Dynamic-Probabilistic Particle Swarms

James Kennedy
US Bureau of Labor Statistics
Washington DC 20212
Kennedy_Jim@bls.gov

ABSTRACT
The particle swarm algorithm is usually a dynamic process, where a point in the search space to be tested depends on the previous point and the direction of movement. The process can be decomposed, and probability distributions around a center can be used instead of the usual trajectory approach. A version that is both dynamic and Gaussian looks very promising.

ACM Categories & Subject Descriptors
I.2.11 Distributed Artificial Intelligence
Multiagent systems

Keywords: Particle swarms

1. INTRODUCTION
Since its introduction in 1995 (Kennedy and Eberhart, 1995; Eberhart and Kennedy, 1995), the particle swarm algorithm has gone through many changes. Though early results were surprisingly good, and though the method had very few moving parts, it turned out to be quite difficult to understand how it worked, in order to improve it. Over the past decade, numerous modifications have been introduced, several of which have turned out to cause genuine improvements in performance, and several of which have helped to understand the dynamics of the swarm and how it is able to solve problems.

The canonical particle swarm algorithm is given as:

```
For each population member i do
    If eval(i) < pbesti then
        For each dimension d do
            pid=xid
        End
        pbesti=eval(i)
    End
    Identify best neighbor g
    For each dimension d do
        vid = khi× (vid +
            rand×(phi/2)×(pid – xid) +
            rand×(phi/2)×(pgd – xid))
        xid=xid + vid
    Next d
Next i
```


GECCO'05, June 25-29, 2005, Washington, DC, USA.
ACM 1-59593-010-8/05/0006.

where khi is usually a constant 0.729, phi=4.1, and rand is a uniform random number generator (Clerc and Kennedy, 2002; Kennedy and Eberhart, 2001).

In the original versions, as well as in the current canonical version, there are three sources of bias toward improvement. First, a particle is influenced by the best positions itself and its neighbors have attained thus far. Second, in standard versions the best-performing neighbor, or even the best-performing particle in the entire population, is chosen to be a source of influence on the particle. Finally, in most standard versions the individual particle's own previous best is used as a source of "influence," that is, the particle is attracted to the point of its own previous success.

The success of Mendes' (e.g., Mendes, Kennedy, and Neves, 2004; Kennedy and Mendes, 2002) fully-informed particle swarm (FIPS) suggests that the second source of bias toward improvement may be spurious. In FIPS, *all* of a particle's neighbors' previous best positions are used, averaged with random weighting, rather than only the best neighbor. Thus it turns out not to be necessary to identify and choose the best one. Further, in FIPS the individual's previous best is not a factor in adjusting its own trajectory. FIPS is a demonstrably powerful algorithm which, when implemented with an appropriate neighborhood topology, outperforms the canonical particle swarm on test problems.

FIPS modifies the canonical algorithm by changing the velocity formula somewhat:

$$v_{id} = khi\times (v_{id} + sum\ (rand\times phi\times(p_{kd} - x_{id})/K)$$

where the k subscript identifies the particle's neighbors and K is how many of them there are.

1.1 Dynamic trajectories
The particle swarm is considered to model a general theoretical perspective of socially situated minds (Smith and Semin, 2004), that is, the human cognitive system as a participant in a rich, dynamic social system where knowledge and learning are collaborative dynamical processes. Simultaneously, the particle swarm has gained attention as a method for engineering and applied mathematics, for its ability to solve very difficult problems.

The view of the particle as moving continuously through space has never fit perfectly well with the psychological model. Some theorists have suggested that thought can be conceptualized in terms of trajectories through a cognitive

space, but there is little in the way of empirical evidence to support the idea. True, an individual's beliefs and attitudes, etc. – their cognitive elements (Festinger, 1957) – are auto-correlated from one moment in time to the next, with similarity higher for proximal points in time than distal ones, but it is difficult to argue that a change in mental state beginning at point A and proceeding to E traverses all the points in between. Partly this reflects difficulty in the assignment of mental states to points on multidimensional continua, but partly it reflects a real quality of minds, that they do in fact sometimes make discontinuous changes, leaping from one region of the cognitive space to another.

