Proceedings of the 2007 IEEE Swarm Intelligence Symposium (SIS 2007)

Some Issues and Practices for Particle Swarms

James Kennedy
Bureau of Labor Statistics
Washington, DC
Kennedy_Jim@bls.gov

Abstract: As the particle swarm paradigm matures, some bad habits are adopted and some good practices are ignored. This paper gives an informal discussion of some of issues and practices that may affect the course of future development of the algorithm.

In the more than ten years since the particle swarm algorithm was first described, researchers have devised literally hundreds of modifications to improve it. Almost every paper reports that its new version performs better than previous ones. Some of the modifications are quite helpful, and yet very few of the innovations are adopted by the wider community. At the same time, some methods in widespread use are neither efficient nor effective.

The present paper informally notes a few instances where bad habits can be broken, and where performance of the particle swarm can be improved with little or no extra effort. As always in this paradigm, parsimony is considered a virtue; a simpler algorithm that performs as well as a more complicated one will be preferred.

Constriction and Inertia

Since Shi and Eberhart (1999) and Clerc and Kennedy (2002) it has been customary to use a set of coefficients on the various terms of the velocity formula. Shi and Eberhart's inertia weight method is described as:

$$v_{id} = W1*v_{id} + rand()*W2*(p_{id} - x_{id}) + rand()*W2*(p_{gd} - x_{id})$$
$$x_{id} = x_{id} + v_{id}$$

where *W1* is the inertia weight; v_{id} is the velocity of individual *i* on dimension *d*; p_{id} is the location of the best problem solution vector found by *i*; x_{id} is *i*'s current position on dimension *d*; and *g* is the index of *i*'s best neighbor. *W2* has been called, since the earliest papers, the acceleration coefficient. Though some researchers find reasons to make one or the other bigger, in general use they are assigned the same value.

The inertia weight is usually less than 1.0. While the first term on the right-hand side of the velocity formula tells the particle to keep going in the direction it moved on the previous time-step, the function of the inertia weight is to attenuate or control that tendency. Various strategies have been introduced, such as varying the inertia weight over time (Shi and Eberhart, 1998), using random values constrained to some range (Eberhart and Shi, 2001), etc.

The Type 1" constriction coefficient (Clerc and Kennedy, 2002) weights the entire right-hand side of the velocity equation:

$$v_{id} = W1*(v_{id} + rand1*W2*(p_{id} - x_{id}) + rand2*W2*(p_{gd} - x_{id}))$$
$$x_{id} = x_{id} + v_{id}$$

Clerc's analysis found that the values to be used for *W1* – now called the constriction coefficient – and *W2* needed to satisfy some constraints in order for the particle's trajectory to converge. Infinite pairs of values satisfied the constraints, but experience showed that some values produced better optimization results than others. In particular, Clerc and Kennedy (2002) recommended using a value of 4.1 for the sum of the acceleration constants, which results in a constriction coefficient value $W1 \approx 0.7298$ and $W2 = 2.05$.

Note that the inertia weight and constriction coefficient variations are algebraically equivalent, if one multiplies the constriction coefficient through the parentheses. The notation gets a little bulky, however:

$$v_{id} = W1*v_{id} + rand1*W1*W2*(p_{id} - x_{id}) + rand2*W1*W2*(p_{gd} - x_{id})$$
$$x_{id} = x_{id} + vi_{id}$$

This paper will use the inertia-weight form, and use Greek letters alpha and beta for the two types of coefficients. Also, since the value used for W2 depends on the sum of the acceleration constants, beta will stand for that sum; thus, each weight in the standard algorithm, with two difference terms, will be given as (β/2):

$$v_{id} = \alpha*v_{id} + rand1*(\beta/2)*(p_{id} - x_{id}) + rand2*(\beta/2)*(p_{gd} - x_{id})$$
$$x_{id} = x_{id} + v_{id}$$

1-4244-0708-7/07/$20.00 ©2007 IEEE

Note that there are no parameters to adjust once this decision has been made. While it may be possible to find slightly better values for either or all coefficients for a particular problem, a version where α=0.7298 and β~2.992 performs well on a wide range of problem types.

