules underlie animal social behavior, including herds, schools, and flocks, and that of humans. As sociobiologist E. O. Wilson [9] has written, in reference to fish schooling, "In theory at least, individual members of the school can profit from the discoveries and previous experience of all other members of the school during the search for food. This advantage can become decisive, outweighing the disadvantages of competition for food items, whenever the resource is unpredictably distributed in patches" (p.209). This statement suggests that social sharing of information among conspeciates offers an evolutionary advantage: this hypothesis was fundamental to the development of particle swarm optimization.

One motive for developing the simulation was to model human social behavior, which is of course not identical to fish schooling or bird flocking. One important difference is its abstractness. Birds and fish adjust their physical movement to avoid predators, seek food and mates, optimize environmental parameters such as temperature, etc. Humans adjust not only physical movement but cognitive or experiential variables as well. We do not usually walk in step and turn in unison (though some fascinating research in human conformity shows that we are capable of it); rather, we tend to adjust our beliefs and attitudes to conform with those of our social peers.

This is a major distinction in terms of contriving a computer simulation, for at least one obvious reason: collision. Two individuals can hold identical attitudes and beliefs without banging together, but two birds cannot occupy the same position in space without colliding. It seems reasonable, in discussing human social behavior, to map the concept of *change* into the bird/fish analog of *movement*. This is consistent with the classic Aristotelian view of qualitative and quantitative change as types of movement. Thus, besides moving through three-dimensional physical space, and avoiding collisions, humans change in abstract multidimensional space, collision-free. Physical space of course affects informational inputs, but it is arguably a trivial component of psychological experience. Humans learn to avoid physical collision by an early age, but navigation of *n*-dimensional psychosocial space requires decades of practice — and many of us never seem to acquire quite all the skills we need!

3 PRECURSORS: THE ETIOLOGY OF PARTICLE SWARM OPTIMIZATION

The particle swarm optimizer is probably best presented by explaining its conceptual development. As mentioned above, the algorithm began as a simulation of a simplified social milieu. Agents were thought of as collision-proof birds, and the original intent was to graphically simulate the graceful but unpredictable choreography of a bird flock.

3.1 Nearest Neighbor Velocity Matching and Craziness

A satisfying simulation was rather quickly written, which relied on two props: nearest-neighbor velocity matching and "craziness." A population of birds was randomly initialized with a position for each on a torus pixel grid and with X and Y velocities. At each iteration a loop in the program determined, for each agent (a more appropriate term than bird), which other agent was its nearest neighbor, then assigned that agent's X and Y velocities to the agent in focus. Essentially this simple rule created a synchrony of movement.

Unfortunately, the flock quickly settled on a unanimous, unchanging direction. Therefore, a stochastic variable called *craziness* was introduced. At each iteration some change was added to randomly chosen *X* and *Y* velocities. This introduced enough variation into the system to give the simulation an interesting and "lifelike" appearance, though of course the variation was wholly artificial.

3.2 The Cornfield Vector

Heppner's bird simulations had a feature which introduced a dynamic force into the simulation. His birds flocked around a "roost," a position on the pixel screen that attracted them until they finally landed there. This eliminated the need for a variable like craziness, as the simulation took on a life of its own. While the idea of a roost was intriguing, it led to another question which seemed even more stimulating. Heppner's birds knew where their roost was, but in real life birds land on any tree or telephone wire that meets their immediate needs. Even more importantly, bird flocks land where there is food. How do they find food? Anyone who has ever put out a bird feeder knows that within hours a great number of birds will likely find it, even though they had no previous knowledge of its location, appearance, etc. It seems possible that something about the flock dynamic enables members of the flock to capitalize on one another's knowledge, as in Wilson's quote above.

The second variation of the simulation defined a "cornfield vector," a two-dimensional vector of *XY* coordinates on the pixel plane. Each agent was programmed to evaluate its present position in terms of the equation:

$$Eval = \sqrt{(presentx - 100)^2} + \sqrt{(presenty - 100)^2}$$

so that at the (100,100) position the value was zero.

