CHAPTER 11

Particle Swarm Optimization

The particle swarm algorithm imitates human social behavior.
—James Kennedy and Russell Eberhart [Kennedy and Eberhart, 2001]

We observe collective intelligence in many natural systems. For example, ants exhibit an extraordinary level of collective intelligence, as we discussed at the beginning of Chapter 10. In such systems intelligence does not reside in individuals but is instead distributed among a group of many individuals. This can be seen in flocks of animals as they avoid predators, seek food, seek to travel more quickly, and other behaviors.

Animal groups can often avoid predators more effectively in a group than alone. For example, it might be easy for a lion to recognize a single zebra because of its contrast with the surrounding landscape, but a group of zebras blend together and are more difficult to recognize as individuals [Stone, 2009]. A group of animals might also appear to be larger, or sound louder, or be more threatening in other ways, than a solitary animal. Finally, it may be difficult for a predator to focus on a single animal when it is part of a large group. These phenomena are called the predator confusion effect [Milinski and Heller, 1978]. [Heinrich, 2002] gives an interesting description of how antelope use the predator confusion effect.

Another way that groups protect themselves from predators is described by the many-eyes hypothesis [Lima, 1995]. When a large group forages for food or drinks from a stream, random effects dictate that there will always be a few animals who are watching for predators. This collaboration not only provides more protection from predators, but also allows each individual more time for feeding and drinking.

Finally, groups protect themselves from predators because of the encounter dilution effect [Krause and Ruxton, 2002]. This can take several forms. First, individual animals might seek the cover and protection of a group as a type of selfish behavior to reduce their chance of being attached [Hamilton, 1971]. Second, as a predator wanders through its territory, it might be less likely to encounter a single group than one of many individuals scattered throughout the territory [Turner and Pitcher, 1986].

Animals also have more success in finding food when they are in groups than when they are alone. At first glance, this might not seem correct. After all, when an individual is in a group, it cannot approach its prey stealthily; and when it catches its prey, it has to share the food with others in the group. However, the success of groups in foraging is related to the many-eyes hypothesis in predator avoidance. With more eyes searching for food, the group has a disproportionately greater chance of success than a single animal that searches for food by itself [Pitcher and Parrish, 1993]. Also, a group increases its chances of success if it can surround its prey.

Animals can also move more quickly when in groups than when alone. This is seen in bicycle riders who ride in a line and draft off of each other. The trailing riders might expend as much as 40% less energy than the lead rider because of wind resistance [Burke, 2003]. The same type of effect, albeit to a lesser extent, can be seen in speed skating, running, swimming, and other sports. In the animal world, drafting can be seen in groups of geese as they fly [McNab, 2002], groups of ducks as they paddle [Fish, 1995], and groups of fish as they swim [Noren et al., 2008].

Particle swarm optimization (PSO) is based on the observation that groups of individuals work together to improve not only their collective performance on some task, but also each individual performance. The principles of PSO are clearly seen not only in animal behavior but also in human behavior. As we try to improve our performance at some task, we adjust our approach based on some basic ideas.

- Inertia. We tend to stick to the old ways that have proven to be successful in the past. "I've always done it this way, and so that is how I am going to continue doing things."

- Influence by society. We hear about others who have been successful and we try to emulate their approaches. We may read about the success of others in books, or on the internet, or in the newspaper. "If it worked for them, then maybe it will work for me too."

- Influence by neighbors. We learn the most from those who are personally close to us. We are influenced more by our friends than by society. In our conversations with others, we share stories of success and failure, and we modify our behavior because of those conversations. Investment advice from our millionaire neighbor or cousin will have a stronger influence on us than the more distant stories of billionaires that we read on the internet.

Overview of the Chapter

Section 11.1 gives a basic overview of PSO and some simple examples. Section 11.2 discusses ways of limiting the velocity of PSO particles, which is necessary for good optimization performance. Section 11.3 discusses inertia weighting and constriction coefficients, which are two features of PSO that indirectly limit particle velocities. Section 11.4 discusses the global PSO algorithm, which is a PSO generalization that uses the best individual at each generation to update each individual's velocity. Section 11.5 discusses the fully informed PSO algorithm, in which every individual's velocity contributes to every other individual's velocity each generation. Section 11.6 approaches PSO learning from the other direction – if we can learn from others' successes, then we can also learn from their mistakes.