Vmax

The earliest versions of particle swarms, before inertia weights and constriction coefficients had been devised, required the use of a constant called Vmax, which limited the velocity of the particle. Without Vmax, the particle simply exploded, velocity increased toward infinity and the particle's trajectory careened wildly and uselessly. Vmax prevented this explosion.

if $v_{id} > Vmax$ then $v_{id}=Vmax$
else if $v_{id} < -Vmax$ then $v_{id}=-Vmax$

Though the particle remained within the predefined search space, the Vmax version without an inertia/constriction factor had no tendency to converge. As an optimal region was discovered, the particle's trajectory might reduce to a certain range of step-sizes, but there was nothing to make it approach a limit of zero velocity as particles converged. If the value of Vmax was set too low, the particle could find itself unable to escape local optima, yet as the value became higher it lost its ability to perform local search very well; values had to be tailored to particular problems. Thus, it was mostly regarded as a necessary evil, as Vmax particle swarms performed notably well but not as well as anyone would have liked.

With the application of coefficients to the first right-hand term of the velocity formula, Vmax became unnecessary. Explosion is prevented by reducing the effect of the previous iteration's velocity, in concert with appropriate acceleration constants. On the other hand, Eberhart and Shi (2000) noted that it was still helpful to use a very liberal Vmax, for instance limiting velocity to the initialization range of the variables or some large fraction of that, in order to prevent the particles flying into "outer space," as they put it.

Thus, at this time, Vmax is an option and not a necessity. It may propitiate convergence by disallowing entirely absurd problem solutions to be proposed, but if it is not applied, the system will still tend to behave well enough. Clamping the velocity when it exceeds a pathological value is generally not a very intrusive technique, as it will only occur rarely, and it does reduce the number of wasted iterations.

The Random Numbers

Wilke (2005) asserted in his dissertation that there is ambiguity in the application of the random numbers in the particle swarm, and that various researchers are using them differently. He wrote:

> It is unclear whether the random numbers are scalar numbers which simply scale the *magnitude* of the cognitive and social components of learning, or whether the random numbers are vectors that scale *each component* of the cognitive and social components of learning.

(In this quote, the "cognitive component" is the ($p_{id} - x_{id}$) term, and the "social component" is the ($p_{gd} - x_{id}$) term.) Wilke listed some references to papers where he believed that the authors had used the same set of random numbers for all dimensions of a particle on a single iteration. Efforts by the present author and another researcher to contact the six researchers named in Wilke's report resulted in only two responses; both researchers reported that they had correctly refreshed the random numbers each time they occurred, or in Wilke's terminology, had scaled each component.

It is possible that the other papers cited did make the error attributed to them by Wilke. At the very least, all of the authors were ambiguous in their algebraic notation, as have been many other writers in the field, the present author included.

In any case, the fact is that every random number should be uniquely generated. There will be a unique random number for each source of influence for every single dimension for every single particle for every single iteration.

It appears that the real issue is the correct way to write algebra or pseudocode for the random numbers; if researchers are using the random numbers incorrectly, it is because the papers they are studying are not clear about it. The rest of the formulas occur within nested loops for iteration, individual, and dimension, and there is a tendency not to properly annotate the fact that there are two unique random numbers nested within those loops. Further, some researchers write it as a variable and some write it as a function, which makes it even more difficult to decipher.

Notation

We can simplify the notation of the algorithm quite a bit, which will help us understand how it works and suggest innovations. First, adding time notation to the last line of the formula, $x_{id}(t+1)=x_{id}(t)+v_{id}(t+1)$, we see by rearranging that $v_{id}(t+1) = x_{id}(t+1) - x_{id}(t)$. Since the same thing happened on the previous time-step, we can infer that $v_{id}(t)=x_{id}(t) - x_{id}(t-1)$. This notation can replace the first reference to v_{id} on the right-hand side of the formula, collapsing the two formulas into one:

$$x_{id}(t+1)= x_{id}(t)+ \alpha*(x_{id}(t)-x_{id}(t-1)) + rand1()*(\beta/2)*(p_{id} - x_{id}(t)) + rand2()*(\beta/2)*(p_{gd} - x_{id}(t))$$

Further, since the last two terms are identical in form, we can simplify the notation by combining them in a summation:

$$x_{id} = x_{id} + \alpha*(x_{id}(t)-x_{id}(t-1)) + \sum(rand()_{idk}*(\beta/K)*(p_{kd} - x_{id}))$$

Where *k* represents the indexes of a source of influence – which will be *i* and *g* in the best-neighbor version that has been discussed thus far, and *K* represents the number of sources of influence on particle *i* (*K*=2 in the canonical version).