Each agent "remembered" the best value and the *XY* position which had resulted in that value. The value was called *pbest[]* and the positions *pbestx[]* and *pbesty[]* (brackets indicate that these are arrays, with number of elements = number of agents). As each agent moved through the pixel space evaluating positions, its *X* and *Y* velocities were adjusted in a simple manner. If it was to the right of its *pbestx*, then its *X* velocity (call it *vx*) was adjusted negatively by a random amount weighted by a parameter of the system: *vx[]=vx[] - rand()*p_increment*. If it was to the left of *pbestx*, *rand()*p_increment* was added to *vx[]*. Similarly, *Y* velocities *vy[]* were adjusted up and down, depending on whether the agent was above or below *pbesty*.

Secondly, each agent "knew" the globally best position that one member of the flock had found, and its value. This was accomplished by simply assigning the array index of the agent with the best value to a variable called *gbest*, so that *pbestx[gbest]* was the group's best *X* position, and *pbesty[gbest]* its best *Y* position, and this information was available to all flock members. Again, each member's *vx[]* and *vy[]* were adjusted as follows, where *g_increment* is a system parameter.

```
if presentx[] > pbestx[gbest] then vx[] = vx[] - rand()*g_increment
if presentx[] < pbestx[gbest] then vx[] = vx[] + rand()*g_increment
if presenty[] > pbesty[gbest] then vy[] = vy[] - rand()*g_increment
if presenty[] < pbesty[gbest] then vy[] = vy[] + rand()*g_increment
```

In the simulation, a circle marked the (100,100) position on the pixel field, and agents were represented as colored points. Thus an observer could watch the flocking agents circle around until they found the simulated cornfield. The results were surprising. With *p_increment* and *g_increment* set relatively high, the flock seemed to be sucked violently into the cornfield. In a very few iterations the entire flock, usually 15 to 30 individuals, was seen to be clustered within the tiny circle surrounding the goal. With *p_increment* and *g_increment* set low, the flock swirled around the goal, realistically approaching it, swinging out rhythmically with subgroups synchronized, and finally "landing" on the target.

3.3 Eliminating Ancillary Variables

Once it was clear that the paradigm could optimize simple, two-dimensional, linear functions, it was important to identify the parts of the paradigm that are necessary for the task. For instance, the authors quickly found that the algorithm works just as well, and looks just as realistic, without craziness, so it was removed. Next it was shown that optimization actually occurs slightly faster when nearest neighbor velocity matching is removed, though the visual effect is changed. The *flock* is now a *swarm*, but it is well able to find the cornfield.

The variables *pbest* and *gbest* and their *increments* are both necessary. Conceptually *pbest* resembles autobiographical memory, as each individual remembers its own experience (though only one fact about it), and the velocity adjustment associated with *pbest* has been called "simple nostalgia" in that the individual tends to return to the place that most satisfied it in the past. On the other hand, *gbest* is conceptually similar to publicized knowledge, or a group norm or standard, which individuals seek to attain. In the simulations, a high value of *p_increment* relative to *g_increment* results in excessive wandering of isolated individuals through the problem space, while the reverse (relatively high *g_increment*) results in the flock rushing prematurely toward local minima. Approximately equal values of the two *increments* seem to result in the most effective search of the problem domain.

3.4 Multidimensional Search

While the algorithm seems to impressively model a flock searching for a cornfield, most interesting optimization problems are neither linear nor two-dimensional. Since one of the authors' objectives is to model social behavior, which is multidimensional and collision-free, it seemed a simple step to change *presentx* and *presenty* (and of course *vx[]* and *vy[]*) from one-dimensional arrays to $D \times N$ matrices, where D is any number of dimensions and N is the number of agents.

Multidimensional experiments were performed, using a nonlinear, multidimensional problem: adjusting weights to train a feedforward multilayer perceptron neural network (NN). One of the authors' first experiments involved training weights for a three-layer NN solving the exclusive-or (XOR) problem. This problem requires two input and one output processing elements (PEs), plus some number of hidden PEs. Besides connections from the previous layer, the hidden and output PE layers each has a bias PE associated with it. Thus a 2,3,1 NN requires optimization of 13 parameters. This problem was approached by flying the agents through 13-dimensional space until an average sum-squared error per PE criterion was met. The algorithm performed very well on this problem. The thirteen-dimensional XOR network was trained, to an $e < 0.05$ criterion, in an average of 30.7 iterations with 20 agents. More complex NN architectures took longer of course, but results, discussed in *Section 5: Results and Early Applications,* were still very good.