11.1 A BASIC PARTICLE SWARM OPTIMIZATION ALGORITHM

Suppose that we have a minimization problem that is defined over a continuous domain of d dimensions. We also have a population of N candidate solutions, denoted as $\{x_i\}$, $i \in [1, N]$. Furthermore, suppose that each individual x_i is moving with some velocity v_i through the search space. This movement through search space is the essence of PSO, and it is the fundamental difference between PSO and other EAs. Most other EAs are more static than PSO because they model candidate solutions and their evolution from one generation to the next, but they do not model the dynamics of the movement of the candidate solutions through the search space.

As a PSO individual moves through the search space, it has some inertia and so it tends to maintain its velocity. However, its velocity can change due to a couple of different factors.

- First, it remembers its best position in the past, and it would like to change its velocity to return to that position. This is similar to the human tendency to remember the good old days, and to try to recapture the experiences of the past. In PSO, an individual travels through the search space, and its position in the search space changes from one generation to the next. However, the individual remembers its performance from past generations, and it remembers the search space location at which it obtained its best performance in the past.

- Second, an individual knows the best position of its neighbors at the current generation. This requires the definition of a neighborhood size, and it requires that all of the neighbors communicate with each other about their performance on the optimization problem.

These two effects randomly influence an individual's velocity and are similar to our own social interactions. Sometimes we feel more stubborn than at other times, and so we are not strongly influenced by our neighbors. Other times we feel more nostalgic than at other times, and so we are more strongly influenced by our past successes. We summarize the basic PSO algorithm in Figure 11.1.

```
Initialize a random population of individuals $\{x_i\}$, $i \in [1, N]$
Initialize each individual's $n$-element velocity vector $v_i$, $i \in [1, N]$
Initialize the best-so-far position of each individual: $b_i \leftarrow x_i$, $i \in [1, N]$
Define the neighborhood size $\sigma < N$
Define the maximum influence values $\phi_{1,\max}$ and $\phi_{2,\max}$
Define the maximum velocity $v_{\max}$
While not(termination criterion)
    For each individual $x_i$, $i \in [1, N]$
        $H_i \leftarrow \{\sigma$ nearest neighbors of $x_i\}$
        $h_i \leftarrow \arg\min_x \{f(x) : x \in H_i\}$
        Generate a random vector $\phi_1$ with $\phi_1(k) \sim U[0, \phi_{1,\max}]$ for $k \in [1, n]$
        Generate a random vector $\phi_2$ with $\phi_2(k) \sim U[0, \phi_{2,\max}]$ for $k \in [1, n]$
        $v_i \leftarrow v_i + \phi_1 \circ (b_i - x_i) + \phi_2 \circ (h_i - x_i)$
        If $|v_i| > v_{\max}$ then
            $v_i \leftarrow v_i v_{\max} / |v_i|$
        End if
        $x_i \leftarrow x_i + v_i$
        $b_i \leftarrow \arg\min\{f(x_i), f(b_i)\}$
    Next individual
Next generation
```

Figure 11.1 A basic particle swarm optimization algorithm for minimizing the n-dimensional function $f(x)$, where x_i is the i-th candidate solution and v_i is its velocity vector. The notation $a \circ b$ means element-by-element multiplication of the vectors a and b.

Figure 11.1 shows that there are several tuning parameters in the PSO algorithm.

- Not only do we have to initialize a population, as with every other EA, but we also have to initialize the population's velocity vectors. There are several ways to initialize velocities. For example, they could be initialized randomly, or they could be initialized to zero [Helwig and Wanka, 2008].

- We have to define the neighborhood size σ of the algorithm. Note that the term "neighborhood size" is ambiguous. Sometimes it means that each individual has σ close neighbors, and sometimes it means that since there are a total of σ individuals in the neighborhood, each individual has $(\sigma - 1)$ close neighbors. One of the initial PSO papers indicates that smaller neighborhoods (as small as two) provide better global behavior and avoid local minima, while larger neighborhoods provide faster convergence [Eberhart and Kennedy, 1995].

- We have to choose the maximum learning rates $\phi_{1,\max}$ and $\phi_{2,\max}$. The parameter ϕ_1, which is called the cognition learning rate, and ϕ_2, which is called the social learning rate, are random numbers distributed in $[0, \phi_{1,\max}]$ and $[0, \phi_{2,\max}]$ respectively. We discuss these further in Section 11.3.3, but for now we simply note the rule of thumb that $\phi_{1,\max}$ and $\phi_{2,\max}$ are often set to about 2.05.