The trajectory aspect of the particle swarm formulas came from their origin in experiments with Reynolds' (1987) boids and Heppner's (Heppner and Grenander, 1990) bird-flocking simulations, computer programs that represented the trajectories of birds on a computer screen. These early models provided sudden and important insights into the behaviors of flocking, schooling, and herding animals, and provided momentum to the new field of complex adaptive systems that hoped to answer theoretical questions with computer programs.

But birds fly in three-dimensional space, and the kinds of mathematical problems typically solved by the particle swarm were unlimited in dimension. From the psychological model, we would say that the problem space was the space of mental elements, where collision of individuals was agreement and not something painful to be avoided. Thus the kinds of collision-avoiding algorithms needed to keep birds afeather were not necessarily appropriate. And as cognitive elements seem to be innumerable, dimension can be high.

1.2 Probabilistic Search

Analysis of particle behavior with fixed "previous best" points found that the particle oscillated around a point that was the mean of the two bests, with a standard deviation that was proportional to, in fact about equal to, the distance between them. Kennedy (2003) demonstrated that a Gaussian random number generator could be substituted for the velocity formula with good results. That early version eliminated the dynamical trajectory that had seemed definitive of the algorithm.

But performance in the Gaussian version was not as good as the canonical algorithm. The next paper in the series of investigations found that the canonical particle swarm search trajectory contained bursts of outliers, that is, series of iterations with extreme values (Kennedy, 2004). Artificially adding such bursts improved the performance to a point comparable – almost – with the canonical algorithm.

Thus it appeared possible that a random number generator could be substituted for the velocity formula without losing the essential quality of the particle swarm algorithm. That essence was found in the collaborative sharing of successes among population members, with an individual's search influenced by its neighbors.

1.3 Dynamic-Probabilistic Search: Trimmed Uniform Probabilities

The question is how the particle chooses the next point in the search space to sample. While the points chosen are distributed symmetrically around the mean of the previous bests, the position at time *t* is dependent on the particle's position at *t-1*. This is what is meant by the term *dynamic* in this discussion: the particle's movement over time is defined as a series of points, each selected on the basis of the previous one.

The probabilistic models described in Kennedy (2003; 2004) are not dynamic in this sense. Those models select the next point solely on the basis of the previous bests, using a random number generator to produce a candidate problem solution vector from a probability distribution. The previous position of the particle is not taken into account. This is what is meant in this discussion by *probabilistic*.

In the canonical particle swarm, the choice of the next point x(t+1) is determined by reference to:

- x(t), the "current" or last-tested position of the particle
- p_i, the particle's previous best point
- p_g, some neighborhood or population best
- khi and phi, which are coefficients that control convergence
- v, the current velocity

The two "p" or previous best vectors vary among implementations. The FIPS formula can be considered a generalized model, where the various neighbors summed may be self and best neighbor, as in *lbest* versions; self and population best, as in *gbest* versions; or all neighbors, in FIPS. Further, the velocity component can be eliminated algebraically. In the usual formulation, there is a line that says:

$$x = x + v$$

This assignment statement means that x(t+1) = x(t) + v(t+1). From this we can see that v(t+1) = x(t+1) – x(t). Since the same thing happened on the previous time-step, we know that v(t)=x(t)-x(t-1). Thus the two formulas of the algorithm can be compressed into one by substitution:

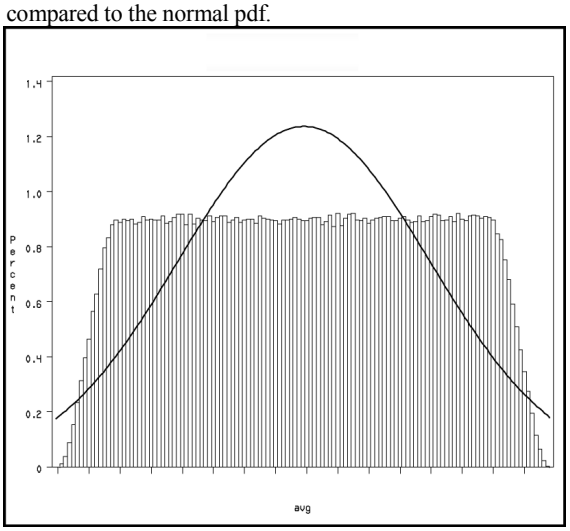
$$x(t+1) = x(t) + khi\times (x(t) - x(t-1)) + khi\times phi\times (sum(rand\times(p_k - x(t)))/K)$$

This reformulation shows us that the dynamical system is dependent on three terms summed. The first is x(t), the particle's current position in the search space; the second is a weighted difference term representing the direction that the particle is already going; and the third includes the influence of the previous bests.

