chapter

seven

The Particle Swarm

This chapter introduces the particle swarm in its binary and real-numbered forms. The book so far has been preparing a context, describing related paradigms in computer science and social science, discussing culture and norms and language and other scientific and philosophical developments that, if we have been successful, will make the particle swarm seem like an obvious thing to propose.

The Adaptive Culture Model in the previous chapter hints at what can happen as a result of the simplest imaginable interactions of the simplest imaginable agents—if these can even be called "agents." Given a large space of possibilities, the population is often able to find multivariate solutions, patterns that solve problems, through a stripped-down form of social interaction.

It is worth emphasizing that individuals in the culture model are not *trying* to solve problems. They are only following the simple rules of the algorithm, which say nothing about the existence of a problem or how to solve it. Yet through reciprocal social influence each individual betters its "fitness" (the term is less appropriate here than in discussion of evolutionary algorithms), and the performance of the population improves. We would not say that the adaptive culture algorithm is an especially powerful way to solve problems, but it is a good introduction to some social algorithms that are.

The particle swarm algorithm is introduced here in terms of social and cognitive behavior, though it is widely used as a problem-solving method in engineering and computer science. We have discussed binary encoding of problems, and the first version of the particle swarm we present here is designed to work in a binary search space. Later in the chapter we introduce the more commonly used version, which operates in a space of real numbers. ■

Sociocognitive Underpinnings: Evaluate, Compare, and Imitate

A very simple sociocognitive theory underlies the Adaptive Culture Model and particle swarms. We theorize that the process of cultural adaptation comprises a high-level component, seen in the formation of patterns across individuals and the ability to solve problems, and a low-level component, the actual and probably universal behaviors of individuals, which can be summarized in terms of three principles (Kennedy, 1998):

- Evaluate
- Compare
- Imitate

Evaluate

The tendency to evaluate stimuli—to rate them as positive or negative, attractive or repulsive—is perhaps the most ubiquitous behavioral characteristic of living organisms. Even the bacterium becomes agitated, running and tumbling, when the environment is noxious. Learning cannot occur unless the organism can evaluate, can distinguish features of the environment that attract and features that repel, can tell good from bad. From this point of view, learning could even be defined as a change that enables the organism to improve the average evaluation of its environment.

Compare

Festinger's social comparison theory (1954) described some of the ways that people use others as a standard for measuring themselves, and how the comparisons to others may serve as a kind of motivation to learn and change. Festinger's theory in its original form was not stated in a way that was easily tested or falsified, and a few of the predictions generated by the theory have not been confirmed, but in general it has served as a backbone for subsequent social-psychological theories. In almost everything we think and do, we judge ourselves through comparison with others, whether in evaluating our looks, wealth, humor, intelligence (note that IQ scales are normed to a population average; in other words,

your score tells you how you compare to others—which is really the point, isn't it?), or other aspects of opinion and ability. Individuals in the Adaptive Culture Model—and in particle swarms—compare themselves with their neighbors on the critical measure and imitate only those neighbors who are superior to themselves. The standards for social behaviors are set by comparison to others.

Imitate

You would think that imitation would be everywhere in nature; it is such an effective way to learn to do things. Yet, as Lorenz has pointed out, very few animals are capable of real imitation; in fact, he asserts that only humans and some birds are capable of it. Some slight variations of social learning are found among other species, but none compare to our ability to mimic one another. While "monkey see, monkey do," well describes the imitative behavior of our cousins, human imitation comprises taking the perspective of the other person, not only imitating a behavior but realizing its purpose, executing the behavior when it is appropriate. In *The Cultural Origins of Human Cognition,* Michael Tomasello argues that social learning of several kinds occurs in chimpanzees, but true imitation learning, if it occurs at all, is rare. For instance, an individual's use of an object as a tool may call another individual's attention to the object; this second individual may use the same object, but in a different way. True imitation is central to human sociality, and it is central to the acquisition and maintenance of mental abilities.

The three principles of evaluating, comparing, and imitating may be combined, even in simplified social beings in computer programs, enabling them to adapt to complex environmental challenges, solving extremely hard problems. Our view diverges from the cognitive viewpoint in that nothing besides evaluation, comparison, and imitation takes place *within* the individual; mind is not found in covert, private chambers hidden away inside the individual, but exists out in the open; it is a public phenomenon.

A Model of Binary Decision

Consider a bare-bones individual, a simple being with only one thing on its mind, one set of decisions to make, yes/no or true/false, binary decisions, but very subtle decisions, where it is hard to decide which choices

to make. For each decision, this supersimplified individual can be in one state or the other, either in the yes state, which we will represent with a 1, or the no = 0 state. It is surrounded by other yes/no individuals, who are also trying to decide. Should I say yes? Should I say no? They all want to make the best choices.

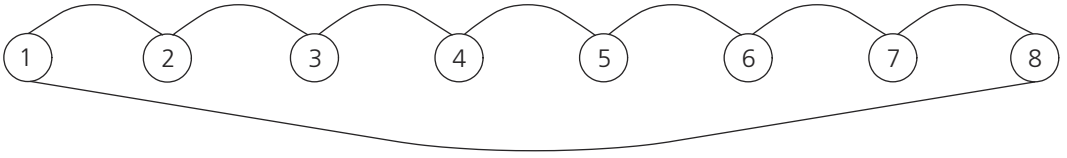
Two important kinds of information are available to these primitive beings. The first is their own experience; that is, they have tried the choices and know which state has been better so far, and they know how good it was. But these social beings have a second consideration; they have knowledge of how the other individuals around them have performed. In fact they are so simple that all they know is which choices their neighbors have found most positive so far and how positive the best pattern of choices was. If these stripped-down beings are anything like people, they know how their neighbors have done by observing them and by talking with them about their experiences.

These two types of information correspond to Boyd and Richerson's individual learning and cultural transmission. The probability that the individual will choose "yes" for any of the decisions is a function of how successful the "yes" choice has been for them in the past relative to "no." The decision is also affected by social influence, though the exact rule in humans is admittedly not so clear. Social impact theory states that the individual's binary decisions will tend to agree with the opinion held by the majority of others, weighted by strength and proximity. But even that rule is somewhat vague, given ambiguities in the concepts of strength and proximity.

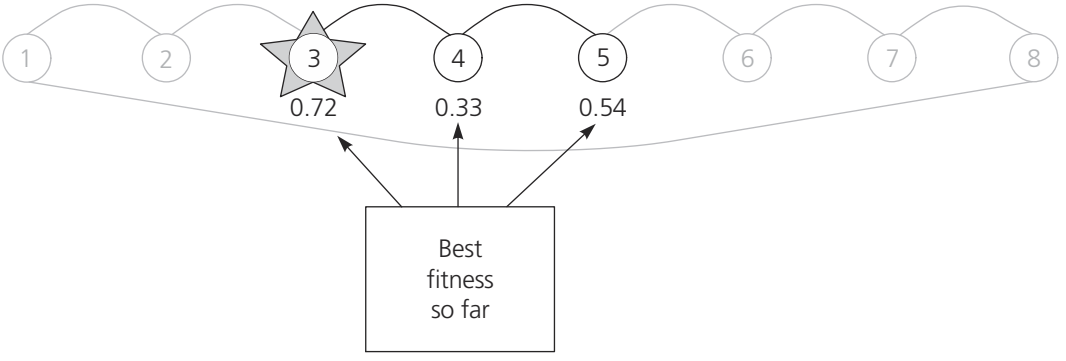
For the present introductory model we will just say that individuals tend to be influenced by the best success of anyone they are connected to, the member of their sociometric neighborhood that has had the most success so far. While we admit this is an oversimplification, it has a kernel of truth that justifies the parsimony it brings to the model.

Individuals can be connected to one another according to a great number of schemes, some of which will be mentioned in Chapter 8. Most particle swarm implementations use one of two simple sociometric principles (see Figure 7.1). The first, called *gbest,* conceptually connects all members of the population to one another. The effect of this is that each particle is influenced by the very best performance of any member of the entire population. The second, called *lbest* (*g* and *l* stand for "global" and "local"), creates a neighborhood for each individual comprising itself and its *k* nearest neighbors in the population. For instance, if $k = 2$, then each individual i will be influenced by the best performance among a group made up of particles $i - 1$, i, and $i + 1$. Different neighborhood topologies may result in somewhat different kinds of effects. Unless stated

The "lbest" neighborhood with $k = 2$. Each individual's neighborhood contains itself and its two adjacent neighbors. The first and last are connected.



Individual #3 has found the best position so far in #4's neighborhood. Therefore, #4's velocity will be adjusted toward #3's previous best position and #4's own previous best position.



The "gbest" neighborhood. Assuming that #3 has found the best fitness so far in the entire population, all others' velocities will be attracted toward its previous best position.

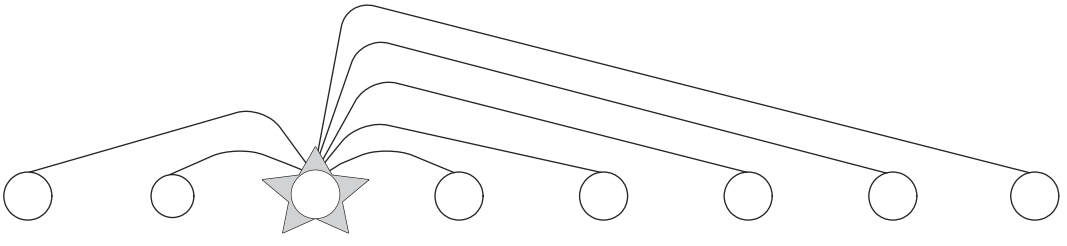


Figure 7.1 The two most common types of neighborhoods.

otherwise, the following discussions will presume lbest neighborhoods with $k = 2$ (sometimes described as "neighborhood = 3").