With this notation, we can easily see that, at each time-step, the particle

1. begins at its current location
2. moves some amount in the same direction it was going, and
3. moves in the direction of a randomly-weighted average of its sources of influence from the initial position x_{id}

The Population Best

The index *g* is used to refer to the member of *i*'s neighborhood that has attained the best result so far. If the population is completely connected in what is called the *gbest* topology, then *g* will be the best member of the entire population. This usage means that every particle has access to the best information that has been discovered by any member of the swarm so far. While this elitism is one source of bias toward better solutions, it also weakens the population in some ways.

When all particles are influenced by the same source, the swarm has little ability to explore in parallel. Some diversity is introduced through the specific search histories of each individual, that is, each one is affected by its own unique $\vec{p}_i$ vector, but this is neutralized by the fact that they are all searching a region half-defined by a single source of influence. As all particles receive the same information about successful problem solutions, they all tend to go to the same place. Thus the effect of the *gbest* topology is to weaken the swarm's ability to navigate around spaces with local optima; whatever region first produces a good result will become an attractor for the entire population, and potentially better locations will never be investigated.

There is also a problem with no known elegant solution, for the particle *g* which has the best function result so far. Its two difference terms are the same, as *i*=*g*. Thus it tends only to oscillate around its own previous success. This is not especially damaging to the swarm's performance, since it affects only one particle in the population at a time, but as that particle is essentially searching without external influence, it does not have much chance of improving. Since $p_{id}=p_{gd}$ in this situation, and the variance of the trajectory is scaled to the difference $|(p_{id}-p_{gd})|$, if no better point is found by another particle, this best one's velocity will eventually approach 0.0 and it will stop moving, though it will continue to be evaluated. Various researchers have published extrinsic ways to cope with this issue; the best thing may be to prevent it in the first place, by avoiding the *gbest* topology.

The worst problem with the *gbest* variation is its tendency to premature convergence. Numerous papers have been written, suggesting ways to repulse particles, reinitialize them, and adjust them in other ways in order to offset the tendency of the *gbest* population to converge prematurely. In fact, the convergence rate is largely a property of the population topology, and is easily controlled through more attentive appreciation of the way solutions propagate through the swarm. A judicious choice of topological structure can improve performance without increasing the complexity of the algorithm.

For simple monotonic functions, the *gbest* configuration works fast. But of course, if a researcher only needs a hill-climber, it should not be necessary to use a stochastic, population-based method like this at all. Further, other more versatile topologies will work perfectly well, if not quite as fast, on monotonic functions. Therefore it is recommended that investigators use the *gbest* population structure only as a last resort unless there is some special reason.

Other Topologies For Best-Neighbor Swarms

The traditional alternative to the *gbest* topology has been one called *lbest*. This is a ring lattice, with each individual particle connected to its *k* nearest neighbors in the population array. Where *K*=2, for instance, each particle *i* communicates only with its immediate neighbors, *i*-1 and *i*+1. Because the mean distance between particles is minimal in the *gbest* and maximal in the *lbest* topology, these two are sometimes considered as opposites.

The *lbest* topology slows the propagation of solutions through the swarm population. In the best-neighbor particle swarm versions, influence can only spread when a particle becomes the "best neighbor" to its neighbor. When *j* is the best neighbor of *i*, then *i* searches around a location partly defined by *j*'s best success so far. If *i* finds a problem solution in that region that is better than its own previous best, then it will store that location in its previous best vector $\vec{p}_i$ and subsequent iterations will be affected by that discovery, at least until something better is found. If, in *lbest*, *i* becomes the best neighbor to its other neighbor, then that neighbor too will be attracted to that region, and eventually the same series of events may unfold. Thus, a good region of the search space will attract one particle after the other, around the population, adding adherents slowly. (Note that we can think of this as if new problems solutions were passed through a stable population, or a dynamic population passed through a stable search space.)