It is noteworthy that the random number function is applied only to the last term of the formula. We can use that knowledge to calculate an expected value of x(t+1). Since rand is a uniform RNG in (0,1), its mean value is 0.5 or one half, we can write the expected position at t+1 as:

$$E(x(t+1)) = x(t) + khi\times (x(t) - x(t\text{-}1)) + khi\times phi\times (sum(0.5\times (p_k - x(t)))/K)$$

Figure 1. Typical histogram of points sampled by averaging three random numbers drawn uniformly from different ranges, compared to the normal pdf.



Though the exact position of the particle at t+1 is dependent on the output of the RNG, its probabilities are distributed around E(x(t+1)). If there were twenty or more neighbors, and each of their bests was equidistant from the particle's current position x, then the probability distribution for picking a point around E(x) would be normal. If there were a smaller number, the search would sample from a t distribution, and if there were two, for instance, self and best other, then the sample would be t-distributed with one degree of freedom, also known as Cauchy.

It would be handy if we could simply simulate the algorithm by sampling from a Cauchy distribution, as many software packages have Cauchy RNGs. There are several problems with this, however, the most important being that the two random terms being averaged are almost never drawn from the same interval, that is, (p_i-x) is almost never equal to (p_g-x). Further, both values (or all values in the FIPS version) will have changed by the next iteration. Each number then is drawn from an independent Cauchy or other t distribution, each with unique parameters. Empirical trials show that the effective probability distribution, as seen in Figure 1, is a truncated triangle, with uniform probability across the middle, decreasing in the tails.

This can be approximated by using a uniform RNG and trimming off the tails.

Having seen what the expect value of the particle will be, we need to determine minimum and maximum values for x at the next time-step. This will be determined by a couple of things. First, a certain limit is found when all random numbers equal 0.0, and another when they equal 1.0. In fact, if all terms averaged are of the same sign, then those set the limits for the iteration:

$$x(t+1) = x(t) + khi\times (x(t) - x(t\text{-}1)) + 0$$

and

$$x(t+1) = x(t) + khi\times (x(t) - x(t\text{-}1)) + khi\times phi\times (sum((p_k - x(t)))/K)$$

It is possible, though, that some previous bests are greater than x, and some are less than it. For instance, it is possible that, with both random numbers equal to 1.0, one (rand×(p-x)) will cancel out another, and the mean will be zero. We can account for both kinds of instances, by use of a program such as the following:

```
lim1=pi – xi
lim2=pg – xi

 *  OPPOSITE SIGN;
 if sign(lim1) ne sign(lim2) then do;
     center=(lim1+lim2)/2;
     width=abs(lim1-lim2);
 end;

 * SAME SIGN;
 else do;
     center=( lim1+lim2)/2;
     width=abs(lim1+lim2);
 end;
```

This code defines the center of the probability distribution and its range, or width. Points may be sampled then, from an area within `width/2` units of `center`.

This algorithm may then be run, using the RNG like this:

```
ranvar=center+ (rand×2-1) ×width/2;

x=x +
    khi× (x - x(t-1)) +
    khi× (phi/2) × ranvar;
```

In other words, take a random number in (-1,+1), multiply `width` by that, divide the product by two, and add it to `center`.

As seen in Figure 1 though, the distribution we want to simulate is not uniform over its entire range. In fact, if the algorithm is run sampling uniformly within `width/2` units of the center, the algorithm does not perform well. Variables quickly explode out of the range of their data type, and the program crashes.