In a sociocognitive instance the individual must arrange an array of decisions or judgments in such a way that they all fit together, what we call "making sense" or "understanding" things. The individual must be able to evaluate, compare, and imitate a number of binary choices simultaneously.

Evaluation of binary strings can be accomplished in one step. In the psychological case, that is, if we are talking about humans, we can again use the concept of cognitive dissonance to evoke the sense of tension that exists when an array of decisions contains inconsistencies. We experience the state as discomfort and are motivated to change something to reduce the tension, to improve the evaluation. Dissonance as described by Festinger provides a single measure of cognitive evaluation, exactly as "fitness" is a single measure of genetic or phenotypic goodness.

How do we improve cognitive fitness? Of course there are plenty of theories about this. In Ajzen and Fishbein's *Reasoned Action Model,* (1980) *intent* is seen as a function of two kinds of things that should be getting familiar by now (see Figure 7.2). On the one hand, intent is affected by the person's *attitude* toward the behavior; for instance, if they believe violence is harmful or immoral, then they may intend not to act violently. This attitude is formed, in Ajzen (pronounced "eye-zen") and Fishbein's theory, by a linear combination of beliefs that the behavior will result in some outcomes (b_i) times the individual's evaluation of those outcomes (e_i):

$$A_o = \sum_{i=1}^{n} b_i e_i$$

This kind of expectancy-value model of attitude has existed in some form for many years, and we will not criticize its linearity or asymptotic issues here (never mind the decades-old debate about summing versus averaging). We are interested in the fact that intent has a second cause, which Ajzen and Fishbein call the *subjective norm.* The subjective norm regarding a behavior is also built up, in their theory, as a linear sum of products, but this time the factors entering into the formula are social. The individual's subjective norm toward a behavior is a sum of the products of their beliefs that certain others think they should or should not perform the behavior, multiplied by the motivation to comply with each of those others:

$$SN_o = \sum_{i=1}^{n} b_i m_i$$

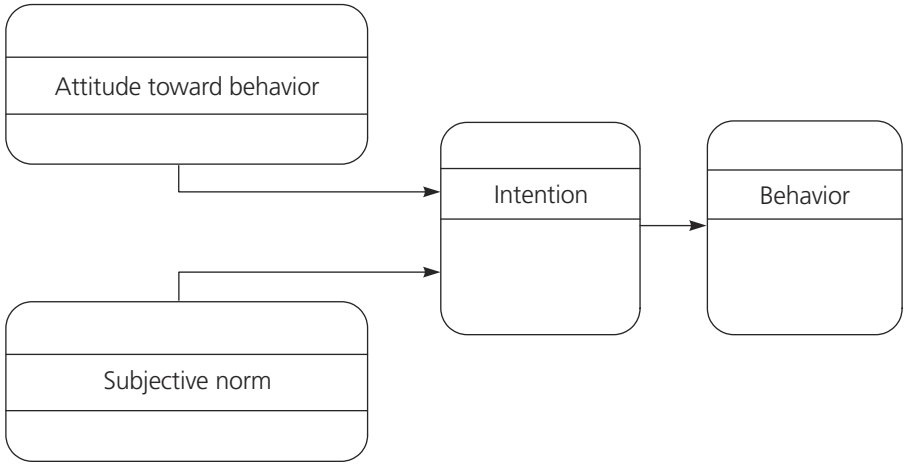


Figure 7.2 According to the Reasoned Action Model, behavior is a function of intention and only remotely of the individual's attitude about the behavior.

To point out the obvious, these two components of the theory of reasoned action map easily onto the components of Boyd and Richerson's cultural transmission model; that is, there is an individual term (individual learning or attitude toward a behavior) and a social term (cultural transmission or subjective norm). These two kinds of concepts are found in other theories as well and are represented in our decision model as the two terms that make up the change formula. We theorize that the coexistence of these two modes of knowledge, that is, knowledge acquired by the senses through experience in the world and knowledge acquired from others, gives humans the intellectual advantage; it is the source of our intelligence.

Besides their past experience and inputs from the social environment, another factor that affects the individual's decision is their current propensity or position regarding the issue. They may start with a strongly negative attitude and have subsequent positive experiences regarding the choice or attitude object—but still have a negative feeling about it. The positive experiences may make the individual *more* likely to choose the positive alternative, but in order to shift the individual's general propensity into the positive domain, the decision threshold would still have to shift upwards. If the individual's initial position is extreme, the probability is lower of its changing—for one thing, the individual is less likely to try the other alternative.

In mathematical terms, we are proposing a model wherein the probability of an individual's deciding yes or no, true or false, or making some

other binary decision, is a function of personal and social factors (Kennedy and Eberhart, 1997):

$$P(x_{id}(t) = 1) = f(x_{id}(t-1), v_{id}(t-1), p_{id}, p_{gd})$$

where

- $P(x_{id}(t)=1)$ is the probability that individual i will choose 1 (of course the probability of their making the zero choice is $1 - P$) for the bit at the dth site on the bitstring
- $x_{id}(t)$ is the current state of the bitstring site d of individual i
- t means the current time step, and $t - 1$ is the previous step
- $v_{id}(t - 1)$is a measure of the individual's predisposition or current probability of deciding 1
- p_{id} is the best state found so far, for example, it is 1 if the individual's best success occurred when x_{id} was 1 and 0 if it was 0
- p_{gd} is the neighborhood best, again 1 if the best success attained by any member of the neighborhood was when it was in the 1 state and 0 otherwise

The decisions themselves will be stochastic, if for no better theoretical reason than that we never know all the forces involved—it is very unlikely that any decision is made based solely on isolated facts pertaining directly to that decision alone. A lot of randomness allows exploration of new possibilities, and a little bit allows exploitation by testing patterns similar to the best one found so far; thus we can balance between those two modes of search by adjusting the uncertainty of decisions.

The parameter $v_{id}(t)$, an individual's predisposition to make one or the other choice, will determine a probability threshold. If $v_{id}(t)$ is higher, the individual is more likely to choose 1, and lower values favor the 0 choice. Such a threshold needs to stay in the range [0.0, 1.0]. We have already seen one straightforward function for accomplishing this, when we talked about neural networks. The sigmoid function

$$s(v_{id}) = \frac{1}{1+\exp(-v_{id})}$$

squashes its input into the requisite range and has properties that make it agreeable to being used as a probability threshold (though there is nothing magical about this particular function).

We wish to adjust the individual's disposition toward the successes of the individual and the community. To do that we construct a formula for each v_{id} in the current time step that will be some function of the difference between the individual's current state or position and the best points found so far by itself and by its neighbors. We want to favor the best position, but not so much that the individual ceases searching prematurely. If we simply added ($p_{id} - x_{id}(t-1)$) and ($p_{gd} - x_{id}(t-1)$) to $v_{id}(t)$, it would move upward when the difference between the individual's previous best and most recent states, or the difference between the neighborhood's best and the individual's most recent states, equaled 1, and would be attracted downward if either difference equaled −1. The probability threshold moves upward when the bests are ones and downward when they are zeroes.

In any situation we do not know whether the individual-learning or the social-influence terms should be stronger; if we weight them both with random numbers, then sometimes the effect of one, and sometimes the other, will be stronger. We use the symbol φ (the Greek letter phi) to represent a positive random number drawn from a uniform distribution with a predefined upper limit. In the binary version the limit is somewhat arbitrary, and it is often set so that the two φ limits sum to 4.0. Thus the formula for binary decision is

$$v_{id}(t) = v_{id}(t-1) + \varphi_1(p_{id} - x_{id}(t-1)) + \varphi_2(p_{gd} - x_{id}(t-1))$$

$$\text{if } \rho_{id} < s(v_{id}(t)) \text{ then } x_{id}(t) = 1; \text{ else } x_{id}(t) = 0$$

where ρ_{id} is a vector of random numbers, drawn from a uniform distribution between 0.0 and 1.0. These formulas are iterated repeatedly over each dimension of each individual, testing every time to see if the current value of x_{id} results in a better evaluation than p_{id}, which will be updated if it does. Boyd and Richerson varied the relative weighting of individual experience and social transmission according to some theoretical suggestions; the current model acknowledges the differential effects of the two forces without preconceptions about their relative importance. Sometimes decisions are based more on an individual's personal experience and sometimes on their perception of what other people believe, and either kind of information will dominate sometimes.

One more thing: we can limit v_{id} so that $s(v_{id})$ does not approach too closely to 0.0 or 1.0; this ensures that there is always some chance of a bit flipping (we also don't want v_i moving toward infinity and overloading the exponential function!). A constant parameter $V_{\max}$ can be set at the

start of a trial to limit the range of v_{id}. In practice, $V_{\max}$ is often set at ± 4.0, so that there is always at least a chance of $s(V_{\max}) \approx 0.0180$ that a bit will change state. In this binary model, $V_{\max}$ functions similarly to mutation rate in genetic algorithms.

Individuals make their decision in a population, where they are influenced by the successes of their neighbors. As each individual's decision is affected by $(p_{gd} - x_{id}(t-1))$, that is, (usually) some other individual's success, they influence one another and tend to move toward a common position. As an individual begins to approximate its neighbor's best position, it may perform better and influence *its* neighbors, and on and on; good decisions spread through the population. We are comfortable calling this the formation of a culture in a computational population.