The *lbest* topology is good at exploring multiple regions in parallel, and so is good for functions with local optima. One subset of neighbors can explore one region of the search

space while another subset explores somewhere else. Eventually, whichever region returns the best function results will win out; chances are, this will be the region containing the superior optimum (with the usual caveats about there being no guarantees). Thus the *lbest* topology increases the probability of finding the global optimum, where a *gbest* population will likely converge on the first decent region it finds.

There are innumerable ways to structure the communications among the particles. The fact is, very little effort is required in writing code to implement a good topology, and as the assignment is only run at the start of a trial, very little CPU time is dedicated to the task. Some memory will be used to store the vector of neighbors for each individual, otherwise computational costs are negligible.

Some topologies, such as *lbest*, are geometrically simple, and can be coded algorithmically. Another example of such a topology is the "square" or von Neumann structure, where the population is conceptualized as a rectangle, and each particle is connected to the ones above, below, and on either side of it. The population is wrapped, so top and bottom are adjacent, as are left and right edges. Each particle communicates with four neighbors, and solutions can propagate out in all those directions. Thus this structure has some of the parallelism of *lbest*, as distant neighborhoods can explore independently, yet it has more communication channels, so better ideas can spread more rapidly.

Custom topologies can be easily implemented using incidence matrices. In its simplest form, this is an $N \times N$ matrix, initialized with zeroes, with a one in the jth column of row i indicating that i is influenced by the particle with index j. If all communications are reciprocal, e.g., j is also influenced by i, then only half of a diagonally-split matrix is needed.

The question of which topology is best, and what makes a good topology, is one that has not been satisfactorily answered (Mendes, et al., 2004). Several researchers (e.g., Clerc, 2006; Liang and Suganthan 2005; Mohais, et al., 2005, etc.) have experimented with adaptive topologies, randomizing, adding, deleting, and moving links in response to aspects of the iterative search, and it seems very likely that the ultimate particle swarm will adapt its social network depending on the situation. Various rules have been proposed for edge adaptation, and some of them perform very well. So far it is not obvious which ones will prove successful. We can imagine topologies where communication is probabilistic, fuzzy, where links are impermanent, unidirectional, or weighted by some factor. The question is still an open one.

FIPS

In the traditional particle swarm the particle receives information from its best neighbor and itself; thus the acceleration constant is divided between two terms. The constants however can be partitioned any number of ways, as long as the sum is correct.

Mendes (2004) developed a type of particle swarm where each particle used information from all its neighbors, and not from its own history. Thus a particle with K neighbors sums K (p-x) differences, each weighted by $rand()*(\beta/K)$. This version, called the Fully Informed Particle Swarm, or FIPS, was found to perform better than the traditional version on an aggregated suite of various test functions, when an appropriate topology was used.

FIPS has some surprising implications about how particle swarms work. Whereas it was noted above that the positive bias derived from using the population best is unnecessary, and may even be detrimental to performance under some conditions, FIPS suggests that it is not necessary to select the best neighbor at all. Further, the particle swarm does not require that a particle be attracted toward its own previous successes – all influence in FIPS comes from external sources.

The pattern of propagation of problem solutions through the population topology is quite different in FIPS. Now a particle need not become its neighbor's best neighbor in order to influence it, as the FIPS particle searches around the mean of all its neighbors' previous best positions in the search space. The random numbers determine whether the particle will search nearer to one neighbor's success or another's, but the general trend drives it toward the centroid.

The topology takes on a different meaning in FIPS, and structures that produce good performance in the best-neighbor varieties will not be expected to do the same in this version. For instance, where the *gbest* topology informs each particle in the best-neighbor versions of the exact location of the best solution found thus far, in FIPS *gbest* introduces a tendency to search around the average of *all* previous-best locations. Because each influence is randomly weighted, the FIPS particle can be attracted toward any or all members of the population, resulting in behavior that is little better than random.

Mendes (2004) found that FIPS performed best with topologies where the mean degree (number of neighbors) was between 4 and 4.25. In the von Neumann configuration, K=4 for every particle, and it has been found to be a good topology for implementing FIPS.

FIPS is still new, and there is more to learn about how to get the most out of it. Researchers will be exploring better implementations of it, both in theory and applications.

Research Methods I

The standard research methodology for studying the properties of optimization algorithms, at least under the evolutionary computation umbrella, has been to optimize a suite of test functions using several competing algorithms and to compare the results. The test suite varies somewhat from one study to the next, but a small number of functions appear in almost all of them. It is common to optimize functions in 30 dimensions where that is possible.