Trimming off the tails is accomplished easily by dividing `width` by a number greater than 2. Dividing by 2.2 prevents crashing, but does not provide good optimization results. It appears that values near 2.5 allow the algorithm to perform well: results will be given in a later section. This version will be called the *trimmed-uniform particle swarm* (TUPS).

1.4 Dynamic-Probabilistic Search: Gaussian Probabilities

Another approach is to focus the search most intensively in the region of the expected value of x, with probabilities decaying for positions more distant from it. A Gaussian distribution fills the need here. The center can be at the expected value:

$$E(x(t+1)) = x(t) + khi \times (x(t) - x(t-1)) + khi \times phi \times (sum(0.5 \times (p_k - x(t)))/K)$$

and the standard deviation should be a function of the spatial distribution of the previous bests. Some experimentation found that a measure formed by summing the absolute values of the differences $(p_i - p_k)$, and dividing by the number of them (this is called `frange`), gave good results. The traditional values of khi and phi lose their meanings in this fundamental reworking of the algorithm, so they were replaced by arbitrary weights which could be tweaked:

$$x(t+1) = x(t) + W1 \times (x(t)-x(t-1)) + W2 \times (avgp-x(t)) + G(0,1) \times (frange/2.0)$$

In the tests that are reported below, W1 retained its previous value of 0.729, and W2=2.187. This algorithm will be called the *Gaussian-dynamic particle swarm* (GDPS).

2. EXPERIMENTS

Six functions were selected that are part of the traditional testbed for evolutionary computation and present a variety of well-known difficulties to a problem-solver. These are shown in the Appendix. All functions were initialized in the range shown in the third column of the table in the Appendix, in order not to capitalize too much on the happy opportunity of the optimal solution falling within the initial ranges of the variables.

All populations consisted of 20 individuals. The canonical particle swarm was connected by a Square topology (Kennedy, 1999), which has been shown to be a relatively good one, and the FIPS versions were connected using the "gr.2ed2" topology, a randomly generated network which performed very well in the FIPS research reported in Mendes (2004), and which is shown in Figure 2. Trials were run for 3,000 iterations, and were repeated 20 times in each condition.

2.1 Experiment One: TUPS

The trimmed-uniform particle swarm is compared in Figure 3 with the canonical and FIPS versions.

Figure 2. Topology "gr.2ed2" used for FIPS versions in the experiments reported below.

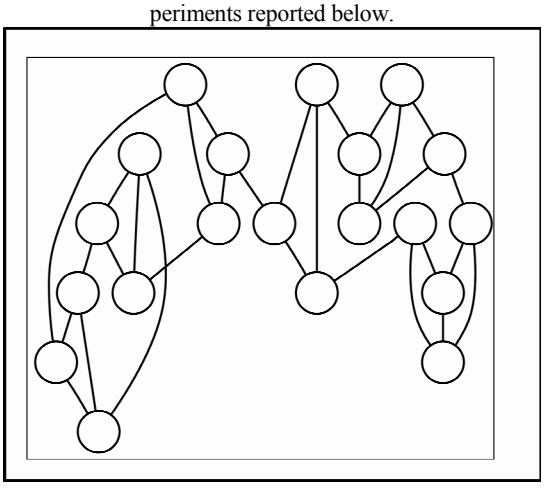
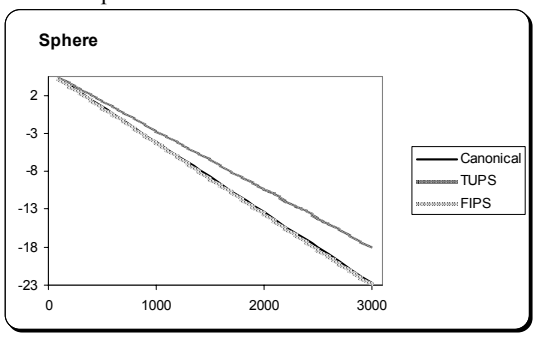
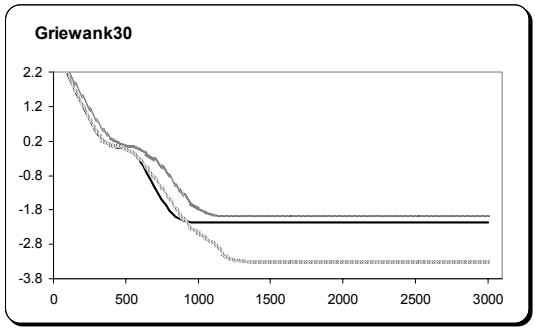


Figure 3. Comparison of TUPS with canonical and FIPS versions of particle swarms on some standard test functions.