In this section we have developed an extremely parsimonious model of binary choice as a function of individual learning and social influence. Individuals tend to gravitate probabilistically toward the decisions that have resulted in successes for themselves and their colleagues. The result is optimization of each individual's decision vector and convergence of the population on an optimal pattern of choices.

The entire algorithm, maximizing goodness, is shown in pseudocode:

```
Loop
  For i = 1 to number of individuals
    if G(x_i) > G(p_i) then do          //G() evaluates goodness
      For d = 1 to dimensions
        p_id = x_id                     //p_id is best so far
      Next d
    End do

    g = i                               //arbitrary
    For j = indexes of neighbors
      If G(p_j) > G(p_g) then g = j     //g is index of best performer
                                        in the neighborhood
    Next j
    For d = 1 to number of dimensions
      v_i(t) = v_id(t-1) + φ1(p_id - x_id(t-1)) + φ2(p_gd - x_id(t-1))
      v_id ∈ (-V_max, + V_max)
      if ρ_id < s(v_id(t)) then x_id(t) = 1; else x_id(t) = 0;
    Next d
  Next i
Until criterion
```

Testing the Binary Algorithm with the De Jong Test Suite

It may seem confusing to jump back and forth between "cognitive models" and "test functions." We are maintaining a generous definition of cognitive models, given the lack of consensus among psychologists about the internal structure of the mechanisms of thought. Thus, to us, a cognitive model is just like any other multidimensional problem where elements interact with one another in a combination possessing some measurable goodness.

Kennedy and Eberhart (1997) tested the binary particle swarm using a binary-coded version of the classic De Jong suite of test problems. The binary versions had already been prepared for experimentation with binary genetic algorithms, so importing them into a binary particle swarm program was straightforward. A population size of 20 was used for all tests. In all cases the global optimum was at $(0.0)^n$. The algebraic forms of the functions are given in Table 7.1.

The binary particle swarm converged quickly on $f1$, also known as the sphere function, encoded as a 30-dimensional bitstring. The best

Table 7.1 Functions used by De Jong to test various aspects of optimization algorithms.

Function	Dimension
$f1(x_i) = \sum_{i=1}^{n} x_i^2; -5.12 \le x_i \le 5.12$	30
$f2(x_i) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2; -2.048 \le x_i \le 2.048$	24
$f3(x_i) = \sum_{i=1}^{n} \text{int}(x_i); -5.12 \le x_i \le 5.12$	50
$f4(x_i) = \sum_{i=1}^{n} i x_i^4 + Gauss(0,1); -1.28 \le x_i \le 1.28$	240
$f5(x_i) = 0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}; -65.536 \le x_i \le 65.536;$ $[a_{ij}] = \begin{bmatrix} -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & 16 & 32 & -32 \ldots \text{etc.} \\ -32 & -32 & -32 & -32 & -32 & -16 & -16 & -16 & -16 & -16 & 16 \ldots \text{etc.} \end{bmatrix}$	34

solution the particle swarm found was 0.000002 away from the "perfect" result of 0.0, which it found on 10 of the 20 trials. It is presumed that the difference between the found optimum and the target is due to imprecision in the binary encoding rather than a failure of the algorithm to hit the target.

On the second function, De Jong's $f2$, in 24 dimensions, the particle swarm was able to attain a best value of 0.000068, compared to a target of 0.0; again, the difference is thought to derive from the precision of the encoding rather than the algorithm. This function was encoded in a 24-dimension bitstring. $f2$ was the hardest of the De Jong functions for the particle swarm; the system converged on the best-known optimum 4 times in this set of 20. The hardness of the function might be explained by the existence of very good local optima in regions that are distant in Hamming space from the best-known optimum. For instance, the local optimum

010111111101111000000111

returns a value of 0.000312, while the bitstring

110111101001110111011111

returns 0.000557, and

111000011001011001000001

returns 0.005439. The best-known optimum, returning 0.000068, was found at

110111101110110111101001.

Thus bitstrings that are very different from one another, in terms of Hamming distance, are all relatively good problem solutions. A search algorithm that relies on hill climbing is unlikely to make the leap from a locally optimal region to the global optimum. The function itself has only one optimum and is hard because of the wide flat regions where movement, whether it is toward or away from the optimum, likely results in no real change in fitness.

The third function, $f3$, is an integer function encoded in 50 dimensions whose target value was attained easily on every trial. De Jong's $f4$ function introduces Gaussian noise to the function, and performance

was measured as an average over the entire population rather than a population best. Finally, on $f5$ the algorithm was able to attain a best value of 0.943665 on 20 out of 20 attempts, in 34 dimensions; we presume that to be the global optimum. The system converged rapidly on this fitness peak every time.

The five functions were implemented in a single program, where the only code changed from one test to another was the evaluation function. All other aspects of the program, including parameter values, ran identically on the various functions. Thus it appeared from this preliminary research that the binary particle swarm was flexible and robust.

No Free Lunch

There is some controversy in the field regarding the evaluation of an algorithm, and maybe our claims that particle swarm optimization is "powerful" or "effective" should be disregarded. Imagine two optimization algorithms, one that searches by following the gradient, that is, a hill-climbing algorithm, and another that searches by hopping randomly around the landscape. Now imagine two problems, one, like the sphere function, where the gradient leads inevitably to the optimum and another that has many optimal regions, some better and some worse, a landscape peppered with hills and mountain ranges.

Of course the descriptions have been contrived so that it will be obvious that each algorithm will perform better on one of the problems. It would be foolish to search this way and that when there is a clear-cut yellow brick road leading directly to Oz, and it is equally foolish to climb the nearest hill in a rugged landscape. In this particular case it is clear that the performance of the algorithm depends on the kind of problem.

In important and controversial papers in 1996 and 1997, David Wolpert and William Macready formalized and generalized this observation, and their analysis has some surprising implications. Not only are some algorithms relatively more or less appropriate for certain kinds of problems—but averaged over all possible problems or cost functions, the performance of all search algorithms is *exactly the same*. This includes such things as random search; no algorithm is better, on average, than blind guessing. Provocatively, Wolpert and Macready question whether natural selection is an effective biological search strategy and suggest that breed-the-worst might work as well as breed-the-best, except that no one has ever conducted the experiment on the massively parallel scale of natural evolution.

The No Free Lunch (NFL) theorem, as Wolpert and Macready (1997) called it, has generated considerable discussion among researchers. Where previously there had been hope that some search strategy could be found—and evolutionary computation researchers thought they had it—that would be a best first-guess approach to any class of problems, research has more recently focused on finding exactly what the strengths and limitations of various search strategies are.

Some observers take NFL to mean that no optimization algorithm can be any better than any other. Of course—that's exactly what the theorem says, isn't it? Actually, the theorem says that no algorithm can be better than any other *averaged over all cost functions*. This is a hugely important condition.

What does it mean to average over all possible cost functions? Think of it this way. We have an optimization problem: exiting a room in the dark. Our special algorithm follows these steps (see Figure 7.3(a)):

- Move in a straight line until you reach a wall.
- Move along the wall until you feel an opening.
- Go through the opening.

There could easily be other algorithms, such as stumble-around-waving-your-arms-in-the-air, ever-widening circles, and so on, and some may help us exit better, some worse, than our own proprietary algorithm.

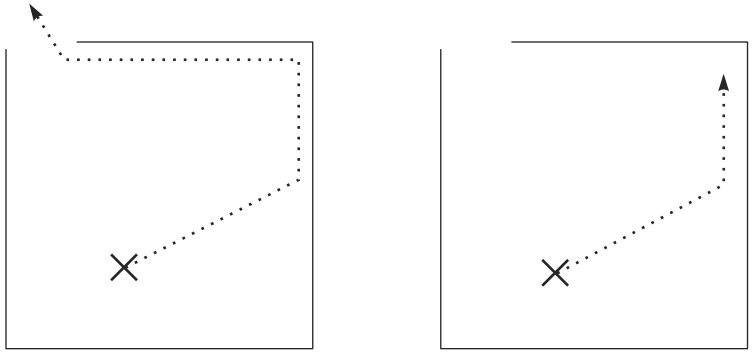


Figure 7.3 Two different algorithms, equally good at different things: the stop-in-corners algorithm is good for finding the way out of a room (a), and the find-corners algorithm is good at getting stuck in the corner of a room (b).

Now imagine that a nefarious NFL advocate has an algorithm called stop-in-corners, which he claims is just as good as our feel-for-opening algorithm. His algorithm goes like this (see Figure 7.3(b)):

- Move in a straight line until you reach a wall.
- Move along the wall until you feel a corner.
- Stop there.

But (you argue) you don't see how he will ever be able to exit a room in the dark, behaving like that! His response is, of course I won't leave the room very well, but there are four corners to the room and only one door, so I will on average be four times as successful at getting stuck in a corner as you will be at exiting. But (you insist) you don't want to get stuck in a corner, you want to exit the room—and your method is better than his for that. He admits that, but adds, my algorithm is as good as yours—averaged over all problems.

