Chapter 6

SWARM INTELLIGENCE

James Kennedy
Bureau of Labor Statistics

Swarm intelligence refers to a kind of problem-solving ability that emerges in the interactions of simple information-processing units. The concept of a swarm suggests multiplicity, stochasticity, randomness, and messiness, and the concept of intelligence suggests that the problem-solving method is somehow successful. The information-processing units that compose a swarm can be animate, mechanical, computational, or mathematical; they can be insects, birds, or human beings; they can be array elements, robots, or standalone workstations; they can be real or imaginary. Their coupling can have a wide range of characteristics, but there must be interaction among the units. Given the diversity of paradigms that call themselves swarm intelligence, this chapter will focus on the particular approach known as *particle swarm optimization*.

1 PARTICLE SWARMS

The particle swarm algorithm is based on a certain insight regarding human behavior and cognition. The insight is simple, as is the algorithm that follows from it. Simply put: *people learn to make sense of the world by talking with other people about it*. This not-very-technical observation enables us to design a family of computer algorithms that encode some population of individuals who propose solutions to a problem, and then are able to refine those solutions by interacting with their "peers," picking up suggestions from their neighbors, and adjusting their own patterns of variables.

Over time, in one of these programs, individuals begin to find good problem solutions, even when the problem is very difficult—for instance, when it is multimodal, nonlinear, noisy, or nondifferentiable. This technique has been used for binary problems, multiobjective problems, dynamic problems that keep changing, and many other tough kinds of problems. It has been used for a variety of engineering problems,

from the maintenance of electrical grids to the classification of physiological variables in early diagnosis of disease.

As will be seen, there have been, over the past decade, many variations in the particle swarm algorithm. Some versions are almost unrecognizable, and some variations are extremely minor tweaks that enhance performance significantly. The following sections will introduce the canonical form of the algorithm and some common variations, and then discuss some ongoing research on different problem domains and different lines of alteration of the algorithm.

1.1 General Characteristics

Every known version of the particle swarm algorithm has certain characteristics. First, every version employs a *population* of particles. The number of these is typically far less than in the usual evolutionary algorithm; most researchers use twenty to fifty particles in a population.

Second, every particle swarm has some sort of *topology* describing the interconnections among the particles. The "traditional" topologies, which are becoming somewhat antiquated in light of current research but are still widely used, are call *gbest* and *lbest*. The *gbest* topology (or *sociometry*, since it is often considered to be like a kind of social network) can be thought as a fully interconnected population; that is, every member of the population can be influenced by every other one. In the standard particle swarm, this means that particles are affected by the individual that has found the best problem solution so far—the very best one in the population. Thus, though *gbest* contains the greatest possible number of connections between pairs of population members, in practice it really only means keeping track of the best solution found. The *lbest* sociometry is a ring lattice, where every particle is connected to the particles on either side of it in the population array. As will be seen, the advantage of this structure is that subpopulations can converge independently on diverse optima in the problem space. Thus the *lbest* topology, while typically slower to converge on an optimum, is also less susceptible to the allure of local optima; its search is slower and more thorough than *gbest*'s. Thousands of other topologies have been tested, as will be described below.

A third characteristic of every particle swarm is some choice of a *change rule*. The particle moves through the search space, selecting a point at time t that is dependent on its position at $t-1$, its previous successes, and the previous successes of its neighbors. There is a "standard" formula for determining the next step, but this has evolved over time, and some researchers even have tried replacing the formula with a kind of random number generator. It may seem that the particle swarm is typified by the trajectories of the particles through the search space, but this view is only correct if the concept of "trajectory" is stretched to include random search around a center.

Evolutionary search is often described in terms of two phases, called *exploration* and *exploitation* [28]. The search algorithm first searches the environment for good regions, and then, having found a good region of the search space, looks for the best point in that region. In the particle swarm, however, step-size—the range of investigation of a particle in the search space—is scaled to consensus in the neighborhood; if a particle and its neighbors have had success in a particular area, then that area will be searched, but if some neighbors are still investigating

other regions of the problem space, the particle will still tend to explore widely. Concepts such as exploration versus exploitation seem to assume certain characteristics of the search space—for instance, that it contains subdivisions that are locally monotonic. The particle swarm's assumptions are more flexible; a well-designed particle swarm can search multiple regions of the space simultaneously, and particles can switch flexibly from one locally optimal region to another.

A fourth characteristic of all known particle swarms is what may loosely be termed the *interaction rule*. A particle considers its successes and some other particles' successes in determining the next point to test in the search space. How this point is chosen, though, may follow any of a number of possible rules, and the list of rules is growing as researchers push the limits of what is known in this young field.

1.1.1 The Canonical Algorithm

This section will present the most common form of particle swarm algorithm as it currently exists, and discuss some of its features.

1.1.2 Constants and Initialization

There is no law that says that particles must be initialized randomly through the problem space, but that is the general practice, given that there is no special knowledge about a better way to do it. Three variables need to be initialized, most importantly the positions of the particles, represented algebraically as $\vec{x}_i$, and their velocities $\vec{v}_i$. If the researcher chooses to initialize $\vec{v}_i$ to a vector of zeroes, then $\vec{p}_i$ should be different from $\vec{x}_i$ in order to make the particles move; but more frequently, $\vec{p}_i = \vec{x}_i$ for the first iteration, and nonzero velocity values propel the particle through the search space in some randomly chosen direction.

*X*max is simply a constant that defines the range of the search space. It may vary on each dimension, but is simplified here as a single constant, as many test functions treat variables identically. The constant *V*max has a history that will be described below. It may be used to constrain search during the iterative phase of the algorithm, but it is a heuristic device only; in the canonical version, *V*max simply serves to initialize particle velocities in-bounds.

The "acceleration constants" phi1 and phi2 and the "constriction coefficient" chi (φ_1, φ_2, and χ) are the result of analysis by Maurice Clerc, described below. The value of chi is derived from the sum of the two phi constants by a formula, and phi1 and phi2 are set to sum to 4.1, just because it works.

1.1.3 Neighborhood Best

In the canonical particle swarm, each particle is influenced by its best neighbor. The set of particles to which a particle *i* is topologically connected is called *i*'s *neighborhood*. The neighborhood may be the entire population or some subset of it. Normally the algorithm loops through the neighborhood, comparing the best function results found so far (*pbest*[*k*]), and assigns the index of the best particle to the variable *g*. Note that not all versions of the particle swarm use the best neighbor; some use an average of all neighbors' previous successes, but these versions are relatively new and not standard.

Table 6.1. Pseudocode representation of the canonical particle swarm algorithm.

```
Xmax ← range of search space
Vmax ← proportional to range of search space
phi = 4.1
chi = 0.792

Initialize
  for i = 1 to number of particles
      for j = 1 to number of problem dimensions
      x[i][d] = uniform rand() in Xmax // position of particle i on dimension d
      v[i][d] = uniform rand() in ± Vmax
      p[i][d] = x[i][d] // for start
    pbest[i] = eval(p[i]) // arbitrary for initialization
    if pbest[i] < pbest[gbest] then gbest = i

Iterate
  for i = 1 to number of particles
    g = index of neighbor with best pbest[i]

    //Select point to test
    for j=1 to dimension
      v[i][d] = chi × (v[i][j] + rand()× phi1 × (p[i][d] – x[i][d]) + rand()× phi2 × (p[g][d] – x[i][d]))
    for j = 1 to dimension
      x[i][d] = x[i][d] + v[i][d]

    // Evaluate new point
    eval = eval(x[i])

    // If it's better than best so far
    if eval < pbest[i] then do
      pbest[i] = eval
      for j = 1 to dimension
        p[i][j] = x[i][j]
      if pbest[i] < pbest[gbest] then gbest = i
Until termination criterion is met
```

1.1.4 Selecting a Point to Test

The particles oscillate around a point defined as a stochastic average between the individual particle's previous best and the best neighbor's previous best. This oscillation is the result of a formula that adjusts the particle's velocity at each time-step. As the particle gets farther from the mean, the velocity becomes smaller until it reverses direction and the particle goes the other way.

As the two terms added to $v[i][d]$ are weighted by uniformly distributed random numbers, the pattern is not cyclic; it gets its characteristic amplitude and wavelength from the value of phi, but does not strictly adhere to a periodic pattern. But this source of irregularity is not the most important source of variation in the particle's trajectory. More importantly, the particle may find a new point in the search space that is better than its previous best. Further, some member of its neighborhood might find a better point. In this case, it may be that the "best neighbor" at time t is different from that at t–1, or it may simply be that the same

neighbor has found a better point, so that $p[g][d]$ has changed, even though g has not. These improvements, of course, guide the progress of the algorithm and are central to its ability to optimize complex functions. The result, though, is an aperiodicity of the particles' trajectories that is very hard to comprehend: this messy, complicated, ever-evolving, highly interactive process is what we call *swarm intelligence*.

The amplitude of the oscillation turns out to be a very important feature of the particle swarm search strategy. It can be seen to have two components. First, as seen above, the formula, even in its deterministic form, produces a wave that searches back and forth. But more importantly, and more difficult to grasp, the difference between the individual's previous best $p[i][d]$ and the neighborhood best $p[g][d]$ wanders constantly as a result of the random coefficients applied to each variable. The random coefficients vary between 0.0 and 1.0, meaning that, if a coefficient equals zero, that variable ($p[i][d]$ or $p[g][d]$) will have no effect. If both coefficients are near zero, the velocity will retain its previous value, and if both are near 1.0, the velocity may be greatly modified. The particle may move into a region on the next step that is bounded by a sort of hyperrectangle, bounded by the current position plus the t–1 velocity in the corner where all the random numbers equal zero, and the current position plus the t–1 velocity plus the differences between the current position and both previous bests, multiplied by their acceleration constants φ_1 and φ_2, when the random numbers equal 1.0.

The size of this hyperrectangle is defined by the difference, on each dimension, between $p[i][d]$ and $p[g][d]$. The meaningful implication of this is that the range of the search is modulated by the difference between particles' previous best points – what we call *consensus*. When a particle and its neighbor have found success in the same region of the search space, they "agree" on where to look for even better points. The degree of consensus scales the extent of the search.