- We have to choose the maximum velocity v_{max}. Empirical evidence indicates that each element of v_{max} should be limited to the corresponding dynamic range of the search space [Eberhart and Shi, 2000]. This seems intuitive; if v_{max} were greater than the dynamic range of the search space, then a particle could easily leave the search space in a single generation. Other results suggest setting v_{max} to between 10% and 20% of the search space range [Eberhart and Shi, 2001]. There are some problems for which we do not have a search space range in mind; that is, we do not have any idea ahead of time about the location of the optimum for which we are searching. In this case we should still enforce a finite v_{max} for best performance [Carlisle and Dozier, 2001].

- We could simplify the velocity update of Figure 11.1 as follows:

$$v_i \leftarrow v_i + \phi_1(b_i - x_i) + \phi_2(h_i - x_i) \tag{11.1}$$

  where ϕ_1 and ϕ_2 are scalars instead of vectors with $\phi_1 \sim U[0, \phi_{1,\text{max}}]$ and $\phi_2 \sim U[0, \phi_{2,\text{max}}]$. In this option, called linear PSO [Paquet and Engelbrecht, 2003], each element of the velocity vector v_i is updated with the same ϕ_1 and ϕ_2 values. However, linear PSO is generally considered to provide worse performance than the standard algorithm of Figure 11.1.

- As with most other EAs, elitism often improves the performance of PSO. We have not shown elitism in Figure 11.1, but we can easily implement elitism as discussed in Section 8.4.

- The update equation $x_i \leftarrow x_i + v_i$ in Figure 11.1 may result in x_i moving outside of the search domain. We usually implement some type of limiting operation to keep x_i within the search domain. For instance, we could include the following two equations after the update equation:

$$\begin{aligned} x_i &\leftarrow \min(x_i, x_{\text{max}}) \\ x_i &\leftarrow \max(x_i, x_{\text{min}}) \end{aligned} \tag{11.2}$$

  where $[x_{\text{min}}, x_{\text{max}}]$ defines the limits of the search domain.

Particle Swarm Topologies

Figure 11.1 shows that each particle is influence by its σ nearest neighbors. The arrangement of the neighbors that influence a particle is call the *topology* of the swarm. Since the neighborhood of each particle in Figure 11.1 changes each generation, it is called a dynamic topology. Since the neighborhood is local (that is, it does not include the entire swarm), it is also called an *lbest* topology.

We can use many other methods to define the neighborhood of each particle [Akat and Gazi, 2008]. For instance, we could define neighborhoods at the beginning of the algorithm so that the neighborhoods are static and do not change from one generation to the next. Or, if the optimization process stagnates, we could at that time randomly redefine the neighborhoods [Clerc and Poli, 2006]. In the extreme case we can have a single neighborhood that encompasses the entire swarm, which means that H_i in Figure 11.1 is equal to the entire swarm for all i, and h_i is independent of i and is equal to the best particle among the entire population. This is called the *all* topology or the *gbest* topology. This is the topology with which PSO was originally developed, and it is still widely used. Another common topology is the*ring* topology, in which each particle is connected to two other particles. The *cluster* topology is one in which each particle is fully connected within its own cluster, while a few particles in each cluster are also connected to an additional particle in another cluster. The *wheel* topology is one in which a focal particle is connected to all other particles, while all of the other particles are connected only to the focal particle. The *square* topology, also called the *von Neumann* topology, is one in which each particle is connected to four neighbors. Figure 11.2 depicts some of these topologies. PSO performance can vary strongly with topology, and researchers have experimented with many other topologies besides the few that we mention here [Mendes et al., 2004], [del Valle et al., 2008].

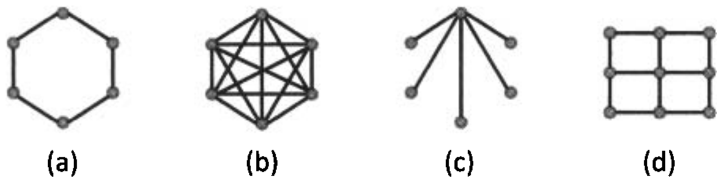

Figure 11.2 Some PSO topologies. (a) represents the *ring* topology, (b) represents the *all* topology, (c) represents the *wheel* topology, and (d) represents the *square* topology. The square topology wraps around from the top to the bottom, and from the left to the right, so that it forms a toroid with each particle connected to four neighbors. Each of these topologies can be either static or dynamic.