The NFL theorem says that, in order to evaluate an algorithm, you have to average it over all cost functions—and there can be very many of those. Some might be

- Exit the room (which is your problem)
- Get stuck in a corner (his problem)
- Find the center
- Find a point halfway between the center and the edge
- Find a point a third of the way between the center and the edge (et cetera, ad infinitum)
- Avoid walls altogether

and so on (another condition is that the search space must be finite, so we don't need to worry about problems from outside our domain). And averaged over all of those, his algorithm and yours are equally good. You might argue that you would never want to find the center or avoid walls altogether, and that gets to the limitation of the No Free Lunch theorem.

While it may be true that no algorithm is better than any other, when averaged over every absurd task that can possibly be imagined, it is perfectly possible that an algorithm would be better than others on the kinds of tasks that we call "problems." If anything, the NFL theorem makes us think about what it is that we try to address with an

optimization algorithm. Most things, even in a finite universe, do not qualify as problems; this may be more a reflection of our way of thinking than anything inherent in mathematics or in the world. Suffice it to say, we do not feel uncomfortable saying that one algorithm, which can reliably find global optima in real problem spaces, is better than another, which can find answers to questions that no one would ever ask.

Multimodality

As part of Bill Spears' investigations of the strengths and weaknesses of genetic algorithms at the Naval Research Laboratory (NRL) and as a former graduate student of Ken De Jong's, he has assembled and posted online a collection of interesting test functions, problems that push and pull and stretch an optimization algorithm to its limit to see what it can and can't do. If there is No Free Lunch, there might be at least Some Kind of Lunch, and researchers want to know what their algorithm is good at.

In collaboration with his NRL colleague Mitch Potter, Spears designed and programmed a "multimodal random problem generator" (De Jong, Potter, and Spears, 1997). The rationale was this: obviously, if a researcher precision-tunes an optimization algorithm to work on one problem, there is a danger that it will fail on everything else. There was a need for a way to come up with different problems, but with some controllable characteristics. The random problem generator offers a way to test an algorithm on novel problems, controlling some aspects of the problems that are expected to affect performance.

Multimodality, in this context, means that a problem has more than one solution or global optimum, conceived as peaks on the fitness landscape. For instance, the problem $x^2 = 25$ is multimodal; it has two optimal solutions: $x = +5$ and $x = -5$. Since a genetic algorithm is often implemented using binary encoding, Spears wrote the program to create multimodal binary problems for the GA to solve. The concept is very straightforward. The researcher defines the dimensionality of the problem, that is, the length of the bitstring, and how many modes or peaks are desired, and the program creates that number of bitstrings, made of random sequences of zeroes and ones. For instance, imagine a researcher has specified that dimensionality $N = 10$ and multimodality or number of peaks $P = 5$. The problem generator might produce these bitstrings:

0100110111

1110010010

1101101010

0100000000

1110100101

With 10-dimensional bitstrings there are $2^{10} = 1{,}024$ possible patterns of bits. The goal for the optimizing algorithm is to find any one of the five peaks that have been defined by the program. The Hamming distance between a bitstring and the nearest optimum provides a fitness evaluation; that is, the more similar the bitstring is to one of the specified peaks, the fitter it is. For 10-dimensional bitstrings, the farthest an individual can be from a peak is 10 Hamming units, and of course a perfect match is a distance of zero from one of the peaks.

Multimodal problems can be hard for genetic algorithms. Recall that in GAs, chromosomes cross over in every generation; sections of successful ones are joined together to produce the next generation's population. In a multimodal situation it is entirely possible that the parts that are joined together come from chromosomes whose fitness derives from their proximity to different optima. For instance, the chromosome 0100110110 is only one bit different from the first optimum defined above, and 0110010010 is only one bit different from the second solution. Putting them together (we'll cut it right in the middle to be fair) could produce the child chromosome 0100110010, which is three bits different from the first optimum (Hamming distance = 3) and three bits different from the second—moving away from both of them. It is exactly the multimodality of the problem that makes crossover ineffective in this case.

GAs rely not only on recombination but on mutation (and sometimes other operators) for moving through a problem space. De Jong, Potter, and Spears tried several modifications of GAs, including one whose only operator was mutation—no crossover—in the multimodal random problem generator, calling it GA-M. In this algorithm, each site on each bitstring has a low probability of changing from a zero to a one or vice versa, usually less than 0.01. At each generation the population is evaluated, the fittest ones are selected, and mutation is applied to them.

Through this process, generations tend to improve; this amounts to a kind of stochastic hill climbing, as the more fit members of the population are more likely to be retained and mutated.

De Jong, Potter, and Spears also tested a GA implemented with crossover only—no mutation—and found that this kind tended to flounder in the early generations, but once the population started to converge on one particular peak or another, improvement came relatively fast. These crossover-only GAs, called GA-C, were very successful at finding one of the optima, if you waited long enough. The same was true of traditional GAs with both crossover and mutation. Mutation-only GAs, on the other hand, constantly improved, generation by generation, but if the dimensionality of the problem was high, the chance of mutating in a direction that led to improvement was very small and grew smaller as the population approached the optimum. When bitstrings were short, mutating chromosomes found optima quickly and efficiently, but "the curse of dimensionality" made bigger problems too difficult for them. Though they might have eventually found the global optimum, improvement

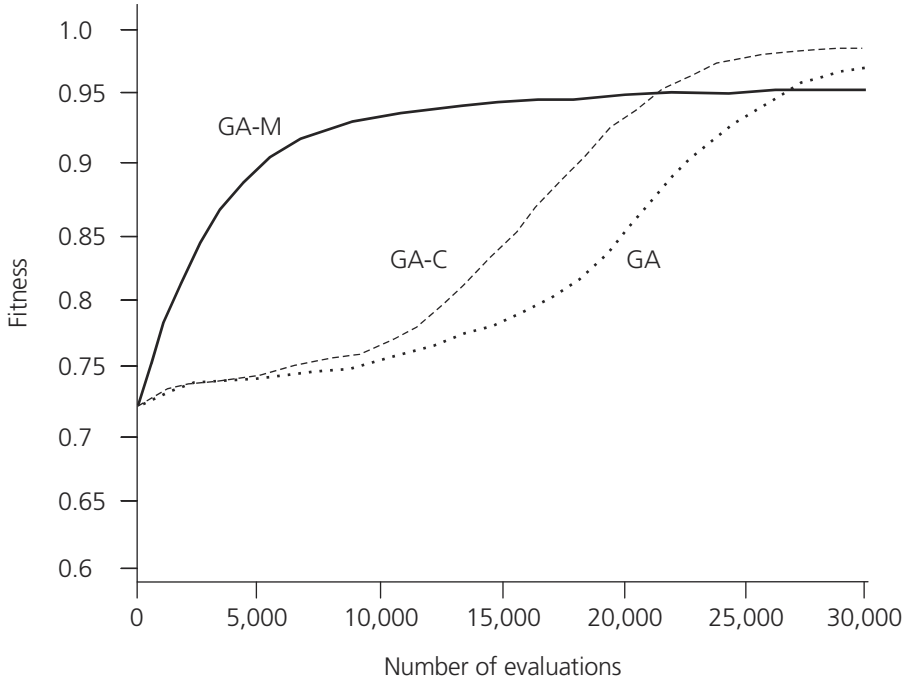


Figure 7.4 Average best-so-far performance of three types of genetic algorithms on 500-peak problems. (From De Jong, Potter, and Spears, 1997.)

decelerated over time. Figure 7.4 shows the performance of the three types of genetic algorithms.

In a follow-up to the De Jong et al. study, Kennedy and Spears (1998) compared the binary particle swarm algorithm with the three variations of GAs in the multimodal random problem generator. The study was constructed in the form of an experiment, where three independent variables were manipulated—algorithm, dimensionality, and multimodality. There were four kinds of algorithms (GA-C, GA-M, GA, and PS), two levels of dimensionality (20 and 100), and two levels of multimodality (20 and 100). In each of the 16 conditions of the experiment, there were 20 observations, and the population size was 100. (This population size, which is much bigger than a typical particle swarm, was used to make the two paradigms commensurate.)

The dependent variable in Kennedy and Spears' experiment was the shape of the best-so-far performance curves over time. This is a multivariate measure, more complicated than those found in the typical experiment, but easily computable with good statistical software. Each condition in the experiment was run 20 times for 20,000 evaluations. The mean best performance was calculated after 20 evaluations, and after 1,000, 2,000, and so on up to 20,000. It was possible to statistically compare the shapes of the performance curves for all comparisons, the question being not how well the various algorithms perform in the long run over the dimensionality and multimodality conditions, but how changes in their performance differed over time (see Figure 7.5).

GA-M performed best of all the algorithms in the early iterations of every condition, but was quickly overtaken by all the others, except in the "lite" condition, with short, 20-bit bitstrings and only 20 peaks. When either dimension or modality or both increased, however, GA-M suffered in its ability to find one of the peaks. The two GA variations with crossover, that is GA-C and GA—which implemented both crossover and mutation—started in every condition with a "dip" in performance, and then rose toward an optimum, almost always finding one of the peaks by the 20,000th evaluation.