In sum, the particle selects points in the search space based on a kind of oscillatory trajectory that carries it back and forth around the region defined by where it has had success before, and where its neighbors have found good solutions.

1.1.5 The Evaluation

The particle swarm searches for the minimum or maximum of a function, and evaluates the entire function all at once. In a sense this resembles the measurement of *fitness* in evolutionary computation methods. Importantly, this approach is distinguished from the variable-by-variable type of evaluation seen in cognitivistic – i.e., traditional artificial intelligence – approaches to problem solving in computer programs. It is not possible to assign credit to variables in terms of their contribution to the evaluation of an entire pattern, nor does it turn out to be necessary.

1.2 The Sociocognitive Metaphor

Traditional AI grew up together with cognitive psychology, as a reaction to the behaviorism that prevailed in psychology in the middle of the twentieth century. The behaviorists had essentially blocked any study of mental phenomena in the universities, so the "cognitive revolution" began with a commonsense approach,

looking at the thinking processes of the individual as an isolated unit. That point of view is consistent with the *experience* of thinking; that is, our thoughts seem like a private monologue or story, occurring in our own phenomenological worlds.

Social psychologists, though, were forming another perspective on cognition. Many studies showed that individuals' beliefs, memories, attitudes, and thought processes were heavily influenced by those around them. As early as 1936, Sherif [67] was showing how reported perceptions were shaped by norms; Asch's [5] famous conformity experiments showed how behavior could be shaped by the influence of others; Crutchfield [19] and Deutsch and Gerard [21] further focused in on factors affecting the social influence of groups on individuals. At the same time, persuasion researchers from Hovland [36] to Cialdini (e.g., [13]) were noting that the choices people make are directly and irrefutably affected by their social atmosphere.

An important study by Nisbett and Wilson [55] showed that people are often unaware of their own cognitive processes, and are unable to report verbally how their own minds arrived at a conclusion – many self-reported cognitive processes are better described as "rationalizations" than "descriptions" of cognitive processes. Note that early AI was largely based on self-reported processes [54]; as such, it is clear that there will be room for improvement over the heuristic algorithms employed in traditional artificial intelligence paradigms.

Fundamental to the particle swarm algorithm is a view of cognition as a social process. The internal monologue of thought is easily seen as an imagined conversation between the thinker and some other – the other may be another person, another instance of the self, or, it seems possible, the self may perceive itself as the listener as another imagined individual produces the monologue. In any case, Levine, Resnick, and Higgins [50] noted that, "Outside the laboratory and the school, cognition is almost always collaborative" (p. 599). According to them, intersubjectivity, a shared understanding of the task and context, is required for coordinated cognitive activity, and they proposed research in a new field they call *sociocognition*, which comprises the integration of social interaction and cognition. Their statement was adamant: "Although some might claim that the brain as the physical site of mental processing requires that we treat cognition as a fundamentally individual and even private activity, we are prepared to argue that all mental activity – from perceptual recognition to memory to problem solving – involves either representations of other people or the use of artifacts and cultural forms that have a social history" (p. 604).

Consider, for example, the topic of false memory, for instance as researched by Elizabeth Loftus [51]. Loftus has shown repeatedly that memories created by suggestion are indistinguishable from "real" memories, that is, memories generated by the individual's first-hand perception of a situation. This research supports the view of memory as a social construction. Decades of work by Albert Bandura [8] has consistently shown the numerous and important ways that human cognition is a function of social learning; Bandura focuses on observational learning, mostly, where the individual sees how someone else has solved a problem, and imitates that. Likewise, Latané's social impact theory research [56] shows an individual's attitudes and beliefs to be a simple function of the strength, immediacy, and number of others who hold that attitude or belief. Very recent research by Wegner [72] shows that even "conscious will" is largely illusory.

The starting-point then for understanding the metaphor underlying this problem-solving technique is to understand that human intelligence itself operates through interaction among individuals. Our conscious experience of thinking is not a good scientific description of cognition. It appears that a better description would focus less on internal, private processes, and more on interpersonal dynamics.

Cognitive dissonance theory [30; 31] portrays the mind as a set of cognitive elements related to one another in complex logical and affective ways. The individual is motivated to find and maintain consistency among the elements; for instance, it is uncomfortable to hold two beliefs that logically contradict one another, or to find oneself in agreement with someone one doesn't like. In the current view, "dissonance" is the result of a cognitive evaluation function, where a vector of cognitive elements and their interrelationships is input, and a single measure of goodness is produced. Thinking, then, is seen to be a process of optimization, of constant searching for ways to arrange and rearrange beliefs and attitudes so as to produce the most consistent – least dissonant – pattern. One important aspect of this process is that people simply talk to one another, and observe one another, and learn from one another how to make sense of a confusing world.

Particle swarms are most typically used by engineers and others who apply mathematics to difficult tasks. These innovators need a method that works, something that can solve hard problems with a minimum of tweaking. The two perspectives on the algorithm, the sociocognitive perspective and the engineering perspective, constantly interact symbiotically to produce new developments in the field—new ways to make the algorithm work better, faster, and more efficiently. The metaphor of interacting human minds is a rich one for improving the paradigm. For instance, we may ask:

- Is human social interaction best conceived as averaging between two points of view?
- Are individuals influenced equally by everyone they know? And, if not, how are the differences best summarized in a computer algorithm?
- Is influence reciprocal between individuals?
- Are individuals affected by other individuals, or by statistical norms of their group?

. . . and so on. Thinking about these kinds of questions sometimes results in new variations of the algorithm, oftentimes with improved optimization results. The social–psychological and applied-mathematical facets of the algorithm benefit mutually from one another.

1.3 Origins of the Present-Day Canonical Particle Swarm

This section describes some steps in the development of the algorithm since its initial discovery in 1994. A subsequent section will describe some current research frontiers, and finally, some speculation about future directions will be offered.

1.3.1 Social Psychology and Genetic Algorithms

The seeds of the particle swarm were planted when the present author worked with Bibb Latané and his colleagues at the University of North Carolina, creating computer simulations of social systems using the Warsaw Simulation System [56]. These simulations, which were supported by decades of human-subjects research by Latané on social impact theory, were based on cellular automata [74]; a population was coded as a grid of individuals who interacted with one another according to rules programmed by the researcher. The outstanding finding of that research was that simple local interaction rules, similar to what Latané had derived from his experimentation and observational research with human subjects, could result in consistent and meaningful large-scale patterns of belief, attitude, and behavior in the society.

In 1992, the present author began meeting with a computational intelligence brown-bag lunchtime group, comprising mostly engineers and computer scientists (including Russ Eberhart), at a research institute that shall remain nameless. In those lively discussions it became apparent that the social dynamics simulated in the Psychology Department at Chapel Hill had much in common with the dynamics simulated in evolutionary computation. Populations of individuals interacted, the population changed over time, and global order emerged from local behavior.

The most important difference had to do with the presence of something called "a problem." The evolutionary algorithms were being used to find the optima of complicated mathematical functions, whereas the social simulations simply iterated to an equilibrium state. But it seemed obvious that social systems do solve problems; the state of human knowledge, for instance, does improve over time. What would happen if the states of the individuals in the social simulations were evaluated on the basis of some measure of goodness, similar to cognitive dissonance?

There are important differences between evolutionary and social–psychological processes. One of the biggest philosophical differences has to do with selection. Evolutionary methods find increasingly better problem solutions by killing off worse-performing members of the population and letting better ones reproduce; each iteration represents a "generation" in the history of the population. Societies, though, retain individuals over iterations; an iteration is simply a unit of time. Individuals are seen as changing over time, rather than being replaced.

The fundamental principle of evolution is competition for survival. Individuals who are allowed to survive may reproduce. Mutation, crossover, and/or other operators may affect the offspring, and as the fittest of each generation is allowed to survive and reproduce, the quality of the population, typically, improves over time.

The fundamental principle of the particle swarm is cooperation and sharing of knowledge. Every individual participates in the population's improvement as both teacher and learner. Over time, due to a kind of ratchet effect [69], the patterns of variables improve. Individuals that communicate with one another tend to gravitate into the same regions of the problem space, as better-performing individuals influence lesser ones. Depending on the population topology and other factors, the individuals may cluster around diverse local optima, or they may all

end up in the same region. When local optima have been well searched, it is quite probable that the particles are clustered around the global optimum, or at least an excellent local one.

1.3.2 Flocking and Schooling

The first particle swarm program was written by modifying a bird-flocking simulation. Two disparate groups of researchers in the 1980s [64; 33] had derived very similar models of the dynamics of bird flocks. Reynolds, working from a computer-graphics perspective (e.g., he wanted to be able to portray realistic bird-flock animations on a computer screen), concluded that bird flocking could be simulated using three rules:

- Separation: steer to avoid colliding with local flockmates
- Alignment: try to move in the same average direction as local flockmates
- Cohesion: steer to move toward the perceived center of the local flock

Heppner, a biologist, made three-dimensional movies of bird flocks, and carefully studied the dynamics of their choreography. His model, though it was developed independently of Reynolds', contained essentially the same three rules, plus a fourth: attraction to a roost. His flocks eventually settled down.

The present author had been experimenting with these flocking models, and added one more feature, with surprising results [45]. Inspired by Heppner's "attraction to a roost," "cornfield vector" was added, which in the first program was simply a two-dimensional point on the plane of the computer monitor. This point was considered to simulate some food on the ground; birds flying past might see the food, or some sign of the presence of food, and most importantly, birds flying past could see that other birds seemed to be zeroing in on some target. Thus, members of the flock were attracted toward positions that other members of the flock had found to be relatively near to food.

The first experiments were shocking. The flock immediately converged on the point on the screen, as if sucked in by a vacuum cleaner.