11.2 VELOCITY LIMITING

It has been found in many applications of PSO that if $v_{\max}$ is not used, PSO particles jump wildly around the search space [Eberhart and Kennedy, 1995]. To see why, consider the basic PSO algorithm of Figure 11.1, but with the simplification $\phi_2 = 0$. The position and velocity update in this case is

$$\begin{aligned} v_i(t+1) &= v_i(t) + \phi_1(b_i - x_i) \\ x_i(t+1) &= x_i(t) + v_i(t+1) \end{aligned} \tag{11.3}$$

where t is the generation number and ϕ_1 is the cognition learning rate. This can be written as

$$\begin{bmatrix} x_i(t+1) \\ v_i(t+1) \end{bmatrix} = \begin{bmatrix} 1-\phi_1 & 1 \\ -\phi_1 & 1 \end{bmatrix} \begin{bmatrix} x_i(t) \\ v_i(t) \end{bmatrix} + \begin{bmatrix} \phi_1 \\ \phi_1 \end{bmatrix} b_i. \tag{11.4}$$

The eigenvalues of the matrix on the right side of the above equation are

$$\lambda = \frac{2 - \phi_1 \pm \sqrt{\phi_1^2 - 4\phi_1}}{2} \tag{11.5}$$

and these eigenvalues govern the stability of the system.[1] If $\phi_1 \in [0, 4]$ then both eigenvalues have a magnitude of 1, which means that the system is marginally

[1]See any book on linear systems, or [Simon, 2006, Chapter 1], for a discussion of stability.

stable, and that $x_i(t)$ and $v_i(t)$ could become unbounded as $t \to \infty$, depending on initial conditions. If $\phi_1 > 4$, then one of the eigenvalues is greater than 1 in magnitude, which means that the system is unstable, and that $x_i(t)$ and $v_i(t)$ increase without bound for almost any initial conditions. This simple example shows why it could be important to use $v_{\max}$ to limit the magnitude of v_i, as shown in Figure 11.1.

However, this analysis assumes that b_i is not a function of x_i. Also, real implementations of PSO are more complicated than Equation (11.3), so our analysis may not be valid for general PSO algorithms. If $\phi_2 > 0$, or if an inertia weight less than 1 is used as we later discuss in Equation (11.9), then it may not be necessary to limit the velocity to get good performance [Carlisle and Dozier, 2001], [Clerc and Kennedy, 2002].

If we want to limit the velocity, then we could limit it in a couple of different ways. One way is to check the magnitude of v_i, and if it is greater than the scalar $v_{\max}$, then scale the components of v_i so that $|v_i| = v_{\max}$:

$$\text{If } |v_i| > v_{\max} \text{ then } v_i \leftarrow \frac{v_i v_{\max}}{|v_i|} \tag{11.6}$$

as shown in Figure 11.1. Another way is to limit the magnitude of each component of v_i. Recall that each individual in the population has n dimensions, so $v_i = \begin{bmatrix} v_i(1) & \cdots & v_i(n) \end{bmatrix}$.[2] With this approach, the maximum velocity of each dimension is specified, so we have $v_{\max}(j)$ defined for $j \in [1, n]$. Velocity limiting of the i-th particle is then performed as follows:

$$v_i(j) \leftarrow \begin{cases} v_i(j) & \text{if } |v_i(j)| \le v_{\max}(j) \\ v_{\max}(j)\,\text{sign}(v_i(j)) & \text{if } |v_i(j)| > v_{\max}(j) \end{cases} \quad \text{for } j \in [1, n]. \tag{11.7}$$

Velocity limiting can be viewed as a control over the exploration-exploitation balance of PSO. A large $v_{\max}$ allows more change in each individual from one generation to the next, which emphasizes exploration. A small $v_{\max}$ restricts changes in individuals, which emphasizes exploitation.

11.3 INERTIA WEIGHTING AND CONSTRICTION COEFFICIENTS

To avoid velocity limiting, we can modify the velocity update equation in Figure 11.1 to prevent the velocity from increasing without bound. In this section, we first discuss the use of inertia weighting in Section 11.3.1. Then we discuss the equivalent but more commonly-used constriction coefficient in Section 11.3.2. Finally, we present some conditions for the stability of the PSO algorithm in Section 11.3.3.