Suganthan, et al. (2005) published a recommendation containing a new test suite of 25 functions. Those authors were concerned that the standard test functions often have

their global optimum at the origin, and that almost none of the functions were rotated, that is, gradients aligned with Cartesian axes, among other things. While several papers have come out recently using the CEC05 test suite, as it is called, there are some difficulties. Most importantly, the descriptions of some of the functions are not clear enough to allow coding by an independent researcher. Thus, nearly all investigators so far who have used the new test suite have literally used code available from the authors. This may induce a high level of standardization, but there are concerns when everyone is running the same code; further, this situation forces programmers to write programs that are compatible with those authors' conventions. Time will tell whether the CEC05 test suite becomes widely accepted; the best prediction now is that some of the functions from that set (the suite is too large, for one thing, and some of the functions are very CPU-intensive and time-consuming) will be added to the list of widely accepted test functions.

Mendes' (2004) thesis reported that evaluations of algorithms depend strongly on the way performance is measured. In running a test function iteratively, it is common to track the population best function result over time. The average of these scores at the end of a number of trials, a measure that Mendes called "average performance," is really a measure of local search, of the algorithm's ability to find the extreme value in a region of the search space. Another measure that is often used compares the population best performance against a criterion which has been determined ahead of time to indicate that the global optimum has been found; this can be viewed as a measure of global search ability.

Mendes' research found a dissociation between these two measures of success. He standardized performance results for each function and combined results from six functions in order to get a measure of the algorithm's performance across the entire test suite. Likewise, proportions of success were aggregated across the set of test functions for each algorithm.

Interestingly, some versions (these were FIPS and canonical particle swarms run with 1,343 different topologies) produced reliably good function results but had low proportions of success, and vice versa. In other words, some topology/interaction-mode versions were good at global or local search, but not the other. There was a small set of FIPS versions that scored well by both standards, but most particle swarm versions did better on one measure than the other. This advises future researchers to decide carefully which measure to use; a good mean function result indicates an algorithm that, when it does well, does very well, while a good proportion of successes demonstrates that the algorithm consistently gets near the global optimum, though it may not find the very best point in the optimal region.

Bounding the Search Space

A rather surprising problem has begun appearing in the literature, surprising because it seems so easy to prevent. For some reason, some researchers have begun clamping particles when they get to the edge of the search space, using various artificial techniques to keep the particle near the boundary, slow down its movement, or otherwise turn it back into the legal search space.

It is considered good practice to initialize the swarm, at least in research implementations, so that the global optimum is outside the range of the initial starting positions (Gehlhaar and Fogel, 1996; Angeline, 1998; Kennedy and Eberhart with Shi, 2001). Some investigators put the initial swarm at the edge of the search space, with the optimum near the center of it. This procedure ensures that the population is not simply collapsing upon the optimum due to some convergence tendency; the population is required to actually move somewhere to search for the optimum.

Some researchers, however, seem to believe that it is necessary to keep the particles inside some given range of the search space. Thus when a particle goes out of bounds, the program may tell it to stop on the boundary, or to bounce back according to some rule or other. This obviously makes it hard for the particle to behave properly when the optimum is near the border of the constrained space, and introduces anomalies into the trajectory.

It is not easy to think of reasons for trying to keep the particle in bounds. Two general approaches that will not upset the particle's trajectory can be recommended for commonly-found situations; these work by tampering with the objective function, not the behavior of the particle. First, it is not unusual to simply add a penalty to the function evaluation when the particle has gone into infeasible territory or outside the search space. If the penalty is adequately large, the forbidden location will not become the particle's previous best, and it cannot have any effect on the particle's or its neighbors' future trajectories.

As it is more meaningful to rate a trial by counting function evaluations than iterations, this second suggestion is preferable: if a particle has gone out of bounds, simply don't evaluate it. This skipped evaluation then can be made up on a later iteration.

If there is concern about the particle exploring outside the familiar search space, it is preferable to limit its velocity, as suggested in Eberhart and Shi (2000), rather than its position.