That first algorithm worked as follows: each bird

- evaluated its distance from the cornfield
- identified some "neighbors" who were nearby on the display plane
- identified which of its neighbors had come closest to the target point, and where that had happened (note that the location of the point was not known, but only the distance from it)
- if its position was to the right of (above) its own previous best point, then it moved some random amount to the left (down), else it moved a random amount to the right (upward)
- if its position was to the right of (above) the best neighbor's best point, it moved a random amount to the left (down), otherwise right (upward)

The success of the food-searching program prompted the second set of experiments, conducted the same day. The evaluation of distance from an arbitrary

point on the screen was replaced by an XOR feedforward neural network. A network was defined with two input nodes, three hidden nodes, and one output, requiring nine connection weights and four biases. Thus optimizing the weights (including biases) meant searching through a thirteen-dimensional space. Weights were initialized with random values, and the program ran iteratively. For testing purposes, two of the weights were graphed on the screen, meaning that the entire flock could be watched as a display of swarming particles (at this time, the algorithm was yet unnamed). The code for evaluating the network was cut from some public-domain source code and pasted into the program where the evaluation of distances to the cornfield vector had been.

Again, the flock had no difficulty finding an optimal matrix of weights. The plotted points zoomed immediately, it seemed, with no hesitation, toward a configuration that resulted in squared error in the network very near zero.

At this point, I sent some code to Russ Eberhart, who compiled it and agreed that this new algorithm did seem to successfully optimize the weights in the network. He and I have worked together on the paradigm ever since.

1.3.3 The Evolution of the Paradigm

We made several changes to the paradigm almost immediately. First, since the task of identifying neighbors in the search space (a vestige of the flocking simulation) was very expensive computationally, we experimented with topological neighbors, both *gbest* and *lbest*, and found that these worked just as well. The advantage here was that neighborhoods were constant, and did not have to be recalculated on every iteration depending on the positions of all the particles in the search space.

Another important change in the first weeks of experimentation involved replacing the inequality rule. The very first versions simply said that if a particle's current position was greater than the stochastic average of its and its neighbor's *pbests*, it would change by a negative amount, and if it was less than the target it would change by a positive amount. The new version used the distance, on each dimension, between the particle's present position and the stochastic average of the previous bests:

```
v[i][j] = v[i][j] +
   rand() × (phi1) × (p[i][d] – x[i][d]) +
   rand() × (phi2) × (p[g][d] – x[i][d]);
      if v[i][d] > Vmax then v[i][d] = Vmax;
      else if v[i][d] < -Vmax then v[i][d] = -Vmax;
x[i][d] = x[i][d] + v[i][d];
```

There was some initial experimentation with the two constants, called above phi1 and phi2; some experiments found that it was better to have the first one bigger than the second, and some found it was better to make the second constant bigger. These two constants, randomly weighted, assign weight to the two "differences," where the first is an "individual" (sometimes "cognitive") term, and the second is an "other" (or "social") term. Because it did not appear that one weighting scheme or the other was superior across a range of situations, and in the interest of parsimony, we decided to make the two constants equal,

and let the random number generator make the decision about which should be larger.

Now that the two phi constants were established to be equal to one another, the system still needed to have values defined for them and for Vmax. Numerous studies were conducted, both published and unpublished, and by convention most researchers felt that a value of 2.0 worked well for each phi. Values that were too low tended to allow the particle to wander too far from promising regions of the search space, and values too high tended to jerk the particle back and forth unproductively.

Vmax was a different kind of problem. Vmax set a limit for the velocity, which otherwise was defined by the size of the previous iteration's velocity and the two differences. Without Vmax the system simply exploded, for reasons that were initially not understood. If the velocity was not limited, it would become larger and larger with each iteration until it exceeded the data type range. With Vmax the particle's trajectory became pseudocyclic, oscillating (or maybe "twitching" is a better word for it) around the average of the previous bests.

When the difference between the individual's and other's pbests were large, Vmax had the effect of slowing the exploratory search by limiting the particle's trajectory to "smallish" steps between the two pbests and slightly beyond them. When the difference between them was small, though, especially when it was small relative to Vmax, the amplitude of the trajectory was modulated by Vmax, and the particle was not able to converge in the later stages of search.

What was the right size for this constant? Research with neural networks and standard testbed problems typically concluded that a value of approximately 4.0 was appropriate. But several papers were published during this time that noted the inability of the swarm to converge on optima – it is usually desirable for an optimizer to search in smaller steps as it approaches the peak of an optimal region. It was clear that something needed to be done about this unsatisfying situation.

1.3.4 Controlling Explosion and Convergence

The present author presented a paper at the 1998 Evolutionary Programming conference (Kennedy, [40]), which plotted the trajectories of some one-dimensional deterministic particles, where the previous bests were combined and did not change with time. In other words, these simplified particles followed the rule:

$$v = v + (phi) \times (p - x)$$
$$x = x + v$$

These graphs showed, first of all, that particles without random coefficients did not explode; their trajectories were very orderly and well-behaved. Second of all, the trajectories for each value of phi were unique, often presenting the appearance that the particle in discrete time was skipping across underlying waveforms, which varied with the value of phi. The graphs were presented to the conference as a kind of puzzle. Why does the system explode when randomness is added? What is the nature of the underlying wave patterns?

These questions resulted in several avenues of research. In an informal talk to a workshop at the Congress of Computational Intelligence in Anchorage later

that same year, Doug Hoskins noted that simply varying phi between two values would cause the velocity to increase out of control. He compared this effect to pumping a swing, where the change in the coefficient added energy to the system.

Ozcan and Mohan [57, 58] analyzed the simplified particle system and determined that the particle was "surfing the waves" of underlying sinusoidal patterns. They proposed that particles might be equipped with "sensors" that determined what kind of search was being conducted; if a particle stayed in a region too long, for instance, a coefficient could be adapted to enlarge the scope of search.

At about this time, Shi and Eberhart [24, 23] introduced a convention they called the "inertia weight," which depreciated the contribution of the *t*-1 velocity. By using an inertia weight less than 1.0, it was possible to control the explosion of the algorithm. The inertia weight algorithm was implemented as

```
v[i][j] = W × v[i][j] +
   rand()× (phi1) × (p[i][d] – x[i][d]) +
   rand()× (phi2) × (p[g][d] – x[i][d]);
x[i][d] = x[i][d] + v[i][d];
```

where W was the new coefficient in the algorithm. Several values were experimented with; those researchers settled on a method of reducing the value of the inertia weight, typically from about 0.9 to about 0.4, over the course of the iterations. Though Vmax was no longer necessary for controlling the explosion of the particles, Eberhart and Shi continued to use it, often setting Vmax = Xmax, in order to keep the system within the relevant part of the search space.

In France, Maurice Clerc was developing a mathematical analysis of the system in order to understand the explosion and convergence properties of the particle swarm. He reduced the simplified, deterministic, one-dimensional, single-particle system to an algebraic matrix, by the following steps.

The simplified, deterministic algorithm can be depicted algebraically as

$$v_{t+1} = v_t + \varphi_1 \otimes (p_i - x_i) + \varphi_2 \otimes (p_g - x_i)$$
$$x_{t+1} = x_t + v_{t+1}$$

And then, by substitution,

$$p = \frac{\varphi_1 p_i + \varphi_2 p_g}{\varphi}, \text{where } \varphi = \varphi_1 + \varphi_2$$
$$y = p - x_t$$

Given these transformations, the system can be written as

$$v_{t+1} = v_t + \varphi y_t$$
$$x_{t+1} = -v_t + (1 - \varphi)\, y_t$$

which can be represented in matrix form:

$$P_t = \begin{pmatrix} v_t \\ y_t \end{pmatrix}$$
$$M = \begin{pmatrix} 1 & \varphi \\ -1 & 1 - \varphi \end{pmatrix}$$

The velocity and position of the system are then calculated:

$$P_{t+1} = M \cdot P_t$$

or, to generalize:

$$P_t = M^t \cdot P_0$$

To control explosion, Clerc reasoned, one must ensure that M^t approaches a limit of zero as time approaches infinity. This is done by ensuring that the eigenvalues of M are less than 1 in module. Clerc accomplished this by application of a system of "constriction coefficients."

The simplest and most widely used version is Clerc's Type 1" constriction:

$$v_t + 1 = \chi\,(v_t + \varphi_1 \otimes (p_i - x_i) + \varphi_2 \otimes (p_g - x_i))$$
$$x_{t+1 = x^t + v^t + 1}$$

where the constriction coefficient χ is defined as

$$\chi = \frac{2k}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|}, \text{ where } \kappa \in [0,1] \text{ and } \varphi > 4.0 \text{ (usually } \kappa = 1.0 \text{ and } \varphi = 4.1.$$

Adding randomness, multidimensionality, and population size brings us back to the pseudocode example given in Table 6.1, the canonical particle swarm of the present time. It should be noted that the Type 1" constriction scheme is mathematically equivalent to the inertia-weight model, with appropriate values of coefficients.

1.4 Current Directions in Particle Swarm Research

Research on the particle swarm has fanned out in many different directions. For this chapter we will touch briefly on some developments and refer the reader to primary sources. This section will cover several specific problem domains to which the algorithm has been applied, and a later section will discuss variations in the algorithm itself.

1.4.1 Multiobjective optimization

It sometimes happens that a problem is defined in terms that require solving multiple problems simultaneously. Often the solutions to the various problems conflict with one another, and the best solution is a kind of compromise. The boundary between the two problems' solution regions tends to comprise multiple points; that is, there can be a set of solutions that equally well satisfies the conflicting demands. The set of solutions is called a *Pareto set*.

Optimization discussed to this point has searched for a single best solution to a problem. In using the particle swarm for multiobjective optimization, however, most researchers try to find a method where each particle represents a single point on the Pareto front and the overall solution is distributed across the entire population. To this writer, at least, the real problem of multiobjective optimization with particle swarms is the coordination of individuals. Human societies are full of reciprocity, roles, and specialization; the ordinary particle swarm has none. The work of a society may be seen as multiobjective optimization – you need to

heat your house as well as feed the family, as well as maintain transportation, etc. There are many objectives that must all be met to ensure survival and comfort. And this multiplicity of objectives is met through role specialization. Similarly, researchers are currently experimenting with ways for particles to "specialize" within the context of a multiobjective problem.