The binary particle swarm performed the best in all conditions except the "lite" one (where it was second best); it found a global optimum on every trial in every condition and did it faster than the comparison algorithms. This is not to say that it would have performed better than *any* GA on these problems, and it may be possible to tune the parameters of a GA to optimize its performance in a particular situation. On the other hand, the significance of these results—which were statistically significant in a multivariate analysis of variance (MANOVA)—should not

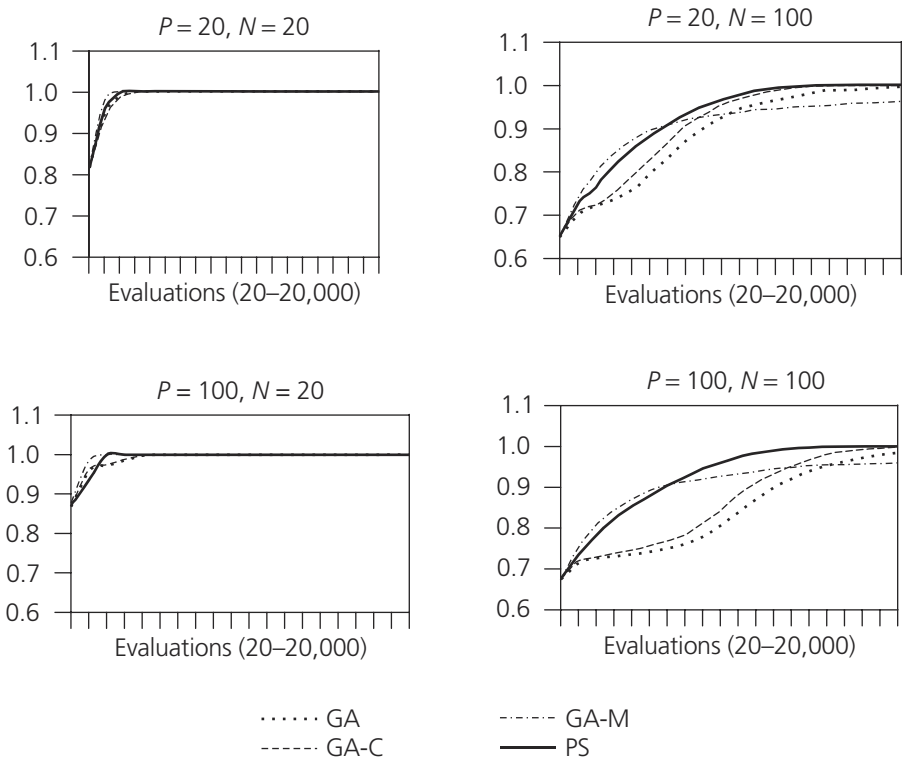


Figure 7.5 Best-so-far performance curves for four kinds of algorithms, with two levels of dimensionality (*N*) and two levels of multimodality (*P*). (From Kennedy and Spears, 1998.)

be underestimated. The binary particle swarm performed very well in the multimodal random problem generator, compared to some tough competitors.

As mentioned above, the traditional genetic algorithm with crossover often has trouble with multimodality: Crossover between parent chromosomes that are near different optima can result in a child chromosome that is not near any optimum at all. The GA with mutation simply didn't have the power to search a large space for one of the target bitstrings. For the binary particle swarm, though, multimodality just means more places to roost, more targets to hit; the effect is just the opposite of the traditional GA with crossover. Subsections of a binary particle swarm population can converge on different optima, and particles that are halfway between two optima have twice as many good directions to turn—not twice as many opportunities to fail.

Minds as Parallel Constraint Satisfaction Networks in Cultures

John J. Hopfield is a well-respected physicist. You might say that he is a "silicon-based" researcher. In 1982, Hopfield published a paper that, according to many neural network researchers, played a more important role than any other single paper in reviving the field of neural networks. One of his developments was the definition of the energy of a neural network: For a given state of the network, the energy is proportional to the overall sum of the products of each pair of node activation values (*Vi, Vj*) and the connection weight associated with them (T_{ij}), that is,

$$E = -0.5 \sum_{i,j;i \neq j} T_{ij} V_i V_j \quad (T_{ii} \equiv 0)$$

He showed that his algorithm for adapting the node activations, given a set of weights connecting the nodes, makes *E* decrease and that eventually a minimum *E* is obtained.

In the parallel constraint satisfaction paradigm as typically published in psychological journals, nodes in a Hopfield network represent elements, often propositions or concepts. When some subset of the nodes' states is clamped on, the network seeks a pattern of states of the other nodes that maximizes harmony or coherence or minimizes energy (*E* in the above equation).

Both binary- and continuous-valued Hopfield networks have been studied, but here we focus on the binary version. Nodes exist in either the active or the inactive state, represented as one and zero, and are linked by bidirectional, symmetrical connections T_{ij}. The usefulness of the network is seen when only part of an input pattern, or a noisy pattern, is introduced; the network outputs the complete pattern that best fits with the clamped pattern.

In the original Hopfield paradigm the weights between nodes are derived from data, but in most "cognitive" versions the weights are derived from the assumptions about the connections among attitudes, beliefs, or behaviors. There is usually some rationale for the choice of weight values, but these vary from one implementation to another. The commonsense fact is that some beliefs are related and others are not, they can be related positively or negatively, and some pairs of beliefs are more strongly related than others. Theorists instantiate this fact into their models with varying degrees of success and credibility.

A binary particle swarm was set up to optimize the simple network structure described by Edwin Hutchins (1995) and described in Chapter 5. A network is composed of six nodes, where the three left nodes are positively connected with one another, the three right nodes are positively connected, and there are negative connections between the right- and left-side nodes. The network is in its optimal state either when the left nodes are active and the right nodes are inactive or when the right nodes are active and the left ones are inactive; in binary terms this means that 000111 and 111000 are globally optimal patterns of node activation. These two states correspond to two interpretations of a pattern of stimuli; Hutchins introduces the model with a story of a shipwreck, where some lights in the distance were interpreted differently by various people, resulting in a collision.

As we described earlier, Hutchins had programmed a group of four networks with varying numbers of connections between them to demonstrate the effects of communication within a group. Somewhat similarly, in the particle swarm each individual is represented as a network, and their interactions allow them to find the optimal activation patterns. When the paradigm was run with 20 individuals, the globally optimal patterns were found every time by all members of the population. We admit it is not an especially difficult optimization problem, as there are only 2^6, or 64, possible network states, with two equally good global optima. The binary particle swarm has been implemented on much larger and more complicated constraint satisfaction networks, with excellent results.

In this example it is important to note the formation of cultures in the population, as seen in Figure 7.6. In the lbest particle swarm, individuals interact with their adjacent neighbors; as a result of this, neighbors become more similar to one another over time, and patterns spread from neighbor to neighbor. As commonly happens, different sections of the population settle on different optima. This seems to be a very good model of the process by which polarization forms in a society. Given the facts as they are known, both conclusions are correct: both result in patterns where the nodes with positive inputs are activated and nodes with negative inputs are inactive. Through discussing the issues among themselves, individuals not only come to agree with their neighbors but also arrive at conclusions that fit together optimally. Social interaction results in cultural convergence on patterns of beliefs, and culture results in relatively good cognitive performance.

We have described a version of particle swarm optimization that is useful when binary variables (1 or 0, true or false, etc.) are used. It is a powerful paradigm that is simple to use, easy to understand, and

000111
000111
000111
000111
000111
111000
111000
111000
111000
000111

Figure 7.6 Example of a population of binary particle swarm individuals' solutions to Hutchins' six-node parallel constraint satisfaction problem.

converges rapidly, even for high-dimensional, multimodal problems. In the real world, however, we usually must deal with continuous numbers. In the next section we show how particle swarm optimization works in continuous numbers, maintaining its simplicity, power, and ease of use.

The Particle Swarm in Continuous Numbers

The progression of ideas has been from a purely qualitative social optimization algorithm—the Adaptive Culture Model—to a model that can be interpreted as qualitative or quantitative—the binary particle swarm. In this chapter we arrive at the "real" particle swarm, which is a truly numeric optimization algorithm (Kennedy and Eberhart, 1995; Eberhart and Kennedy, 1995). The particle swarm algorithm searches for optima in the infinite search space that is often symbolized as R^n—the n-dimensional space of real numbers. (Actually, of course, it searches in computable space. And we are using PCs, so caveats related to things such as round-off errors are valid.)

The Particle Swarm in Real-Number Space

In real-number space, the parameters of a function can be conceptualized as a point. If we think of a psychological system as a kind of information-

processing function, then any measure such as the psychotherapist's MMPI, a public opinion survey questionnaire, a risk-seeking inventory, a management consultant's Myers-Briggs, or the "What's Your Love-Q?" in the back of *Cosmo* will produce real numbers that can be interpreted as a point in a psychological space. In engineering applications it is customary to think of system states as points in multidimensional space. The multidimensional space is referred to by various names, depending on the situation. Names include state space, phase space, and hyperspace.

It is a small philosophical leap to suggest that multiple individuals can be plotted within a single set of coordinates, where the measures on a number of individuals will produce a population of points (see Figure 7.7). Being near one another in the space means that individuals are similar in the relevant measures; if the test is valid, there may be real similarities between the individuals. If various vectors of parameters to a mathematical function are being tested, then we would expect points in the same region to have correlated function outputs and correlated fitness.

In this view of individual minds as points in a space, change over time is represented as movement of the points, now truly *particles*. Forgetting and learning might be seen as cognitive decrease and increase on some dimensions, attitude changes are seen as movements between the negative and positive ends of an axis, and emotion and mood changes of numerous individuals can be plotted conceptually in a coordinate system. As multiple individuals exist within the same high-dimensional framework, the coordinate system contains a number of moving particles. One insight from social psychology is that these points will tend to move toward one another, to influence one another, as individuals seek agreement with their neighbors.

Another insight is that the space in which the particles move is heterogeneous with respect to evaluation: some regions are better than others. This is of course true for functions and systems as well as psychology; some points in the parameter space result in greater fitness than others. A vector of cognitive, mathematical or engineering parameters can be evaluated, and it is presumed that there exists some kind of preference or attraction for better regions of the space.