Research Methods II

Many statistical techniques have been developed for the analysis of experimental data. The question in this field is usually how to tell if one algorithm performs better than another on a function or set of functions. Sadly, today's standard analysis consists of a graph of several performance curves over time and a statement to the effect that "my algorithm is better than their algorithm."

In order to decide whether the difference between two sets of trials reflects a real difference, it is necessary to know something of the variance of the trials. At the least, standard deviations should be reported, it is better to report confidence

intervals, and even better to do inferential statistical tests comparing the conditions.

It is not necessary to get extremely sophisticated in this. For instance, in comparing numerical performance results for two algorithms on a function, a *t*-test can be run. These tests use the variance as a scale to determine whether the difference between sample means is "significant."

But note: there is a problem with performing multiple significance tests. Because the tests are probabilistic and use samples of observations to estimate populations, there is some chance that the conclusions are incorrect; and the greater the number of tests, the greater is the probability that one of the tests will appear significant when it actually is not.

Several methods have been devised for correcting for this: Duncan, Sheffé, Tukey, and Bonferroni adjustments are just a few. A simple method for getting valid significance tests from a number of *t*-tests is a modified Bonferroni adjustment (Jaccard and Wan, 1996). The simple Bonferroni adjustment simply takes alpha, the significance level, and divides it by the number of tests being conducted. Thus, if you were conducting five *t*-tests, and defining significance as $p<0.05$, then you would divide 0.05/5=0.01; then when you have done the *t*-test, only accept as significant those with $p<0.01$.

The problem with this adjustment is that it is too conservative. The researcher ends up rejecting results that are, in fact, significant. The Jaccard and Wan adjustment is conducted in these steps:

- Rank results so that *p*-values go from smallest to largest
- Number ranks inversely, from *N* to 1
- For each test, divide the *p*-value by its inverse rank: this is the new significance level
- Go from top to bottom, comparing the *p*-value to the new significance level, until a nonsignificant result is attained

This simple adjustment ensures that researchers do not report significant results based on chance.

"Origin-Seeking Bias"

Monson and Seppi (2005) published a paper with the provocative title "Exposing Origin-Seeking Bias in PSO," which has caused some readers to assume that there is origin-seeking bias in the particle swarm. This would mean that the algorithm is better at solving problems whose solutions lie at the origin, which includes many of the standard test functions. They also discussed another kind of bias, which is that of finding solutions at the center of the initialization region.

It is hard to see why the particle swarm would favor the origin. There is nothing in the formulas that would seem to cause the particle to gravitate toward that region. Importantly, the only thing that affects the position of the particle is the velocity, and if there is an "origin-seeking bias" in the velocity, e.g., the value tends toward zero, it only means that the particle will tend to stop moving.

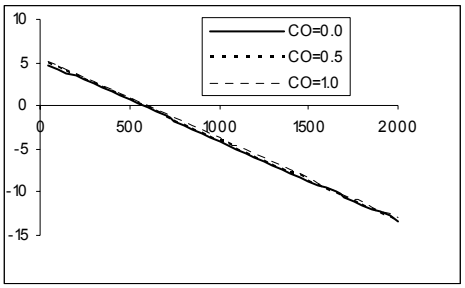
Monson and Seppi did not test a standard version of particle swarm, though. They used Clerc's TRIBES algorithm, which is a kind of particle swarm, though an unusual one, and it did not show signs of bias. They also tested the "pivot method," which is not similar to PSO, the bare bones PSO (Kennedy, 2003), which is an unusual version not considered standard, and a Gaussian method that they devised. Thus, findings of bias could not very well be attributed to the particle swarm, as their title suggested.

Those writers reported, on the basis of inspection of some performance graphs, finding origin-seeking bias in the pivot method and in the bare bones PSO, with TRIBES and the Gaussian PSO not showing any such marked tendency.

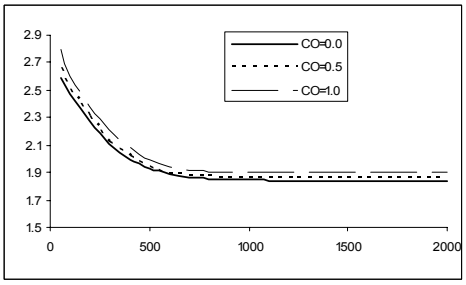
In order to further check this phenomenon, some experiments were conducted, using a typical two-term best-neighbor particle swarm, with von Neumann topology, N=20, on the same three functions that Monson and Seppi used: sphere, Rastrigin, and Rosenbrock, in 30 dimensions. The optimum was offset by 0, 50, and 100 percent of the problem space given in their paper.