Mexican researcher Coello Coello [17, 18] has developed a particle swarm variation he calls MOPSO, for *m*ulti*o*bjective *p*article *s*warm *o*ptimization. MOPSO imposes a kind of grid on the search space, and maintains a global repository of nondominated vectors (e.g., vectors that best meet the optimization criteria) in each section of the grid. Each hypercube section of the grid is assigned a fitness that is inversely proportional to the number of points in it; this gives advantage to sections with fewer particles and helps in finding distributed solutions. Rather than having particles influence one another, solutions stored in the repository are substituted for the best neighbor; the repository member to influence a particle is chosen through roulette-wheel selection. Coello Coello reports that MOPSO performed "competitively" with other algorithms, in terms of the goodness of solutions found, and was computationally faster than other methods.

Parsopoulos and Vrahatis [59] adapted some techniques from genetic algorithms in order to develop a "multi-swarm" that could optimize multiple objectives simultaneously. The vector-evaluated genetic algorithm (Shaffer, [66]) subdivides the population into sections assigned to each of the objectives. Parsopoulos and Vrahatis's vector-evaluated particle swarm optimization (VEPSO) approach divided the population into two swarms for testing on two-objective problems. Each swarm was evaluated on one of the criteria, but social influence came from the best-performing particle in the other swarm.

Hu and Eberhart [37] modified the particle swarm in a different way to get it to optimize two simultaneous objectives. A different neighborhood was created on each iteration, using the K neighbors that were nearest in the fitness value space of the first objective function, where K was the neighborhood size. The neighbor to influence was then chosen as the one that had done the best so far on the second objective function. The particle's previous best, *pbest*, was only updated when a multiobjective solution was found that dominated the previous *pbest*.

Hu and Eberhart [37] note that there is no well-established measure for performance on multiobjective problems. Nevertheless, they report good results with their particle swarm on a set of standard problems from the literature.

In sum, several groups of researchers are approaching the multiobjective optimization case from different angles, slowly converging on a set of procedures for solving these knotty problems with particle swarms. These approaches necessitate some thinking about how the particles can interact with one another in order to blanket the solution space; each particle must find its appropriate role in the solution set.

The metaphor of human sociocognition should supply researchers with plenty of ideas for this interesting challenge. How do people coordinate their activities? Human social influence is something more than imitation – what are the factors that influence people to behave differently from one another, yet in concert? Consideration of these questions might result in simple yet effective advances in particle swarm theory.

1.4.2 Dynamic Problems

An obvious weakness with the particle swarm approach is its dependence on past successes as indicators of the locations of possible future successes. When the problem holds still, this can be sufficient, but many problems, especially in the real world, are constantly changing. Thus a particular type of difficult situation exists when the optima move around the parameter space.

In one of the earliest studies of this problem, Carlisle and Dozier [10] periodically replaced the previous best vector $\vec{p}_i$ with the current position $\vec{x}_i$, essentially "forgetting" past successes. They tested this modification using a dynamic version of the sphere function $f(\vec{x}) = \sqrt{\sum (g_j - x_j)^2}$, which is a simple measure of the distance from a particle's current position to the optimum $\vec{g}$, which was moved around the parameter space. Carlisle and Dozier tested two conditions, one in which the previous best was reset at regular intervals, and one where the previous best was reset when the particle detected that the optimum had moved some specified distance from its previous position. They also manipulated the speed with which the optimum moved.

In the no-reset control condition, the "full model" particle swarm was able to optimize well when the target moved slowly, but suffered when it moved fast. (A "social-only" version, using

$$v[i][j] = W \times v[i][j] + rand() \times (phi2) \times (p[g][d] - x[i][d])$$

also performed quite well in all tests.) Resetting more frequently resulted in the population performing better when the target was moving fast, but worse when it moved slowly; resetting less frequently had the opposite result. Resetting when a criterion had been breached also resulted in improvement, but nothing better than what was found using the simpler technique. In a later paper [11], these researchers took the approach of posting a "sentry" particle, which evaluated changes in the objective function in order to tell the swarm when to make adjustments.

Eberhart and Shi [25, 26] had similar ideas. After every 100 iterations, they reset the population using a technique similar to Carlisle and Dozier's, setting the "previous best" to the evaluation of the current position. They compared their results to previous work with evolutionary algorithms by Angeline [2] and Bäck [6] and found that error in tracking the optimum with the particle swarm was "several orders of magnitude less than that obtained by either Angeline or Bäck." Besides the three-dimensional sphere functions that the other authors had tested, Eberhart and Shi looked at higher dimensions and found that resetting the previous bests was sufficient to track optima in all cases.

Hu and Eberhart [38] reevaluated the best particle to detect changes in the position of the optimum. The *gbest* particle was evaluated using two different techniques. *Changed-gbest-value* meant that the *gbest* position was evaluated at every iteration, and change in its evaluation meant that the evaluation function had changed. In the *fixed-gbest-value method*, the *gbest*, and sometimes the second-best, particle was monitored; if it did not change for a certain number of iterations, e.g., if improvement ceased, then it was assumed that the function had changed, and a response was needed.

The response was to reinitialize part of the population. These researchers experimented with various proportions, as well as various numbers of iterations set to wait in the fixed-*gbest*-value condition before responding. Their results show that the populations were able to track dynamic optima quite well, especially when a small percent (10%) of the particles were rerandomized in response to change.

In sum, results so far are showing the particle swarm to be competitive with evolutionary computation methods for optimizing dynamically changing functions. Several tricks have been proposed for overcoming the particle swarm's reliance on memory for previous successes, which are not relevant if a problem has changed significantly.

1.5 Binary Particle Swarms

Much of the literature on genetic algorithms is focused on the binary implementation, probably because that was the focus of Holland's [35] pioneering work. At any rate, binary implementations can be useful for encoding discrete spaces as well as numeric ones—for instance, through noting the presence or absence of a feature. Thus it would seem useful to be able to run the particle swarm on binary spaces.

Kennedy and Eberhart [44] proposed a very simple adjustment to the canonical algorithm of the day (this was before the inertia weight and constriction coefficients had been discovered). The method involves a change in the way velocity is conceptualized. Whereas in real numbers velocity is a change in position, when optimizing variables in a discrete space velocity might better be thought of as a probability threshold.

Thus, where the real-numbered algorithm was given as

```
v[i][d] = khi × ( v[i][j] +
   rand()× phi1 × (p[i][d] – x[i][d]) +
   rand()× phi2 × (p[g][d] – x[i][d]) )
x[i][d] = x[i][d] + v[i][d]
```

the probability-threshold technique "squashes" $v[i][d]$ into the (0,1) interval, then generates a random number from a uniform distribution and compares it to the threshold. If the random number is less than the threshold, the variable $x[i][d]$ will be assigned the value of 1, and otherwise 0. A common sigmoid function is used for squashing:

$$S(x) = \frac{1}{1 + \exp(-x)}$$

Thus the binary algorithm can be written as

```
v[i][d] = khi × (v[i][j] +
   rand()× phi1 × (p[i][d] – x[i][d]) +
   rand()× phi2 × (p[g][d] – x[i][d]))
if rand() < S(v[i][d]) then x[i][d] = 1
else x[i][d]=0
```

Kennedy and Spears [48] compared this algorithm with several varieties of genetic algorithms (GAs), using Spears' multimodal random problem generator;

the binary particle swarm was the only algorithm of four that were tested (GA with mutation and no crossover, GA with crossover and no mutation, GA with both crossover and mutation, and particle swarm) to find the global optimum on every single trial. And it found the optimum fastest, except for the simplest condition, with low dimensionality and a small number of local optima, where it was slightly outperformed by the mutation-only GA (which was the worst in other conditions).

Agrafiotis and Cedeño [1, 12] used particle swarms to select a set of features that optimized some chemical criterion. When they wanted K features to be selected, they modified a particle swarm so that the locations of particles were treated as probabilities in a roulette wheel. The probabilities were calculated as

$$p_{ij} = \frac{x_{ij}^{\alpha}}{\sum x_{ij}^{\alpha}}$$

where α is a scaling factor they call *selection pressure*. Values of α greater than 1.0 lead to a tendency to favor the selection of highly fit features, while values below 1.0 give more approximately the same probability of selection to all features. These authors used α=2 in their experiments. The x_{ij} variables are constrained to the interval [0,1]. The roulette-wheel selection is performed K times to select K features.

Agrafiotis and Cedeño found that their particle swarm method performed better than simulated annealing in two ways. First, the fitness of the best models found by particle swarms were better than the best found by simulated annealing; as the authors comment, "although annealing does converge with greater precision [less variance in results], it converges to sub-optimal models that are perhaps more easily accessible in the fitness landscape." Second, particle swarms produced a more diverse set of good solutions; the authors explain this outcome by saying that the particle swarm "casts a wider net over the state space and [,] capitalizing on the parallel nature of the search," is able to find disparate solutions.

Other versions of the binary particle swarm have been suggested. Mohan and Al-kazemi (2001), for instance, have proposed an array of approaches. They call their binary algorithms DiPSO, for *di*screte *p*article *s*warm *o*ptimization. Their multiphase discrete particle swarm optimization model, called M-DiPSO, assigned coefficients to the three terms on the right-hand side of the velocity adjustment, and used a hill-climbing technique to select the next position. In their reported experiments, they used coefficients of (1, 1, –1) sometimes and (1, –1, 1) other times, depending in part on the *phase* of the algorithm. The phase is switched if some number of iterations (they used five) has gone by without improvement. In effect, the phase-switching means that sometimes particles are attracted to one of the two bests, and sometimes are repelled. Thus, by alternating phases, the system explores more thoroughly.

1.6 Topology and Influence

In the very first particle swarms, every individual was influenced by the very best performer in the population. Since only the best member of a particle's neighborhood actually influences it, this was equivalent to a social network where every individual was connected to every other one. Early experimentation, however,

found advantages with a topology where each individual was connected to its adjacent neighbors in the population. These two approaches were called *gbest* (for "global best") and *lbest* ("local best").