The position of a particle i is assigned the algebraic vector symbol $\vec{x}_i$. Naturally there can be any number of particles, and each vector can be of any dimension. Change of position of a particle could be called $\Delta\vec{x}_i$, but in order to simplify the notation we call it $\vec{v}_i$, for velocity. Velocity is a vector of numbers that are added to the position coordinates in order to move the particle from one time step to another:

$$\vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t)$$

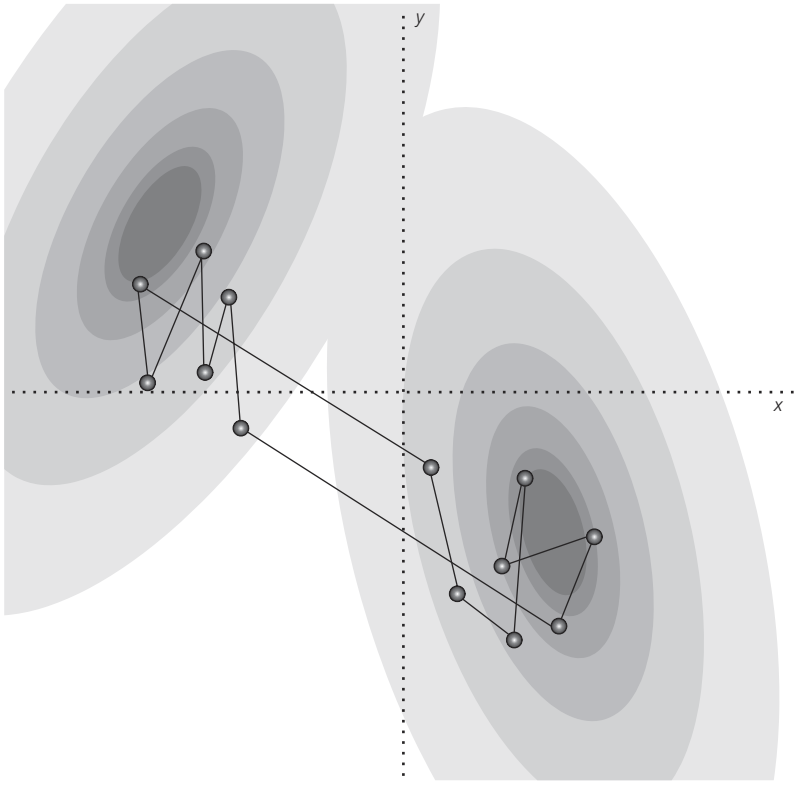


Figure 7.7 Particles in a real-number space are connected to topological neighbors, and neighbors tend to cluster in the same regions of the search space.

The question then is to define rules that move the particle in the desired way, in a way that optimally allocates trials while searching for optima. The particle swarm algorithm samples the search space by modifying the velocity term.

Social-psychological theory suggests that individuals moving through a sociocognitive space should be influenced by their own previous behavior and by the successes of their neighbors. These neighbors are not necessarily individuals who are near them in the parameter space, but rather ones that are near them in a topological space that defines the sociometric structure of the population. Who influences you is defined by your connections in the social network, not by positions in a belief space—there may be someone who agrees closely with you about everything, whom you have never met (and whom no one you know has ever met—and no one *they* know . . .), and who consequently does not influence you. Thus in the continuous-number particle swarm, as in the

binary version, a neighborhood is defined for each individual, based on the position in the topological population array. The population array is commonly implemented as a ring structure, with the last member a "next-door" neighbor to the first member.

As the system is dynamic, each individual is presumed to be moving—this should really be called *changing*—at all times: this is the Lewinian concept of locomotion. The direction of movement is a function of the current position and velocity, the location of the individual's previous best success, and the best position found by any member of the neighborhood:

$$\vec{x}_i(t) = f(\vec{x}_i(t-1), \vec{v}_i(t-1), \vec{p}_i, \vec{p}_g)$$

Further, just as in the binary version, change (which is now defined in terms of velocity instead of probability) is a function of the difference between the individual's previous best and current positions and the difference between the neighborhood's best and the individual's current position. In fact the formula for changing the velocity is identical to the one used to adjust probabilities in the binary version, except that now variables are continuous, and what is adjusted is the particle's velocity and position in R^n:

$$\begin{cases} \vec{v}_i(t) = \vec{v}_i(t-1) + \varphi_1(\vec{p}_i - \vec{x}_i(t-1)) + \varphi_2(\vec{p}_g - \vec{x}_i(t-1)) \\ \vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t) \end{cases}$$

where, as before, the φ variables are random numbers defined by an upper limit. The effect of this is that the particle cycles unevenly around a point defined as a weighted average of the two "bests":

$$\frac{\varphi_i \vec{p}_i + \varphi_2 \vec{p}_g}{\varphi_1 + \varphi_2}$$

Because of the random numbers, the exact location of this point changes on every iteration.

For reasons that will be discussed in the next chapter, the system as given thus far has a tendency to explode as oscillations become wider and wider, unless some method is applied for damping the velocity. The usual method for preventing explosion is simply to define a parameter $V_{\max}$ and prevent the velocity from exceeding it on each dimension d for individual i:

$$\text{if } v_{id} > V_{\max} \text{ then } v_{id} = V_{\max}$$
$$\text{else if } v_{id} < -V_{\max} \text{ then } v_{id} = -V_{\max}$$

The effect of this is to allow particles to oscillate within bounds, although with no tendency for convergence or collapse of the swarm toward a point. Even without converging, the swarm's oscillations do find improved points in the optimal region. One approach to controlling the search is the implementation of an "inertia weight." Another method, developed by French mathematician Maurice Clerc, involves a system of "constriction coefficients" applied to various terms of the formula. These approaches are discussed in the next chapter.

Pseudocode for Particle Swarm Optimization in Continuous Numbers

In sum, the particle swarm algorithm in the space of real numbers is almost identical to the binary one, except that $\vec{v}_i$ defines increments of movement rather than a probability threshold, as illustrated by the following pseudocode:

```
Loop
   For i = 1 to number of individuals
      if G(x_i) > G(p_i) then do          //G() evaluates fitness
         For d = 1 to dimensions
            p_id = x_id                   //p_id is best so far
         Next d
      End do

      g = i                               //arbitrary
      For j = indexes of neighbors
         If G(p_j) > G(p_g) then g = j    //g is index of best performer
                                          in the neighborhood

      Next j
      For d = 1 to number of dimensions
         v_id(t) = v_id(t − 1) + φ1(p_id − x_id(t − 1)) + φ2(p_gd − x_id(t − 1))
         v_id ∈ (−V_max, + V_max)
         x_id(t) = x_id(t − 1) + v_id(t)
      Next d
   Next i
Until criterion
```

Implementation Issues

One issue faced by anyone implementing the algorithm is how to initialize the population. The positions and velocities of the particles are usually initialized randomly. The initial random positions are often distributed over the dynamic range of each dimension. The initial velocities are often distributed randomly over $[-V_{\max}, V_{\max}]$.

Another question faced at implementation time is, "How many particles should I use?" There is no pat answer to this question, but the experience of the authors indicates that choosing somewhere between 10 and 50 usually seems to work well. One of the authors [JK] tends to run with populations at the lower end of this range, while another [RE] often prefers higher numbers of population members. If you are used to working with traditional genetic algorithms, you should probably consider starting with fewer particles than the number of GA chromosomes you would use.

The particles' flight patterns can be interesting to watch. We find that you can sometimes learn something about the problem being modeled or optimized by watching the particles fly. Pictures of the swarm that are snapshots in time, such as the figures of this book must be, cannot do justice to the system's dynamics. We therefore suggest that you watch the flight patterns for yourself. You can do this by compiling and running the source code we provide on the book's web site. But an even easier way is to go to the web site and run the Java applet. You can choose from a number of test functions and set the parameters for each function.

An Example: Particle Swarm Optimization of Neural Net Weights

The first real implementation of the particle swarm algorithm was a model that bridges psychological theory and engineering applications. The feedforward artificial neural network is a statistical model of cognition that inputs vectors of independent variables and outputs estimates of vectors of dependent variables. The network is structured as a set of weights, usually arranged in layers, and the optimization problem is to find values for the weights that make the mapping with minimal error. It is beyond the scope of this book to go into detail about neural networks; you are encouraged to refer to books that contain basic information on

neural networks and their applications such as Eberhart, Simpson, and Dobbins (1996) or Reed and Marks (1999).

Normally, feedforward networks are optimized through some sort of gradient descent algorithm, traditionally using backpropagation of error. The cognitivistic interpretation is that the individual makes judgments about stimuli through adjustments based on personal experience. In contrast to this, the particle swarm view places the individual in a social-psychological context. The individual makes adjustments by integrating individual experience with the discoveries of others. Cognition in this view is a collaborative enterprise, occurring in a transpersonal milieu within which the individual is only a part.

Besides its ability to provide insights to the understanding of cognition, the feedforward neural network is also an extremely useful data-analysis tool, comprising a superset of which regression analysis is a member. These networks are often referred to as "universal function approximators" because they are able to mimic any kind of mathematical function. Networks excel in matters of estimation, for instance, when it is desirable to make a prediction from some data; their shortcoming for statistical use is related to difficulty in explaining a result, once it is obtained. This shortcoming has two levels to it: not only is there a human relations problem in trying to understand or explain how an estimate was produced by passing data through dozens or hundreds of weighted connections, but also it is so far not known how to make statistical inferences to a population from a network, that is, how to estimate the degree of confidence in a result.