3.5 Acceleration by Distance

Though the algorithm worked well, there was something aesthetically displeasing and hard to understand about it. Velocity adjustments were based on a crude inequality test: if *presentx* > *bestx*, make it smaller; if *presentx* < *bestx*, make it bigger. Some experimentation revealed that further revising the algorithm made it easier to understand and improved its performance. Rather than simply testing the sign of the inequality, velocities were adjusted according to their difference, per dimension, from best locations:

*vx[][] = vx[][] + rand()*p_increment*(pbestx[][] - presentx[][])*

(note the parameters *vx* and *presentx* have two sets of brackets because they are now matrices of agents by dimensions; *increment* and *bestx* could also have a *g* instead of *p* at their beginnings.)

3.6 Current Simplified Version

It was soon realized that there is no good way to guess whether *p-* or *g-increment* should be larger. Thus, these terms were also stripped out of the algorithm. The stochastic factor was multiplied by 2 to give it a mean of 1, so that agents would "overfly" the target about half the time. This version outperforms the previous versions. Further research will show whether there is an optimum value for the constant currently set at 2, whether the value should be evolved for each problem, or whether the value can be determined from some knowledge of a particular problem. The current simplified particle swarm optimizer now adjusts velocities by the following formula:

```
vx[][] = vx[][] +
        2 * rand() * (pbestx[][] - presentx[][]) +
        2 * rand() * (pbestx[][gbest] - presentx[][])
```

3.7 Other Experiments

Other variations on the algorithm were tried, but none seemed to improve on the current simplified version. For instance, it is apparent that the agent is propelled toward a weighted average of the two "best" points in the problem space. One version of the algorithm reduced the two terms to one, which was the point on each dimension midway between *pbest* and *gbest* positions. This version had an unfortunate tendency, however, to converge on that point whether it was an optimum or not. Apparently the two stochastic "kicks" are a necessary part of the process.

Another version considered using two types of agents, conceived as "explorers" and "settlers." Explorers used the inequality test, which tended to cause them to overrun the target by a large distance, while settlers used the difference term. The hypothesis was that explorers would extrapolate outside the "known" region of the problem domain, and the settlers would hill-climb or micro-explore regions that had been found to be good. Again, this method showed no improvement over the current simplified version. Occam's razor slashed again.

Another version that was tested removed the momentum of *vx[][]*. The new adjustment was:

```
vx[][] =        2 * rand() * (pbestx[][] - presentx[][] ) +
                2 * rand() * (pbestx[][gbest] - presentx[][] )
```

This version, though simplified, turned out to be quite ineffective at finding global optima.

4 SWARMS AND PARTICLES

As was described in *Section 3.3*, it became obvious during the simplification of the paradigm that the behavior of the population of agents is now 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 [6], 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*. The allocation of responses between *pbest* and *gbest* ensures a diversity of response. The population changes its state (mode of behavior) only when *gbest* changes, thus adhering to the principle of stability. The population is adaptive because it *does* change when *gbest* 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 [7] 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 3.4*, 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.

Intriguing informal indications are that the trained weights found by particle swarms sometimes generalize from a training set to a test set better than solutions found by gradient descent. For example, on a data set representing electroencephalogram spike waveforms and false positives, a backpropagation NN achieved 89 percent correct on the test data [2]. The particle swarm optimizer was able to train the network so as to achieve 92 percent correct.

The particle swarm optimizer was compared to a benchmark for genetic algorithms in Davis [1]: 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 [1] in terms of the number of evaluations required to reach certain performance levels.

6 CONCLUSIONS

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.

The authors of this paper are a social psychologist and an electrical engineer. The particle swarm optimizer serves both of these fields equally well. Why is social behavior so ubiquitous in the animal kingdom? Because it optimizes. What is a good way to solve engineering optimization problems? Modeling social behavior.

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. This algorithm belongs ideologically to that philosophical school that allows wisdom to emerge rather than trying to impose it, that emulates nature rather than trying to control it, and that seeks to make things simpler rather than more complex. Once again nature has provided us with a technique for processing information that is at once elegant and versatile.

ACKNOWLEDGMENTS

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. The input and comments of Roy Dobbins and Pat Simpson are appreciated.

REFERENCES

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

[2] Eberhart, R. C. and R. W Dobbins (1990). *Neural Network PC Tools: A Practical Guide.* Academic Press, San Diego, CA.

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

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

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

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

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

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

[9] Wilson, E.O. (1975). *Sociobiology: The new synthesis.* Belknap Press, Cambridge, MA.