It was noted that the *gbest* topology had a tendency to converge very quickly on good problem solutions but had a (negative) tendency to fail to find the best region of the search space. *Lbest*, on the other hand, was slower but explored more fully, and typically ended up at a better optimum. The simple explanation is that the local-influence topology buffers and slows the flow of information so that individuals can search collaboratively with their neighbors in one region of the search space while other particles search in other regions.

Kennedy [41] experimented with some alternative social-network structures and found that varying the topology had a big effect on results. A paper by Watts and Strogatz [71] had shown that changing a small number of random links in a social network could drastically shorten the mean distance between nodes without affecting the degree of clustering, defined as the proportion of nodes' neighbors that were neighbors of one another.

Kennedy's 1999 paper arranged the particle swarm population of twenty particles into various configurations and then modified the configurations, or "sociometries," by randomly varying one connection. The sociometries used were classical ones [9], including

- the wheel, where one population member was connected to all the others, with no other connections in the population
- the ring, equivalent to *lbest* or a ring lattice, where all individuals were connected with their immediate neighbors
- the star, equivalent to *gbest*, with all individuals connected to all others, and
- random edges, with every individual randomly connected to two others

The small-world manipulations were not especially effective in that study, as the populations were really too small for that phenomenon to show itself. The differences between the various topologies, however, were quite noticeable, and further research explored that aspect of the algorithm more thoroughly.

It is useful to think for a minute about how information spreads in a particle swarm population. When a particle i finds a relatively good problem solution, it still may not be the best thing found in j's neighborhood (i and j being linked), and so j will not be affected by i's discovery. Eventually, though, if i has actually found a good region of the search space, its performance will improve as it explores that area, until it does become the best in j's neighborhood. Then j will be attracted to search the region between (and beyond) them, and may eventually find that i has indeed found a higher peak in the fitness landscape. In this case, j's performance will improve, its "previous best" will be in i's region, and j's other neighbors will be attracted to that region. It is perhaps overly facile to simply say that information has "spread" from i to j's neighbors.

Kennedy and Mendes [46] tested several thousand random graphs, sociometries that were optimized to meet some criteria, including average degree or number of links per individual, variance of degree (high or low), clustering, and

variance in clustering. They used two distinct measures of a particle swarm's problem-solving ability. The first, "performance," is the best function result obtained after some number of iterations. In order to compare across functions, these results were standardized, that is, they were transformed to give a mean of 0.0 and standard deviation of 1.0; results could then be averaged across problems. The second measure, called "proportion," was the proportion of trials in which a particular version of the algorithm met criteria from the literature that indicate that the global optimum has been found. For instance, in a multimodal problem there may be many local optima, the peaks of which will be relatively good results, but there is, in the test suite used in this research, only one globally best solution.

Kennedy and Mendes found that different topologies were successful depending on which measure was used. For instance, when looking at performance – best function result at 1,000 iterations – the best swarms comprised sociometries with high degree; that is, each individual had many connections to others. But the best swarms by proportion – the ability to find the global optimum – were mostly of moderate degree. The best by this measure had a degree of five (mean degrees of 3, 5, and 10 were tested in that study).

Kennedy and Mendes [47] included another modification of the particle swarm, which they called the *fully informed particle swarm (FIPS)*. Recall that Clerc analyzed a particle swarm of one particle, where the two terms added to the velocity were collapsed into one. He concluded that the coefficient φ should equal 4.1, with a χ constriction factor of approximately 0.7298, calculated from the formula given above. Since two terms are added to the velocity in a real particle swarm, i.e., the term representing the particle's own previous best and that representing the neighborhood best, the φ coefficient was divided by two, with half being assigned to each term.

Kennedy and Mendes noted that there was no reason the coefficient should not be subdivided even further. Thus they modified the particle swarm by adjusting the velocity using information drawn from all the neighbors, not just the best one. So where the canonical particle swarm can be depicted algebraically as

$$\vec{v}_i \leftarrow \chi\left(\vec{v}_i + U(0,\varphi_1) \otimes (\vec{p}_i - \vec{x}_i) + U(0,\varphi_2) \otimes (\vec{p}_g - \vec{x}_i)\right)$$
$$\vec{x}_i \leftarrow \vec{x}_i + \vec{v}_i$$

the FIPS algorithm is given as

$$\vec{v}_i \leftarrow \chi\left(\vec{v}_i + \sum_{n=1}^{N_i} \frac{U(0,\varphi) \otimes (\vec{p}_{nbr\,(n)} - \vec{x}_i)}{N_i}\right)$$
$$\vec{x}_i \leftarrow \vec{x}_i + \vec{v}_i$$

where *nbr(n)* is the particle's *n*th neighbor.

The FIPS algorithm does not perform very well at all with the *gbest* topology, or in fact with any with high degree in the population topology. With *lbest*, however, or with topologies where particles have very few (e.g., three) neighbors, FIPS performs better than the canonical particle swarm as measured by both performance and proportion in a suite of standard test functions.

Sociometry, then, was found to be a more important factor in a FIPS swarm than in one where information was taken only from the best neighbor. It seems intuitively obvious that information from many neighbors may include conflicts, since these neighbors may have found success in different regions of the search space; thus the averaged information is likely to be unhelpful. In versions using the best neighbor only, though, a bigger neighborhood is more likely to contain better information, as a bigger net is likely to catch a bigger fish. All in all, the FIPS particle swarm with appropriate sociometry outperformed the canonical version on all the testbed problems.

It is interesting to go back to the sociocognitive metaphor here for understanding and for new ideas. Ideas sweep through a population, one person to another, with "better" ideas – ones that explain the most with the least inconsistency – prevailing. But the variables that affect social influence are very complex. Theorists such as Latané focus on high-level factors, while others look "inside the head" at cognitive factors; one traditional social-psychological view looks at qualities of the message source, the message itself, and the recipient of the message [61].

In designing a particle swarm, then, it is not immediately obvious how one should code the interactions of individual particles. It is certainly reasonable to assume that people tend to accept information from experts and authorities, that is, people who seem to have achieved some success, but it is also reasonable to say that people accept information from people they know well, and that people accept information that fits well with their previous views. The interpersonal flow of information still does not have a well-fitted, comprehensive theory specific enough to be implemented in a computer program. Still, it seems that the particle swarm algorithm can and will be improved by integrating more of what is known about human social behavior. It is expected that the nature of the interactions of the particles will undergo important changes in years to come.

1.7 Gaussian Particle Swarms

The velocity-adjustment formula is arbitrary. It is entirely possible that a different formula could be used to move the particles back and forth around the average of their own successes and good points found by their neighbors. A few researchers have tampered with the formula, but not very much.

Kennedy [43] conducted an experiment to discover the distribution of the points that are searched by the canonical particle swarm. The individual and neighborhood bests were defined as constants, –10 and +10 in that study, and the formula was iterated a million times to produce a histogram. The histogram revealed that the algorithm searches a bell-shaped region centered halfway between the two best points, and extending significantly beyond them. (This last effect is often referred to as "overflying" the best points.) The distribution appeared to be a typical gaussian bell curve.

This result should not be surprising when one considers that the points are chosen by averaging pairs of random numbers. The central limit theorem would predict that the means would be normally distributed. This point was also noted by Secrest and Lamont [65] as well as Higashi and Iba [34].

Kennedy modified the algorithm by substituting a gaussian random number generator for the velocity change formula, using the midpoint between p[i][d] and

p[g][*d*] on each dimension as the mean of the distribution, and their difference as the standard deviation. Thus the resulting algorithm can be algebraically described as

$$\vec{x}_i \leftarrow G\left(\frac{\vec{p}_i + \vec{p}_g}{2}, \left|\vec{p}_i - \vec{p}_g\right|\right)$$

where *G(mean, s.d.)* is a gaussian random number generator.

Other versions were tested, for instance by using the mean of all neighbors, à la FIPS, for $\vec{p}_g$ in the gaussian formula just given. Some versions selected random neighbors from the population, some used the population best, and some used an "interaction probability" (IP) threshold of 0.50; this meant that a vector element was only changed if a probability test was passed. Half the time, then, *x*[*i*][*d*]=*p*[*i*][*d*].

The best performing version in that study was the random-neighbor, IP=0.50 version, though other versions were quite competitive. There did not seem to be any significant performance difference between the gaussian versions and the canonical algorithm with velocity.

It would seem, then, that the door is open for experimentation with new formulas for moving the particle. What is the best strategy for exploring a wide variety of problem spaces? The adjustment of velocity by traditional terms based on the differences may not be the best approach.

1.8 Particle Swarms and Evolutionary Computation

A reader familiar with evolutionary computation will have noted that the gaussian perturbation suggested here is similar to that employed in evolutionary programming and especially evolution strategies. Evolutionary programming does not employ crossover or recombination, but only mutation, which is usually gaussian or Cauchy distributed. Evolution strategies (ESs), however, do typically feature interaction between population members, which is called *recombination* and is considered metaphorically to resemble the mixing of phenotypes in sexual reproduction.

In fact, intermediate recombination in ES is very similar to the interactions of individuals in gaussian particle swarms. Parents are selected, similar to the individual particle and its best neighbor, and averages are calculated for each vector element. These means are then perturbed through gaussian mutation with a standard deviation that is evolved, in ES, as a vector of "strategy parameters" along with the object parameters that are being optimized [7].

This adaptation of strategy parameters in ES means that optimization of an *n*-dimensional problem requires at least a 2*n*-dimensional vector to be adapted iteratively, as there is a standard-deviation variable for every function parameter. Why isn't this necessary in the particle swarm? The answer is that the standard deviation of the "mutation" in the particle swarm is determined by the distance between the individual and its source of social influence. This is true whether the velocity formula is used, or gaussian randomness: the adaptation is inherent.

One other evolutionary computation paradigm uses consensus to determine step-size, and that is *differential evolution (DE)* [62]. In DE, a vector drawn from

a population is modified by some values that are derived from the differences between two other vectors that are randomly drawn from the population. Thus, as in the particle swarm, the variance of change, or step-size, is determined by the size of differences among population members.