Log-linear graphs were plotted for the three functions, shown in Figure 1. On the sphere and Rastrigin functions, there was no visible difference among the three versions. On the Rosenbrock function, there were differences, but the pattern was not clear. With offset=0.00, the mean best performance after 2,000 iterations was 74.516; with offset=50.00, performance mean was 58.482; and with offset=100.00, performance was 132.826. Thus, moving the optimum away from the origin actually improved performance, though moving it further made it worse.

Sphere



Rastrigin



Rosenbrock

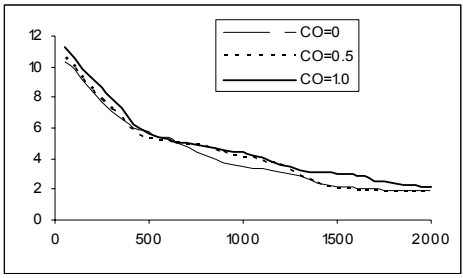


Figure 1. Graphs of performance of three functions, varying the center offset (CO) of each function by the proportion given.

Note that the version used was not intended to be an especially excellent particle swarm, but just a typical one. The Rosenbrock results called for closer examination. Final means for the three conditions are given in Table 1.

Table 1. Mean best performance for three levels of offsets on the 30-dimensional Rosenbrock function.

Offset%	Mean	S.D.
0.00	74.52	50.67
50.00	58.48	59.20
100.00	132.83	233.96

The offset of 100 percent resulted in a relatively high mean performance, and a very high standard deviation. Looking at the raw scores revealed an outlier, one trial with a mean of 1082.59. Removing this trial dropped the mean for the offset=100 cell to 82.84, s.d.=70.91. Thus all conditions, with one outlier removed, performed within one standard deviation of one another. *T*-tests comparing all pairs of means were nonsignificant, $p>0.25$, even with the outlier included, as large standard deviations explained the differences among means.

Since Angeline (1998), it has been standard practice for particle swarm researchers to initialize their populations in such a way that the global optimum is outside the range covered by the population; very many published results support the fact that the swarm is not limited to finding solutions inside the initialization range.

In sum, there does not appear to be any reason to think that the ordinary particle swarm algorithm has any bias toward the origin.

Probabilistic Algorithms

The trajectory of the particle sweeps it back and forth across a range of points that is centered on the mean of the sources of influence and is scaled to the differences among them. The series of positions tested represents a sample from a probability distribution, and some researchers have experimented with various ways to simulate the particle trajectories by directly sampling, using a random number generator (RNG), from a distribution of some theoretical interest.

Numerous researchers have been looking into this line of thought. For example, Krohling (2004) updated the velocity using the absolute value of a Gaussian distribution with mean=0.0 and s.d.=1.0 with good results. Sun, et al. (2004) used a strategy based on a quantum Delta potential well model to sample around the previous best points. Richer and Blackwell (2006) show good results using a Lévy distribution. Kennedy (2003) sampled with a Gaussian distribution, and later with various probability distributions (Kennedy, 2006).

These projects provide the insight that the power of the particle swarm algorithm is not to be found in the trajectories of the particles, but in their interactions. Because they teach one another and learn from one another, the population as a whole is able to progress from random guesses to very accurate solutions to difficult problems.

Conclusions

Early papers reported on a simple algorithm based on a metaphor of social behavior; primitive as it was, even those first swarms were powerful enough to catch the attention of the larger research community. The early reaction was, "We know it works, but we can't figure out *how* it works!"

Today, approximately eleven years later, a lot has been understood, but basic questions remain. The swarm works through the interactions of members of the population, but as has been seen, the exact methods for moving the particles, for communicating among the particles, and for structuring the population as a whole are quite flexible. Some methods work better than others, and methods often interact with one another as well as with features such as problem type, and it is still difficult to express fundamental principles describing how the swarm works and how best to implement it.