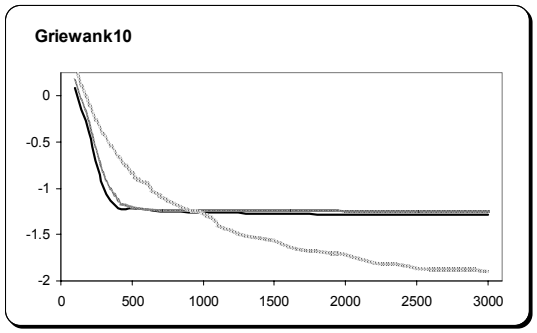
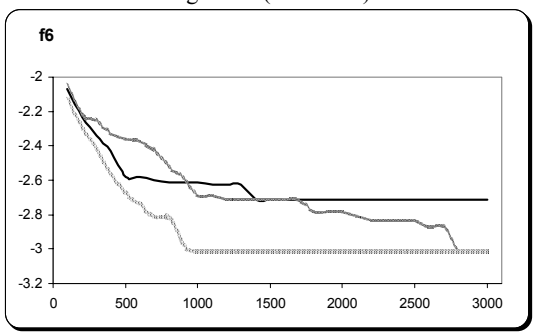
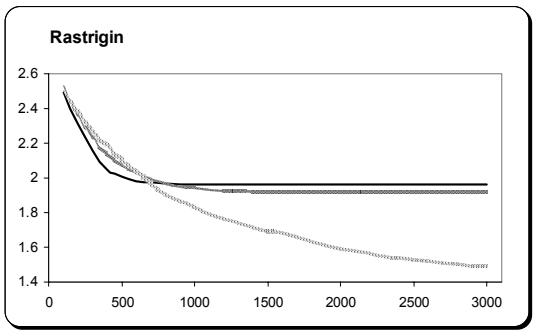
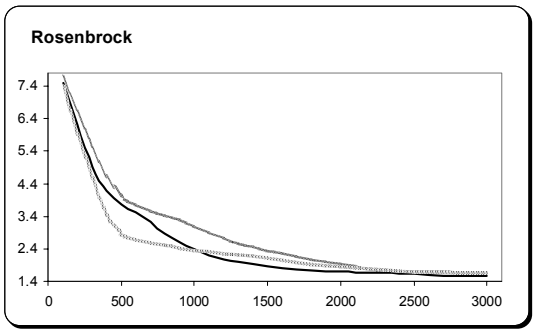


Figure 3. (continued)





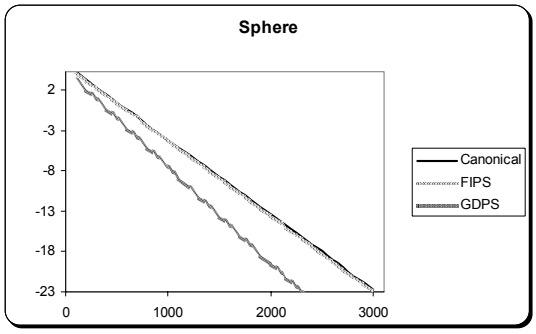


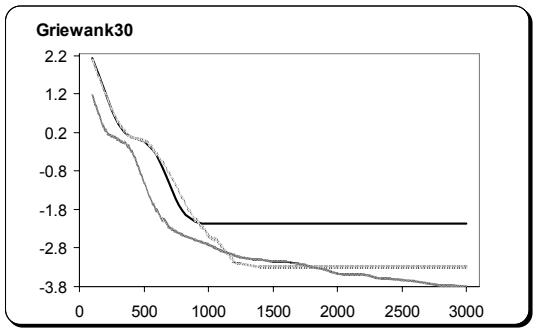
As can be seen, TUPS behaved quite similarly to the canonical algorithm, and FIPS outperformed both.