In both ES and DE, problem solutions that will interact with one another are chosen at random. The particle swarm, however, has a topological structure such that each problem solution has a subset of the population with which it will share information. (This is arguable in the *gbest* versions.) Some ES researchers have experimented with fixed topologies [73], but what can this possibly mean in a simulation of evolution? It would seem to mean that two parents have two children, who then mate with one another and have two children, who then mate with one another and so on. If the neighborhood size exceeds two, then the incest is somewhat less intense, as siblings will only mate occasionally. But it is certainly not like anything in nature.

Thus we discover a chasm between the evolutionary methods and the ones we are calling *social*, which has to do with the persistence of the individual over time. Darwinisan evolution, at least as it is implemented in computer programs, functions through *selection*, differential reproduction probabilities depending on fitness. The social models (here we can include the ant swarms, cultural algorithm [63], and some of the fixed-topology ES versions just mentioned), on the other hand, feature the survival of individuals from one "generation" (the word is not appropriate here, and is used to make that point) to the next.

Evolution introduces a bias toward better problem solutions by allowing only the fitter population members to reproduce. The parents are, at least potentially, replaced at each turn, leaving offspring that are altered copies of themselves. The particle swarm implements a positive bias by a process that may be properly called *learning*; that is, the states of individuals are changed by their experiences, which include interactions with one another.

It can be argued that particle swarms instantiate a kind of selection, if one considers the previous bests, e.g., $\vec{p}_i$, as the parent individual, and the current location $\vec{x}_i$ as something like an offspring [4, 43]. Each iteration then presents an occasion for competition between the parent and the child, and if the child is better it kills the parent and takes its place. This view calls to mind (1+1)-ES, where a candidate solution is mutated, and if the descendant is better than the parent, then the descendent becomes the ancestor for the next generation.

But first – is this evolution at all? Selection cannot be taken to have occurred in every situation where a better exemplar is chosen over a lesser one, for instance (1+1)-ES.

Philosophically central to this issue is the question of what makes up an individual. We can imagine, for instance, a kind of biological kingdom where an orgasm gave its own life to its offspring and perished at the moment the torch was passed. In this case, observers would almost certainly develop a vocabulary that allowed them to track the vital "spirit" as it passed from generation to generation. In other words, if A transferred its life-force to B, and at the moment B accepted it A lost it, then B would probably be named A-Junior, or A-the-ninetieth, or something to indicate that the spirit of A was in this particular organism. The life-force passed through the generations would be considered "the individual," and the organism inhabited by it at the moment would be

some other kind of transient thing. The lack of such a concept seems to be what is missing in attempts to explain the particle swarm, and certain kinds of ES, as evolutionary processes. It is the concept that in describing an individual's sense of continuity from moment to moment, from day to day, is called *self* or *ego*.

The various EC paradigms have a tendency to blend together, being separated as much by sociological boundaries as technical differences. One method uses tournament selection, another uses roulette-wheel selection, by tradition as much as by necessity. Yet though there is much overlap, and much that can be learned by one paradigm from another, there are also some hard differences. I am arguing here that the use of selection distinguishes an evolutionary group of algorithms from another group in which individuals survive over iterations. I am also suggesting, gently, that some processes that have been labeled "evolutionary" are not. For instance, it seems very inelegant to try to explain how selection is at work when one parent gives birth to one offspring at a time and eventually perishes when it is replaced by its own child.

It seems more appropriate, in these cases, to view population members as persistent individuals that change over time – that learn. These are two different kinds of methods for biasing a population's performance toward improvement: reproduction allowing only for the fittest members of the population, versus change or learning in individuals over time. These two methods can be combined in an application, as Angeline showed in 1998, where he added selection to a particle swarm and found that performance was improved. But they are not variations on the same thing.

The "social" family of algorithms relies on the interactions between individuals, which in EC is called *crossover* or *recombination*. For instance, several researchers have written "cognition-only" particle swarm programs, where the last, social, term of the velocity adjustment was simply omitted. These versions are uniformly terrible at finding any optimum unless they are initialized near it and there is a clear monotonic gradient. It is the influence of neighbors that forces a particle to explore the space more widely when there is diversity in the population, and more narrowly when there is agreement. It is social influence that tells the particle where to look, and tells it how big its steps should be through the space. And the sociometric structure of the population helps determine how quickly it will converge on an optimum, as opposed to maintaining searches of diverse regions. In sum, the interactions among individuals are crucial to the "social" family of algorithms.

Why isn't crossover, for instance, a qualifying characteristic for membership in the "social" group? That is because social creatures don't die as soon as they socialize. Oh, we could find some kind of black-widow analogy, but we would be stretching it. The behaviors of truly social organisms are changed through their interactions with their conspeciates; this behavior change is called *learning*, and when behaviors are learned from others, it is called *social learning* [8].

1.9 Memes

It would appear that, in discussing the evolution of cognitive patterns in a population, we must be talking about the same thing that people who talk about

memes are talking about. Memes [28] are patterns, perhaps ideas, snatches of music, behavioral patterns, that seem to spread through a population, as if taking on a life of their own.

The evolutionary nature of memes seems self-evident, which should of course be a sign that something is wrong.

I imagine that a meme is like a schema in Holland's analysis of the behavior of genetic algorithms. On a ten-dimensional binary genetic representation, say, 1001010011, it may be that one subset of bits, say the 101 starting at the fourth site, confers some important fitness value and comprises some kind of logically interrelated genetic pattern.

In this case, then, the entire bitstring 1001010011 must be considered to be the cognitive state of some individual. As the meme spreads through the population, more and more individuals will find their fourth, fifth, and sixth mental slots filled with the meme-pattern 101. This appears to be the only way that a meme could be represented.

This view is fine for a static snapshot of the population. We see that some individuals contain or embody the meme, and others don't. Theoretically, what we expect to see in our simulation is some dynamic representation of the adoption of the meme by new population members (and eventually its replacement by some other meme).

But let us imagine that this population is modeled by a genetic algorithm (GA) of some sort. We have an immediate problem, which is that population members have to die. Whether we implement crossover or mutation, or both, we are expected to kill off the less fit members of the population and replace them with a new generation. Because the GA works by selection, it is necessary that only a subset produce offspring, and that the offspring replace the ancestors. It is simply not going to be possible to model the spread of memes in a population using a genetic algorithm.

Richard Dawkins [28] invented the concept of memes, or at least introduced the term to our vocabulary. So it is noteworthy that his innovative EC program, "Biomorphs," represents a kind of asexual evolution that utterly fails to mimic the supposed behavior of memes. A single Biomorph on the screen is selected, and clicked, and then a new generation of Biomorphs appears, all of which are descended from the one that was clicked on in the previous generation. They are all slightly mutated forms of the original, and the rest of the previous generation has disappeared, died, gone forever.

It does not seem reasonable to believe that memes are "evolutionary" at all, in any Darwinian sense. The evolution of ideas belongs to the second class of paradigms, the social methods. It is not hard at all to model memetic evolution, for instance, in a particle swarm, where individuals are manifest as vectors of variables, which can even be binary variables; in this case, the meme can be portrayed as a pattern in the vector, say 101 starting in the fourth site on a ten-dimensional bitstring. The spread of the meme is seen in changes in the states of population members, with the particular meme 101 moving through the social network from individual to individual as it is adopted by new population members.

This modeling is possible because individuals maintain their identities over time, and learn by changing. While a meme may be a distinct and easily identifiable pattern of behavior, it does not have a life of its own; it does not leap from

person to person like a virus (which memes are frequently compared to), infecting one mind after the other. No, the active agent is the individual, the teenage kid who adopts the behavior. Just as a language has no existence independent of its speakers – even writing is only a pattern of marks to someone can't read it – so the phenomena known as memes only have their existence in the states of those who participate in them, own them, use them, adopt them, provide habitation for them.

2 EXTERIORIZING THE PARTICLE SWARM

The particle swarm algorithm is only a recipe for solving problems, and does not need to be run in a computer program. At least two implementations demonstrate that some parts of the program can be run in the real world.

A major pharmaceutical company needed a medium for growing bacteria [16]; this involved finding an optimal mixture of ingredients for the particular organism. In order to solve this problem, Cockshott and Hartman mixed up some batches with random amounts of each ingredient, introduced the organism to it, and waited for it to grow. After some period of time, they measured the amount of the organism that had grown in each mixture, and, using that measure as an "evaluation function," calculated proportions of the ingredients, from the particle swarm formulas, to try in the next round of trials. In their research, time-steps were weeks long, and measurement occurred outside the computer, but the particle swarm recipe was followed literally.

Cockshott and Hartman report that the optimized ingredient mix they eventually derived was over twice as good as a mix found using traditional methods. Also, interestingly, the mix discovered through the particle swarm process was very different from other mixes that had been found; it lay outside what had been considered the "feasible" regions of the search space.

Kennedy [42] wrote a program that extended the particle population to include a human being. A user interface displayed a "problem," which was to find the desired pattern of squares on a grid similar to a checkerboard. Underlying the problem was an NK landscape function, which could be adjusted to make the problems more or less hard; the landscape could be monotonic, which would allow a simple greedy strategy of flipping colors one at a time, or local optima could be introduced. The user in the "exteriorized particle swarm" was also shown his own previous best pattern, as well as the best pattern so far found by a population of particles.

While no carefully designed experiment was conducted, informal testing indicated that the presence of the humans, who were allowed to use any cognitive strategy they wished, did not improve the performance of the swarm. Further, the global optimum was found at least as often by a computational particle as by a human one.

These instances show the particle swarm to be more than a type of computer program; it is a way of doing things. The algorithm is derived from observations of human behaviors, and can be turned around and used as a process for guiding human behavior, for instance for getting "out of the box," finding solutions to problems that have proven intractable.

2.1 The Future of the Particle Swarm

There is something satisfying in thinking about the particle swarm paradigm itself as a big particle swarm. The method originated in the integration of bird-flocking, neural networks, social psychology, evolutionary computation, and other diverse subjects, and continues to evolve through the blending of existing particle swarm theory with new topics and situations. In the particle swarm tradition, much of the work has been reported and discussed at conferences and symposia, and much of the research has been collaborative, as researchers explore their own ideas and adopt the ideas of their colleagues. This section will speculate on the future of particle swarms in terms of applications, tweaks, and theory.