A classic test problem for the feedforward network is the "XOR problem." The exclusive-or (abbreviated XOR) logical operation is true if one and only one argument is true, unlike the usual Boolean OR operator, which is true if any argument is true. This little logic puzzle accepts a pair of binary inputs, and outputs a 0 if they are the same, 1 if they are different from one another. The network is trained with a data set of inputs and outputs, shown in Table 7.2.

The feedforward neural network has been widely studied as a kind of model of human cognition, especially categorization, as it is able to simulate the human knack for lumping things together that do not seem to belong together. *Linear separability* exists when some weighted additive combination of properties can be used to classify examples, in other words, when things are graphed by their attributes and a straight line or surface can be drawn that perfectly separates the two kinds of things. Human categorization often—perhaps usually—is not linearly separable.

Table 7.2 The XOR function outputs a zero when the inputs match and a one if they are different from one another.

Inputs		*Output*
1	1	0
1	0	1
0	1	1
0	0	0

The XOR problem exemplifies this, as it is not linearly separable (see Figure 7.8).

Besides the two input and one output nodes required, it has been established that at least two "hidden nodes" are required for this problem, unless the inputs are to be directly connected to the outputs. Thus for purposes of illustration, we will use a network with two hidden nodes (see Figure 7.9). Weighted connections link each input node to each hidden node and each hidden node to the output node. Each hidden and output node also has a bias weight associated with it. Thus this network requires optimization of nine weights. The particles, in other words, move in nine-dimensional space, looking for patterns of weight values that effectively output the correct value for a given pair of inputs.

A population of 20 individuals is defined, where each individual starts the trial as a random vector of weights; neighborhoods are parsimoniously defined as each individual's adjacent array neighbors (borrowing from cellular automaton methodology). Inputs are entered into the network, and the mean squared difference between the obtained and observed values is calculated. The objective of course is to minimize this error measure. Error is computed for each member of the population; each particle compares its current position to its previous best and its neighbors' previous best and adjusts its velocity toward the points where those values were found. Eventually some member of the population will have found a pattern of weights that meets the criterion, for example, average sum-squared error < 0.02. One early study (Kennedy, 1997) found the network weights could be optimized to the criterion, with $V_{\max} = 2$ and $\varphi_{\text{total}} = 4$, in a median of 70 iterations, using a population of 20 particles. We note that some recent adaptations of the particle swarm algorithm permit faster optimization of neural net weights.

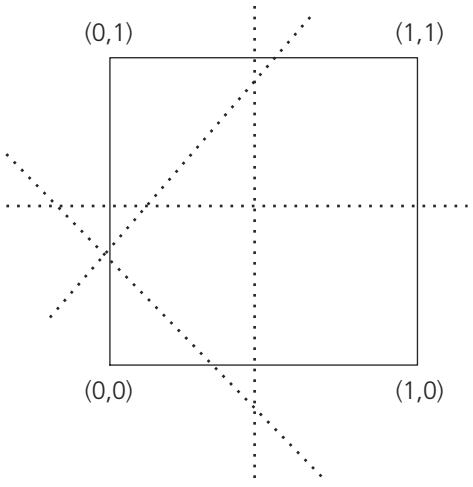


Figure 7.8 The XOR problem is not linearly separable. No line can be drawn such that the corners (0,0) and (1,1) are on one side of it and the corners (0,1) and (1,0) are on the other side.

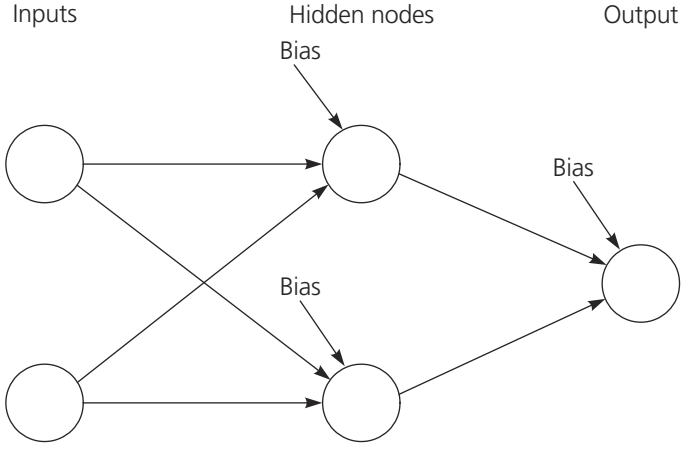


Figure 7.9 The XOR feedforward network takes a pair of inputs. It outputs 1 if the two inputs are different and 0 if they are the same.

A Real-World Application

The XOR network example has been used to ease you into the subject of neural networks. The first real-world application of particle swarm optimization to a neural network was one designed to provide an estimate of the state of charge of a pack of batteries in an electric vehicle. The network originally had eight inputs, five hidden nodes, and one output. Eventually, the network was simplified to five inputs, three hidden nodes, and one output. A team that included one of the authors [RE] was training feedforward neural networks using the backpropagation algorithm. Although the networks were fairly small, there were approximately 2,500 data sets being used to train them, and each training session using backpropagation required approximately 3.5 hours.

It was a simple matter to delete the backpropagation training algorithm and substitute particle swarm optimization. The results were dramatic. Training time required to achieve the same sum-squared error value dropped from about 3.5 *hours* to about 2.2 *minutes*. This was the first time we got a glimpse of how powerful the particle swarm algorithm is when applied to real-world optimization problems. For a more complete description of this application, see Eberhart, Simpson, and Dobbins (1996). Although the training of the network did not *depend* on particle swarm optimization to be successful, it became obvious that it was a subject worthy of further investigation.

Subsequently, many neural networks have been trained using particle swarm optimization. In these and other applications, some "rules of thumb" have evolved. The first regards the selection of V_{max}. The standard sigmoid function used in feedforward networks essentially saturates with inputs of about +/− 10. Therefore, we often consider the dynamic range of weights to be something like 20. It seems reasonable that we limit the maximum velocity to some percentage of the dynamic range. A V_{max} of 2.0 has thus been found to be successful for training feedforward networks with sigmoid activation functions. The maximum acceleration constant φ_{total} has generally been set to 4.0 for these applications. (Other ways to limit velocity, such as the damping weight and constriction factor approaches, are described in Chapter 8.)

The second rule of thumb regards the setting of the neighborhood size. Although no theoretical foundation exists, it has been found that a neighborhood size of about 10–20 percent of the population often works well. Thus, when the population comprises 40 particles, we often choose a neighborhood size of two or three adjacent neighbors on both sides of each particle, for a neighborhood size of 5 or 7, including *i*.

As will be seen in Chapter 9 on applications of particle swarms to engineering problems, the particle swarm paradigm has been used on very large networks. Further, in an exciting recent development, the particle swarm can actually be used to *design* the neural net; nodes can be simplified and deleted from the network at the same time that optimal values for weights are being found.

The training (weight adjustment) of multilayer artificial neural networks with nonlinear transfer functions was previously thought to be intractable. Though the techniques such as backpropagation of error developed for optimizing weights are known to be imperfect—they are susceptible to local optima—the methods have been considered good enough for most uses. In fact, some tricks (we are thinking here of Scott Falman's "quickprop" algorithm) have been found that propitiate the discovery of robust solutions.

As has been seen, there exists a largish library of functions for the testing of the strengths and weaknesses of optimization algorithms, including the De Jong test suite and other functions that have been collected over the years. Any new optimization algorithm, including the particle swarm, has to prove itself by triumphing over these diabolical mathematical minefields, which are contrived with multiple optima, corrugation, serration, cavities and bumps, high dimensionality, infeasible regions, epistasis, dynamic distortions, and deception (this latter is a term for problems where the gradient would move a hill climber away from the global optimum). In our studies and those of others, the particle swarm algorithms have been successful at solving these problems.

The Hybrid Particle Swarm

Now that we've shown you how binary and real-valued particle swarms work, we'd like to briefly mention one of the current subjects of our research: hybrid particle swarms. As you might guess, a hybrid swarm combines binary and real-valued parameters in one search. In this section, we give you a few ideas about *why* you might want to implement such a system and *how* to do it.

Why would we want a hybrid algorithm? Perhaps we have a problem with both continuous and binary variables; one example is an explanation facility for a multiple-symptom diagnostic system. Let's say you have designed a neural-network-based diagnostic system to diagnose

various abdominal diseases such as appendicitis and nonspecific abdominal pain. Some of the symptoms such as body temperature and white blood cell count are continuous variables and are best represented as real numbers. Other inputs such as the answer to the question "Have you had your appendix removed?" are binary values. So the mix of symptoms includes real and binary numbers. This poses no trouble when training a neural network, whether you use a more traditional approach such as backpropagation or our favorite approach using particle swarm optimization.

It can be very difficult to explain how a network has made the decision it has. As the information is encoded in complex patterns of connection weights, usually linking several layers of nodes, it is rarely clear what it is about a pattern of inputs that led to the diagnosis. It is possible to construct an *explanation facility* that provides the physician with a tool to understand how the network diagnoses. We typically use an evolutionary algorithm to determine two kinds of things. First, where are the hypersurfaces that represent differential diagnoses? A differential diagnosis is that set of inputs (or a suite of such inputs) that lies on the borderline between two diagnoses. In other words, the differential diagnoses are those sets of inputs for which the chances are equal that the symptoms represent appendicitis and nonspecific abdominal pain, for example. Second, what are quintessential examples of each diagnosis? In other words, what combinations of inputs result in the maximum likelihood of a disease such as appendicitis, according to the system?