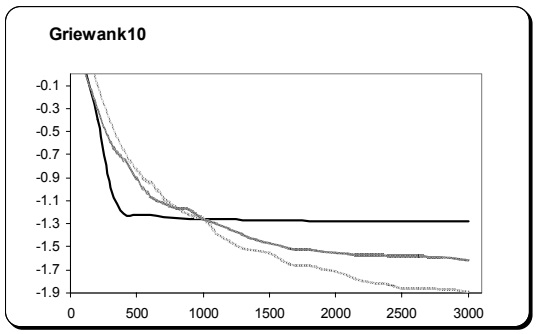
2.2 Experiment Two: GDPS

GDPS found better solutions than the others on all functions except the 10-dimensional Griewank, where FIPS performed best. It also found them faster. The graphs show that the algorithm was still progressing even when the runs were terminated after 3,000 iterations; this version seems resistant to premature convergence.

Figure 4. Comparing the Gaussian-dynamic particle swarm to the canonical and FIPS versions.







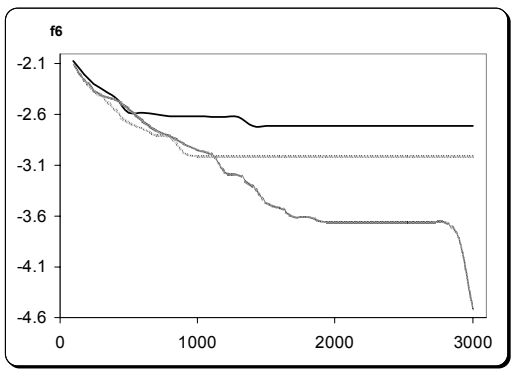
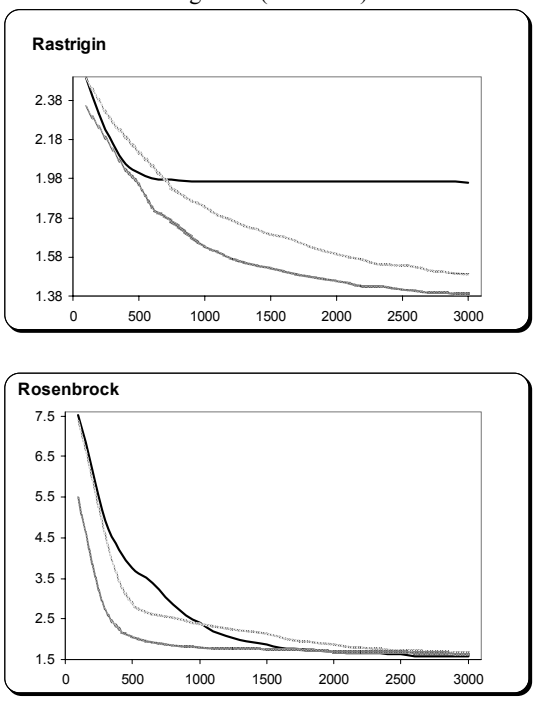


Figure 4. (continued)



As each trial went for 3,000 iterations, data from the last iteration were compared using t-tests, between algorithms. The results are seen in Table 1.

Table 1. P-values from t-tests comparing algorithms on the test functions. P<0.05 is considered significant.

	TUPS		GDPS	
Func.	**Canon**	**FIPS**	**Canon**	**FIPS**
Sphere	0.0290*	0.0295*	0.0006^x	<.0001^x
Grie,30	0.2098	0.0003*	<.0001^x	0.5162
Grie.10	0.7425	<.0001*	0.0040^x	0.0342*
F6	0.3888	1.0000	0.0385^x	0.1678
Rast.	0.2166	<.0001*	<.0001^x	0.0301^x
Ros.	0.5169	0.8614	0.6697	0.7306

* "Old" algorithm was significantly better; ^{x} New algorithm was better.

As can be seen, TUPS differed from the canonical PSO only on the sphere function, and was significantly inferior to FIPS on four of the six test functions. GDPS was significantly better than the canonical algorithm on all but one function, and was significantly better than FIPS on two, with FIPS better than it on one function.