2.1.1 Applications

The particle swarm is most often used as a tool for engineers, and its applications cover an extremely wide domain. As it is a method for optimizing vectors of variables, and many engineering problems require exactly that; as it is a fast and efficient method for problem-solving, with a minimum of parameters to adjust; and as it is very easy to code and maintain, and is unpatented and otherwise free of burden, the paradigm is used in many applications.

The most common use of particle swarms is in optimizing the weights in feedforward neural networks. These networks are often used for the analysis of complex, e.g., nonlinear, data. Since the problem space of the neural net typically contains many local optima, gradient descent approaches such as backpropagation of error are sometimes inadequate; the solution is highly dependent on the starting point.

Eberhart and Shi [24] extended the use of the particle swarm beyond the weight matrix, letting it optimize the structure of the network as well. Since each node in the standard feedforward network has a sigmoid transfer function, by using the function

$$output_i = \frac{1}{(1 + e^{-k \cdot input})}$$

Eberhart and Shi optimized the exponent coefficient k, which is the slope of the function, along with the network weights. This parameter was allowed to take on negative as well as positive values. If k became very small, the node could be eliminated. Thus, it is possible, using this method, to generate a network structure along with the weights.

Particle swarms have been used on problems in domains as diverse as reactive power and voltage control by a Japanese electric utility [76], diagnosis of human tremor [22], and multiple-beam antenna arrays [32]. Ujjin and Bentley [70] used particle swarm optimization in a worldwide web "recommender system," which helps users navigate through Internet entertainment and shopping sites. Xiao et al. [75] used the algorithm for clustering genes.

It is not possible to imagine the range of future applications of the particle swarm. Any problem that has multiple variables is a candidate for analysis with this approach.

2.1.2 Tweaks

The particle swarm algorithm has been modified in many ways, some of them fundamental and some trivial. For instance, the *V*max constant, which was necessary in versions without inertia or constriction factors, is now optional; some researchers still set a value for *V*max just to keep the particles from going "outside the solar system" [27]. Eberhart and his colleagues, for instance, commonly set *V*max equal to the range of the problem space. The inertia weight has undergone numerous adjustments: sometimes it decreases over time, and sometimes it is given a random value within a range. The constants that weight the two terms adjusting the velocity are usually equal, but not always, and sometimes larger or smaller values are used for them. Sometimes the velocity is modified by one term instead of two, sometimes by more than two.

Clerc's analysis seems to have shown the best values for some of the system constants, and contemporary particle swarm researchers usually adhere to his recommendations [15]. But, as has been seen, even these analytically derived values are sometimes tweaked.

Individuals in the particle swarm imitate successes: is this a realistic depiction of human behavior? It is obvious that much more interesting particle trajectories could be discovered by modeling processes from nature. Sometimes humans rebel against their groups, especially when these become too restrictive or when individuals are not attaining satisfaction. Human specialization has already been mentioned; this topic is especially relevant when discussing multiobjective optimization, where each particle represents a point in the Pareto set, and the pattern across the population comprises the entire problem solution. Even when a problem has only one objective, it is important to maintain diversity in the population in order to prevent stagnation and premature convergence.

The topology of the particle swarm has been modified quite a lot, as well. Several researchers have experimented with adaptive topologies [14], where some rules are programmed for pruning connections and adding others. If we think of all that is known about the complexities of human social networks, groups, and relationships, we see that there is a vast gold mine of things to try, ways to organize the population so that knowledge spreads realistically from person to person. Should links between particles be symmetrical? Should they be weighted probabilistically or made fuzzy? What are the best topological structures in general, and are there problem features that correlate with topology features? There are very many questions here, to be answered by some future researchers.

The particle swarm has been hybridized with various evolutionary computation and other methodologies. Angeline's incorporation of selection has been mentioned. Something similar was used by Naka et al. [53]. Zhang and Xie [77] hybridized the particle swarm with differential evolution, and reported good results. Parsopoulos and Vrahatis [60] used the nonlinear simplex method to initialize the particle swarm parameters. Clearly, tricks from other disciplines can be borrowed and integrated with the basic particle swarm framework. It is expected that future research in this area will open up new and powerful approaches to optimization and problem solving.

2.1.3. Theory

The particle swarm is not very well defined. For instance, Zhang and Xie [77] remark that the gaussian particle swarm is "a variety of evolution strategies." Of course, the present viewpoint is that some varieties of evolution strategies are particle swarms, but, point of view aside, it is clear that evolutionary methods overlap, and that the theory of particle swarm optimization is still in its infancy.

Clerc's remarkable analysis of the single particle's trajectory moved the field ahead significantly, and allowed the development of universal parameters that fit every problem and did not need to be adjusted. The next step is a comparable analysis at a higher level, looking at the whole population's trajectory through the search space. Perhaps one of the readers of this chapter will have an insight that makes such a global analysis possible. The analysis needs to explain not only the interactions of individual particles but also the global dynamics of the entire population, and needs to explain how graph-theory aspects of the population topology interact with convergence properties of the particles.

A theme of the current chapter is the richness of the metaphor of human behavior for informing developments in the digital implementations of the algorithm. The point of this is really that human beings are the most intelligent thing we know. We tend to define intelligence in terms of human behavior, whether intentionally or not. The human mind classifies, remembers, communicates, reasons, and empathizes better than any computer product known. And so the question is, what are the qualities of human behavior that make it so intelligent? The particle swarm approach ventures to guess that interpersonal interaction is an important part of it, and draws on the science of social psychology for inspiration. Programs that solve problems through interactions of computational entities are a kind of validation of social-psychological theorizing, and so the metaphor and the engineering tool inform one another.

What is really needed is a general theory of populations that includes evolutionary algorithms as well as social ones. It appears that a rather small toolbox can construct a great variety of problem-solving methods. For instance, gaussian perturbation can be seen in evolutionary programming, evolution strategies, and particle swarms, and the interactions of individuals in particle swarms greatly resemble crossover or recombination in evolutionary algorithms. But selection is different from learning. Topological linkage can be seen in particle swarms and some ES versions, and the type of interaction can be similar in those paradigms, e.g., gaussian mutation around a midpoint between neighbors or parents.

And what is essential for a particle swarm? It appears that the velocity formula is arbitrary, and it has been shown that it is not necessary to interact with the best neighbor. What is left? It appears that the essence of the particle swarm is a population of individuals that persist over time and learn from one another. Hopefully the future will provide a comprehensive theory that explains the variations in methods for such implementations.

REFERENCES

[1] D. K. Agrafiotis and W. Cedeño (2002): Feature selection for structure-activity correlation using binary particle swarms. *Journal of Medicinal Chemistry*, *45*, 1098–1107.

[2] P. J. Angeline (1997): Tracking Extrema in Dynamic Environments. *Evolutionary Programming*, pp. 335–345.

[3] P. Angeline (1998a): Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, (eds.), *Evolutionary Programming VII*, 601, 610. Berlin: Springer.

[4] P. J. Angeline (1998b): Using selection to improve particle swarm optimization. *IEEE International Conference on Evolutionary Computation*, Anchorage, AK, USA.

[5] S. Asch (1956): Studies of independence and conformity: I. A minority of one against a unanimous majority. *Psychological Monographs*, 70 (9).

[6] T. Bäck (1998): On the behavior of evolutionary algorithms in dynamic environments. In D. B. Fogel, H.-P. Schwefel, Th. Bäck, and X. Yao (eds.), *Proc. Fifth IEEE Conference on Evolutionary Computation (ICEC'98)*, Anchorage AK, pp. 446–451, IEEE Press, Piscataway, NJ.

[7] T. Bäck, F. Hoffmeister, and H. Schwefel (1991): A survey of evolution strategies. In Lashon B. Belew and Richard K. Booker (eds.), *Proc. 4th International Conference on Genetic Algorithms*, pp. 2–9, San Diego, CA, Morgan Kaufmann.

[8] A. Bandura (1986): Social Foundations of Thought and Action: A Social Cognitive Theory. Englewood Cliffs, NJ: Prentice-Hall.

[9] A. Bavelas (1950): Communication patterns in task-oriented groups. *Journal of the Acoustical Society of America*, *22*, 727–730.

[10] A. Carlisle and G. Dozier (2000): Adapting particle swarm optimization to dynamic environments. *Proc. Int. Conf. Artificial Intelligence*, 2000, 429–434, Las Vegas, NV, USA.

[11] A. Carlisle and G. Dozier (2002): Tracking Changing Extrema with Adaptive Particle Swarm Optimizer. *ISSCI, 2002 World Automation Congress*, Orlando, FL, USA, June, 2002.

[12] W. Cedeño and D. K. Agrafiotis (2003): Using particle swarms for the development of QSAR models based on k-nearest neighbor and kernel regression. *Journal of Computer-Aided Molecular Design*, *17*, 255–263.

[13] R. B. Cialdini (1984): Influence: The Psychology of Persuasion. Quill Publishing.

[14] M. Clerc (1999): The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization. *Congress on Evolutionary Computation*, Washington, D. C., pp. 1951–1957.

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

[16] A. B. Cockshott and B. E. Hartman (2001): Improving the fermentation medium for Echinocandin B production. Part II: Particle swarm optimization. Process Biochemistry, *36*, 661–669.