Finding these sets of inputs is an example of an "inverse problem" and is something that evolutionary algorithms are good at. So now the question is *how* we do it. You've probably already figured out the answer. We simply operate on (medical pun intended) binary inputs with the binary particle swarm algorithm and treat (oh no, another medical pun) the continuous variables with the real-valued particle swarm.

We've done some preliminary tests, and the approach seems to work well. We emphasize, however, that it is a subject of ongoing research and development.

Science as Collaborative Search

The kind of decision processes instantiated in both the binary and real-valued particle swarm algorithms exemplify a tendency that has been widely regarded as a flaw or error when seen in human cognition. Karl

Popper (1959) revolutionized scientific methodology by persuading us that it is impossible to confirm a hypothesis—it is only possible to disprove one. In his famous example, even if you have seen a million white swans, and never in your life have you seen any other color, you still have not conclusively proven that "all swans are white." On the other hand, a single black swan will *dis*prove the statement. Modern scientific methodology is based on the philosophy of *null hypothesis testing,* which takes the tack of trying to prove the hypothesis that your research hypothesis is in fact false, that is, you look for black swans. A hypothesis cannot be tested unless it is falsifiable, and scientific proof relies on identifying what would happen if the hypothesis were indeed false and then discovering if those events occur in an experimental situation.

While it is logically impossible to prove a hypothesis by accumulating support for it, this is exactly the approach people normally take. Cognitive psychologists call this tendency *confirmation bias,* the propensity to irrationally seek confirmation for our beliefs, rather than falsification. Klayman and Ha (1987) turned the issue around by pointing out that falsification is not a good strategy for determining the truth or falsehood of many hypotheses. They proposed that people tend to use a "positive test strategy," which is defined as testing cases that are expected to produce the hypothesized result, rather than testing cases that are intended to fail to produce it. They suggested that people use the positive test strategy (+testing) as a default heuristic. Further, they noted that "as an all-purpose heuristic, +testing often serves the hypothesis tester well" (p. 225).

Another way of looking at this is to compare *truth* and *certainty.* Most of the time, people solving a problem don't require knowledge that something be established as *true;* they only require that it be established to a level of *certainty.* As Karl Popper said in a recent interview with writer John Horgan (1996), "We must distinguish between truth, which is objective and absolute, and certainty, which is subjective." Adjusting your hypotheses toward the consensus position and testing cases that confirm what you already believe are methods for increasing the sense of certainty. One thing that will undermine that sense is of course contradiction by empirical facts; thus, "+testing" can only work if it is consistent with phenomena in the world. While it is possible to build up certainty in the absence of truth, the two are not independent—a fact that can be capitalized on. Strategies that increase certainty may be likely to discover truths as well.

In the model we have just described, individuals move *toward* their previous successes; confirmation bias is fundamental to this strategy. But

this is an elaborated, social confirmation bias: individuals seek to confirm not only their own hypotheses but also those of their neighbors. Paradoxically, though we may be pointing out that people are not very scientific in their thinking, especially insofar as science is supposed to be mathematical and deductive, even scientists act like this. What Thomas Kuhn (1970) calls a *paradigm* is a kind of confirmatory social convergence of scientists in a theoretical decision space: "A paradigm is what the members of a scientific community share, and conversely, a scientific community consists of men who share a paradigm" (p. 176). The scientists come to agreement on the use of terminology, acceptable research methods, and other aspects of their work, and it is by intense communal focus on a narrowly defined subject domain that the scientists are able to fully exploit the learning that has preceded them. In the particle swarm analogy, a Kuhnian "revolution" occurs when an individual finds a better region of the search space and begins to attract its neighbors toward it by becoming the best in the neighborhood.

In the 1960s and 1970s some evolutionary theorists began to propose a correspondence between scientific and evolutionary processes that continues to be reiterated (Campbell, 1965, 1974; Popper, 1972; Lorenz, 1973; Atmar, 1976; Dawkins, 1987). In this view, an individual member of a species represents a hypothesis about the logical properties of the environment; the validity of the hypothesis is shown by the survival of the individual. This inductive approach to learning leads to constantly improving prediction of the important aspects of the environment. As in previous discussion of the memetic view, our objection to the too-literal acceptance of this view has to do with the difference between selection, as it occurs in evolution, and change as it appears in learning. A scientist often has a long career spanning the comings and goings of multiple paradigms. Hypotheses are ideas that are held in the minds of scientists, who are able through constant refinement, through constant adaptation—through learning—to improve the validity of their hypotheses. The evolutionary perspective looks at the mutation and selection of ideas per se, while the particle swarm view looks at the adaptive changes of individuals who hold those ideas.

In informal human social search of a problem space, little effort is typically made to carefully choose data, and both measurement and sampling error are extremely plentiful—a glance through any textbook in social or cognitive psychology will reveal dozens of "heuristics," "biases," and "errors" in human information processing. We propose that many of the biases result from the "particle swarm" tendency of individuals to move toward self- and social confirmation of hypotheses—

a tendency that, while logically invalid, in fact results in excellent information-processing capabilities. We don't agree that human thinking is faulty; we suggest on the contrary that formal logic is insufficient to solve the kinds of problems that humans typically deal with.

Emergent Culture, Immergent Intelligence

After some number of iterations the members of the particle swarm populations are found to have congregated around one or more of the optima. In cases where multiple global optima are discovered by the population, topological neighbors tend to cluster in the same regions of the search space. These clusters extend beyond hard-coded neighborhoods. When an individual finds a relatively optimal combination of elements, it draws its adjacent neighbors toward itself; if the region is superior, the neighbors' evaluations will improve as well, and they will attract *their* neighbors, and so on. If another subset of the population is attracted to a different but equally good region of the problem space, then a natural separation of groups is seen to emerge, each with its own pattern of coordinates that may easily be thought of as norms or cultures. ("We used to say 'customs' when we were talking about norms; now the norm, of course, is to say 'norm' " (Picker, 1997, p. 1233).)

When one solution is better than another, it usually ends up absorbing the lesser pattern, though in some cases mediocre "compromise" individuals on the borders of groups prevent the spreading of better solutions through the population. The polarization of these artificial populations into separate cultures appears very similar to the convergence of human subpopulations on diverse norms of attitude, behavior, and cognition. Interaction results in conformity or convergence on patterns that are similar for proximal individuals and may be different between groups.

The formation of cultures in particle swarm trials is not specified in the computer programs and is not readily predictable from the definitions of interactions in the programs. It thus would be considered an emergent effect, though we grant that "emergence" might actually be a term that represents the simplicity of our own minds; a property not of the system but of our failure to understand it.

There is another important feature of the behavior of a particle swarm, or of a human society, and that is the *immergence* of cognitive adaptation of individuals as a result of the top-down effect of *emergent*

culture. Participants in the system become intelligent, acquiring whatever qualities have been defined as "good" in the fitness function, as a result of the cultural optimization enabled by local interactions (see Campbell, 1990).

The cultural convergence of individuals in the search space allows the intensive exploitation of optimal regions. Relatively good combinations of elements, which in human society may be beliefs, behaviors, problem-solving steps, opinions, and so on, receive focused attention. As a result, the performances of individuals are improved. Culture, the emergent result of bottom-up processes, enables top-down immergent mental phenomena, optimizing the cognitive processes of individuals.

According to this perspective, minds and cultures are intimately interwoven products of the interactions of individuals. There is no need to postulate any great distinction between "internal" and "external" information processing—it all works together. This is not to deny any individual's conscious experience, or to say that any two individual minds are identical to one another, and certainly is not to predict the spread of insipid homogeneity through a society. Rather, the diversity of minds, of explorations and explanations and exploitations, provides the raw material for the emergence of culture and simultaneous immergence of intelligent behavior.

Emulation of superior cognitive positions allows individuals to adapt efficiently to complex cognitive landscapes. As an optimizer, the particle swarm algorithm has been shown to perform very well on a wide range of hard test functions. The obvious conclusion is that mutual collaborative emulation can result in individual adaptation: intelligence. A population of social entities evaluating, comparing, and imitating is able to zero in on good solutions to complex problems.

Summary

The No Free Lunch theorem argues that no single algorithm can optimize better than any other, if we compare them on all possible objective functions. But it turns out that most "possible" functions are uninteresting; they fail to qualify to be considered problems. Given the subset of situations that researchers really do concern themselves with, it *is* possible to demonstrate that one algorithm has the advantage over another. One way to find these differences is by trying the algorithms on sets of problems that are known to be difficult, for different reasons; one may

have very many local optima, for instance, while another features complex interactions among variables. Several kinds of algorithms have proven themselves superior in their ability to optimize various kinds of difficult functions. Simulated annealing, various evolutionary computation methods, and now the particle swarm are among these.

The particle swarm algorithm imitates human social behavior. Individuals interact with one another while learning from their own experience, and gradually the population members move into better regions of the problem space. The algorithm is extremely simple—it can be described in one straightforward formula—but it is able to surmount many of the obstacles that optimization problems commonly present. In Chapter 8 we will see that the simple formula generates rich and complex effects. Researchers have investigated numerous ways to manipulate the search trajectories of the particles, and some of these ways have resulted in improvements and insights.