3. DISCUSSION

The particle is often conceptualized in terms of its trajectory through the problem space, its position at each point in time stochastically dependent on its position at the previous moment. It moves in an oscillatory pattern centered around the centroid of the previous bests that influence the particle; the probability distribution of points sampled around this centroid however is complex and so far has not been successfully simulated using random number generators. If such a technique is found, it would result in a particle swarm that could be described as probabilistic, and not dynamic. Some parameters, for instance, locations of previously discovered good solutions, would simply be passed from the searchers to the RNG, and a new candidate solution would be generated.

The versions described in this paper used a RNG to generate a point in a probability distribution which was centered around the expected position on a dimension, assuming knowledge of the previous position of the particle and what direction it was already moving. The first of these attempted to simulate the canonical two-term particle swarm by mimicking the trimmed uniform probability distribution that emerges from the particle trajectory in the standard version. Results on some test problems suggested that it was a pretty good copy.

The second version improved on existing particle swarms by taking what had been learned and extending it. Understanding that the expected value of a particle's position on a dimension at time t was a function of various factors including its position at time t-1, the Gaussian-dynamic particle swarm sampled from a normal distribution around the expected position of the particle at the next iteration.

The TUPS version, while it emulated a canonical particle swarm, is limited in its searching ability. The point to be chosen always falls within a certain range – this is true of the canonical algorithm, as well. Both always sample from a hyperrectangular range of values in the problem space. The Gaussian model, though, can jump out of that area on any given iteration; search is focused around the expected value of x, with tails extending theoretically to infinity. This strategy for problem solving seems superior to the standard one, and may be better than the FIPS.

4. REFERENCES

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

Eberhart, R. C. and Kennedy, J. (1995). A new optimizer using particle swarm theory. *Proceedings of the Sixth International Symposium on Micromachine and Human Science*, Nagoya, Japan. pp. 39-43.

Festinger, L. (1957*). A Theory of Cognitive Dissonance*. Evanston IL: Row, Peterson.

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

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. (2003). Bare bones particle swarms. *Proceedings of the IEEE Swarm Intelligence Symposium 2003 (SIS 2003)*, Indianapolis, Indiana, USA. pp 80-87.

Kennedy, J. (2004). Probability and dynamics in the particle swarm. *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, Portland, OR, 340-347.

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., with Shi, Y. (2001). *Swarm Intelligence.* San Francisco: Morgan Kaufmann/ Academic Press.

Kennedy, J., and Mendes, R. (2002). Population Structure and Particle Swarm Performance. *Proceedings of the 2002 World Congress on Computational Intelligence*.

Mendes, R. (2004). Population Topologies and Their Influence in Particle Swarm Performance. Ph.D. Thesis, Escola de Engenharia, Universidade do Minho, Portugal.

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

Reynolds, C. W. (1987). Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics, 21*, 25-34.

Smith, E.R., and Semin, G.R. (2004). Socially situated cognition: Cognition in its social context. *Advances in Experimental Social Psychology, 36,* 53 - 117.

Functions used in experiments. Populations were initialized in the range between the two values under "Init. Range."

Function	**Formula**	**Init. Range**
Sphere	$f_1(\vec{x}) = \sum_{i=1}^{n} x_i^2$	50, 100
Schaffer's f6	$f_6(\vec{x}) = 0.5 + \frac{(\sin\sqrt{x^2+y^2}) - 0.5}{(1.0+0.001(x^2+y^2))^2}$	50, 100
Griewank	$f_7(\vec{x}) = \frac{1}{4000}\sum_{i=1}^{n}(x_i-100)^2 - \prod_{i=1}^{n}\cos(\frac{(x_i-100)}{\sqrt{i}}) + 1$	300, 600
Rosenbrock	$f_9(\vec{x}) = \sum_{i=1}^{n}(100(x_{i+1}-x^2)^2 + (x_i-1)^2)$	15, 30
Rastrigin	$f_{10}(\vec{x}) = \sum_{i=1}^{n}\left\lfloor x_i^2 - 10\cos(2\pi x_i) + 10\right\rfloor$	2.56, 5.12