[17] C. A. Coello Coello and S. Lechuga (2001): MOPSO: A Proposal for Multiple Objective Particle Swarm Optimization. Technical Report EVOCINV-01-2001, Evolutionary Computation Group at CINVESTAV, Sección de Computación, Departamento de Ingeniería Eléctrica, CINVESTAV-IPN, México.
[18] C. A. Coello Coello and M. S. Lechuga (2002): MOPSO: A proposal for multiple objective particle swarm optimization. *IEEE Congress on Evolutionary Computation,* 2002, Honolulu, HI, USA.
[19] R. S. Crutchfield (1955): Conformity and character. *American Psychologist, 10*, 191–198.
[20] R. Dawkins (1989): The Selfish Gene, 2nd ed. Oxford: Oxford University Press.
[21] M. Deutsch and H. B. Gerard (1955): A study of normative and informational social influences upon individual judgment. *Journal of Abnormal and Social Psychology*, *51*, 629–636.
[22] R. C. Eberhart and X. Hu (1999): Human tremor analysis using particle swarm optimization. *Proc. Congress on Evolutionary Computation 1999*, Washington, D. C. 1927–1930. Piscataway, NJ: IEEE Service Center.
[23] R. C. Eberhart and Y. Shi (2000): Comparing inertia weights and constriction factors in particle swarm optimization. *Proc. CEC 2000*, San Diego, CA, pp. 84–88.
[24] R. C. Eberhart and Y. Shi (1998): Evolving artificial neural networks. *Proc. 1998 Int. Conf. Neural Networks and Brain*, Beijing, P. R. C., PL5–PL13.
[25] R. C. Eberhart and Y. Shi (2001a): Tracking and optimizing dynamic systems with particle swarms. *Proc. Congress on Evolutionary Computation 2001*, Seoul, Korea. Piscataway, NJ: IEEE Service Center.
[26] R. C. Eberhart and Y. Shi (2001b): Particle swarm optimization: developments, applications and resources. *Proc. Congress on Evolutionary Computation 2001*, Seoul, Korea. Piscataway, NJ: IEEE Service Center.
[27] R. C. Eberhart (2003): Introduction to particle swarm optimization (tutorial). *IEEE Swarm Intelligence Symposium*, Indianapolis, IN, USA.
[28] A. E. Eiben and C. A. Schippers (1998): On evolutionary exploration and exploitation. Fundamenta Informaticae. IOS Press.
[29] J. E. Fieldsend and S. Singh (2002): A multi-objective algorithm based upon particle swarm optimisation, an efficient data structure and turbulence. *Proc. 2002 U.K. Workshop on Computational Intelligence* (Birmingham, UK, 2–4 Sept. 2002), pp. 37–44.
[30] L. Festinger (1957): A Theory of Cognitive Dissonance. Evanston IL: Row, Peterson.
[31] L. Festinger (1954/1999): Social communication and cognition: A very preliminary and highly tentative draft. In E. Harmon-Jones and J. Mills (eds.), Cognitive Dissonance: Progress on a Pivotal Theory in Social Psychology. Washington D. C.: AP Publishing.
[32] D. Gies and Y. Rahmat-Samii (2003): Reconfigurable array design using parallel particle swarm optimization. *Proceedings of 2003 IEEE Antennas and Propagation Symposium* (in press).
[33] F. Heppner and U. Grenander (1990): A stochastic nonlinear model for coordinated bird flocks. In S. Krasner (ed.), The Ubiquity of Chaos. Washington, D. C.: AAAS Publications.

[34] N. Higashi and H. Iba (2003): Particle swarm optimization with gaussian mutation. *Proc. IEEE Swarm Intelligence Symposium 2003 (SIS 2003)*, Indianapolis, IN, USA, pp. 72–79.

[35] J. H. Holland (1975): Adaptation in Natural and Artificial Systems. Ann Arbor: The University of Michigan Press.

[36] C. Hovland (1982): Communication and Persuasion. New York: Greenwood.

[37] X. Hu and R. C. Eberhart (2002a): Multiobjective optimization using dynamic neighborhood particle swarm optimization. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002)*, Honolulu, HI, USA, pp. 1677–1681.

[38] X. Hu and R. C. Eberhart (2002b): Adaptive particle swarm optimization: detection and response to dynamic systems. *IEEE Congress on Evolutionary Computation*, Honolulu, HI, USA.

[39] X. Hu (2002): Multiobjective optimization using dynamic neighborhood particle swarm optimization. *IEEE Congress on Evolutionary Computation*, Honolulu, HI, USA.

[40] J. Kennedy (1998): The behavior of particles. Evolutionary Programming VII: *Proc. Seventh Annual Conference on Evolutionary Programming*, San Diego, CA, pp. 581–589.

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

[42] J. Kennedy (2000): Human and Computer Learning Together in the Exteriorized Particle Swarm. Socially Intelligent Agents: The Human in the Loop, pp. 83–89. Technical Report FS-00-04, AAAI Press.

[43] J. Kennedy (2003): Bare bones particle swarms. *Proc. IEEE Swarm Intelligence Symposium 2003 (SIS 2003)*, Indianapolis, IN, USA, 80–87.

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

[45] J. Kennedy and R. C. Eberhart (1995): Particle swarm optimization. *Proc. IEEE Int. Conf. on Neural Networks*, *4*, 1942–1948. Piscataway, NJ: IEEE Service Center.

[46] J. Kennedy and R. Mendes (2002): Population structure and particle swarm performance. *IEEE Congress on Evolutionary Computation*, Honolulu, HI, USA.

[47] J. Kennedy and R. Mendes (2003): Neighborhood topologies in fully-informed and best-of-neighborhood particle swarms. *In Proc. 2003 IEEE SMC Workshop on Soft Computing in Industrial Applications (SMCia03)*, Binghamton, NY.

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

[49] B. Latané (1981): The psychology of social impact. American Psychologist, *36*, 343–356.

[50] J. M. Levine, L. B. Resnick, and E. T. Higgins (1993): Social foundations of cognition. *Annual Review of Psychology*, *44*, 585–612.

[51] E. F. Loftus and K. Ketcham (1994): The Myth of Repressed Memory: False Memories and Allegations of Sexual Abuse. New York: St. Martin's Press.
[52] C. K. Mohan and B. Al-kazemi (2001): Discrete particle swarm optimization. *Proc. Workshop on Particle Swarm Optimization*. Indianapolis, IN: Purdue School of Engineering and Technology, IUPUI (in press).
[53] S. Naka, T. Genji, K. Miyazato, and Y. Fukuyama (2002): Hybrid particle swarm optimization based distribution state estimation using constriction factor approach. *Proc. Joint 1st International Conference on Soft Computing and Intelligent Systems and 3rd International Symposium on Advanced Intelligent Systems (SCIS & ISIS)*.
[54] A. Newell and H. Simon (1963): GPS: A program that simulates human thought. In Feigenbaum and Feldman. (ed.), Computers and Thought. McGraw-Hill, New York.
[55] R. E. Nisbett and D. W. Wilson (1977): Telling more than we can know: Verbal reports on mental processes. *Psychological Review*, *84*, 231–259.
[56] A. Nowak, J. Szamrej, and B. Latané (1990): From private attitude to public opinion: A dynamic theory of social impact. *Psychological Review*, *97*, 362–376.
[57] E. Ozcan and C. Mohan (1999): Particle swarm optimization: surfing the waves. *Proc. 1999 Congress on Evolutionary Computation*, 1939–1944. Piscataway, NJ: IEEE Service Center.
[58] E. Ozcan and C. K. Mohan (1998): Analysis of a simple particle swarm optimization system. *Intelligent Engineering Systems Through Artificial Neural Networks*, *8*, 253–258.
[59] K. E. Parsopoulos and M. N. Vrahatis (2002a): Particle swarm optimization method in multiobjective problems, *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC 2002)*, pp. 603–607.
[60] K. E. Parsopoulos and M. N. Vrahatis (2002b): Initializing the particle swarm optimizer using the nonlinear simplex method. In A. Grmela and N. E. Mastorakis (eds), Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation, pp. 216–221. WSEAS Press.
[61] R. E. Petty and J. T. Cacioppo (1981): Attitudes and persuasion: Classic and contemporary approaches. Dubuque, IA: Wm. C. Brown.
[62] K. V. Price (1999): An introduction to differential evolution. In D. W. Corne, M. Dorigo, F. Glover (eds), New Ideas in Optimization. McGraw Hill.
[63] R. G. Reynolds (1994): An introduction to cultural algorithms. *Proc. Third Annual Conference on Evolutionary Programming*, pp. 131–139.
[64] C. W. Reynolds (1987): Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, *21*, 25–34.
[65] B. R. Secrest and G. B. Lamont (2003): Visualizing particle swarm optimization—gaussian particle swarm optimization. *Proc. IEEE Swarm Intelligence Symposium 2003 (SIS 2003)*, Indianapolis, IN, USA, pp. 198–204.
[66] J. D. Schaffer (1985): Multiple objective optimization with vector evaluated genetic algorithms. In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pp. 93–100.
[67] M. Sherif (1936): The Psychology Of Social Norms. New York: Harper Brothers.

[68] Y. Shi and R. C. Eberhart (1998): Parameter selection in particle swarm optimization. *Proc. Seventh Annual Conference on Evolutionary Programming*, pp. 591–601.

[69] M. Tomasello (1999): The Cultural Origins of Human Cognition. Cambridge, MA: Harvard University Press.

[70] S. Ujjin and P. J. Bentley (2003): Particle swarm optimization recommender system. I*n Proc. IEEE Swarm Intelligence Symposium 2003*, Indianapolis, IN, USA.

[71] D. Watts and S. Strogatz (1998): Collective dynamics of small-world networks. *Nature, 363*:202–204.

[72] D. M. Wegner (2002): The Illusion of Conscious Will. Cambridge, MA: The MIT Press.

[73] K. Weinert, J. Mehnen, and G. Rudolph (2001): Dynamic Neighborhood Structures in Parallel Evolution Strategies (Technical Report). Reihe CI 112/01, SFB 531, University of Dortmund.

[74] S. Wolfram (1994): Cellular Automata and Complexity: Collected Papers. Reading, MA: Addison-Wesley.

[75] X. Xiao, R. Dow, R. C. Eberhart, B. Miled, and R. J. Oppelt (2003): Gene clustering using self-organizing maps and particle swarm optimization. *Second IEEE International Workshop on High Performance Computational Biology*, Nice, France.

[76] H. Yoshida, Y. Fukuyama, S. Takayama, and Y. Nakanishi (1999): A particle swarm optimization for reactive power and voltage control in electric power systems considering voltage security assessment. *1999 IEEE International Conference on Systems, Man, and Cybernetics*, *6*, 502.

[77] W. J. Zhang and X. F. Xie (2003): DEPSO: hybrid particle swarm with differential evolution operator. *IEEE Int. Conf. on Systems, Man & Cybernetics (SMCC)*, Washington, D. C. USA.