Parsimony has always been a valued ethic within the paradigm. While many researchers have shown that the algorithm can be improved by adding features to it, making it more complicated, the real research goal, at least at this time in the life of the young paradigm, is to strip it down to its essentials. When the principles that drive improvement over time are understood, then better versions can be built on that firm foundation.

In the meantime, as the particle swarm is used for various applications, a set of common practices has evolved. Generally these are practices that have proven themselves, but sometimes suboptimal methods become common in the community, as the swarm of researchers flocks around local optima.

The particle swarm is an optimistic tradition, based on cooperation and social learning. As the research community collaborates, greater wisdom accumulates about this new approach to problem solving. Over time, better problem solutions will propagate through the population, and those who have fixated on locally optimal ways of programming the swarm will be attracted to better methods. It just seems to be the way science works.

References

Angeline, P. J. (1008). Using selection to improve particle swarm optimization. *Proceedings of the 1998 IEEE Congress on Evolutionary Computation*, Anchorage.

Clerc, M. (2006). Particle Swarm Optimization. London: ISTE Publishing Company.

Clerc, M., and Kennedy, J. (2002). The particle swarm: Explosion, stability, and convergence in a multi-dimensional complex space. *IEEE Transactions on Evolutionary Computation, 6*, 58-73.

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

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

Gehlhaar, D. K., and Fogel, D. B. (1006). Tuning evolutionary programming for conformationally flexible molecular docking. In *Evolutionary Programming*, 419-429.

Jaccard, J., and Wan, C. K. (1996). *LISREL approaches to interaction effects in multiple regression*. Thousand Oaks, CA: Sage Publications.

Kennedy, J. (2006). In Search of the Essential Particle Swarm. *Proceedings of the 2006 IEEE World Congress on Computational Intelligence* (DVD).

Kennedy, J. (2003) Bare bones particle swarms. *Proceedings of the IEEE Swarm Intelligence Symposium*, 80-87. Indianapolis, IN.

Kennedy, J., and Eberhart, R. C., with Shi, Y. (2001). *Swarm Intelligence*. Morgan Kaufmann/ Academic Press.

Krohling, R. A. (2004). Gaussian swarm: A novel particle swarm optimization algorithm. *Proceedings of the 2004 IEEE Conference on Cybernetics and Intelligent Systems (CIS)*, Singapore, 372-376.

Liang, J. J., and Suganthan, P. N. (2005). Dynamic Multiswarm Particle Swarm Optimizer (DMS-PSO). *Proceedings of the IEEE Swarm Intelligence Symposium*, Pasadena, CA, 124-129.

Mendes, R., Kennedy, J., and Neves, J. (2004). The fully informed particle swarm: simpler, maybe better. *IEEE Transactions on Evolutionary Computation, 8*, 204-210.

Mendes, R. (2004). Population Topologies and Their Influence in Particle Swarm Performance. Doctoral Thesis, Universidade do Minho.

Mohais, A., Mendes, R., Ward, C., and Postoff, C. (2005). Neighborhood Re-Structuring in Particle Swarm Optimization. *Proceedings of the 18th Australian Joint Conference on Artificial Intelligence (AI 2005)*, Sydney.

Monson, C. K., and Seppi, K. D. (2005). Exposing origin-seeking bias in PSO. *Proceedings of the 2005 conference on Genetic and Evolutionary Computation*, 241 - 248.

Sun, J., Feng, B., and Wenbo, X. (2004). Particle swarm optimization with particles having quantum behavior. *Proceedings of the IEEE Congress on Evolutionary Computation*, Portland, Oregon, 325-331.

Richer, T. J., and Blackwell, T. M. (2006). The Levy Particle Swarm. *Proceedings of the 2006 Congress on Evolutionary Computation* (CEC-2006), Vancouver, Canada.

Shi, Y., and Eberhart, R. C. (1998). A modified particle swarm optimizer. *Proceedings of the IEEE Int. Conf. on Evolutionary Computation*, 69-73.

Suganthan, P. N., Hansen, N., Liang, J. J., Deb, K., Chen, Y.-P., Auger, A., and Tiwari, S. (2005). Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization. Technical Report, Nanyang Technological University, Singapore, May 2005 AND KanGAL Report #2005005, IIT Kanpur, India.

Wilke, D. N. (2005).Analysis of the particle swarm optimization algorithm. Master's dissertation, University of Pretoria, South Africa.