11.3.1 Inertia Weighting

An inertia weight is often used in PSO applications. As we see from the velocity update equation of Figure 11.1, a particle tends to maintain its velocity from one

[2]Note that j in the term $v_i(j)$ here indicates a specific element of the vector v_i, while t in the term $v_i(t)$ in Equation (11.3) indicates the value of v_i at the t-th generation. This notation is not consistent, but its meaning should be clear from the context.

generation to the next, although some velocity changes are allowed due to the learning rates:

$$v_i(k) \leftarrow v_i(k) + \phi_1(b_i - x_i(k)) + \phi_2(k)(h_i(k) - x_i(k)) \text{ for } k \in [1, n] \tag{11.8}$$

where n is the problem dimension. However, it has been found empirically that decreasing inertia during the optimization process may provide better performance. Equation (11.8) is thus modified to the following equation:

$$v_i(k) \leftarrow wv_i(k) + \phi_1(k)(b_i(k) - x_i(k)) + \phi_2(h_i(k) - x_i(k)) \tag{11.9}$$

where w is the inertia weight, which often decreases from about 0.9 at the first generation to about 0.4 at the last generation [Eberhart and Shi, 2000]. This helps slow down the velocity of each particle as the generation count increases, which improves convergence.

[Clerc and Poli, 2006] recommend PSO parameters for velocity updates of the form of Equation (11.9). In that paper, the population size is set to 30, the neighborhood size is set to four and the neighborhoods are fixed until the PSO process stagnates, at which time the neighborhoods are randomly reinitialized. The velocity update is implemented as shown in Equation (11.9) with the following recommended parameters [Clerc and Poli, 2006, Equation 19]:

$$\begin{aligned} w &= 0.72 \\ \phi_1(k) &\sim U[0, 1.108] \text{ for } k \in [1, n] \\ \phi_2(k) &\sim U[0, 1.108] \text{ for } k \in [1, n]. \end{aligned} \tag{11.10}$$

Reference [Clerc and Poli, 2006] also proposes other PSO variations for improved performance. The stability of PSO with the velocity update of Equation (11.9) is discussed in [Poli, 2008].

A more extensive set of recommended PSO parameters are found for velocity updates of the form of Equation (11.9) in [Pedersen, 2010] and are shown in Table 11.1. These recommended parameters apply when the neighborhood for each particle is the entire swarm, so that h_i (for all i) in Equation (11.9) is equal to the best individual in the population.

Problem Dimension	Function Evaluations	N	w	ϕ_1	ϕ_2
2	400	25	0.3925	2.5586	1.3358
		29	−0.4349	−0.6504	2.2073
2	4,000	156	0.4091	2.1304	1.0575
		237	−0.2887	0.4862	2.5067
5	1,000	63	−0.3593	−0.7238	2.0289
		47	−0.1832	0.5287	3.1913
5	10,000	223	−0.3699	−0.1207	3.3657
		203	0.5069	2.5524	1.0056
10	2,000	63	0.6571	1.6319	0.6239
		204	−0.2134	−0.3344	2.3259
10	20,000	53	−0.3488	−0.2746	4.8976
20	40,000	69	−0.4438	−0.2699	3.3950
20	400,000	149	−0.3236	−0.1136	3.9789
		60	−0.4736	−0.9700	3.7904
		256	−0.3499	−0.0513	4.9087
30	600,000	95	−0.6031	−0.6485	2.6475
50	100,000	106	−0.2256	−0.1564	3.8876
100	200,000	161	−0.2089	−0.0787	3.7637

Table 11.1 Recommended PSO parameters for various problem dimensions and available fitness function evaluations [Pedersen, 2010]. N is the population size, and w, ϕ_1, and ϕ_2 are the recommended parameters for Equation (11.9) when each particle's neighborhood is comprised of the entire swarm. The table shows that some problem configurations have more than one recommended set of parameters because multiple sets of parameters give almost the same performance on the benchmarks.

11.3.2 The Constriction Coefficient

Instead of Equation (11.9), inertia weighting is often implemented with a constriction coefficient. This implementation, which accomplishes the same thing as the inertia weight, involves writing the velocity update equation as

$$v_i \leftarrow K\left[v_i + \phi_1(b_i - x_i) + \phi_2(h_i - x_i)\right] \tag{11.11}$$

where K is called the constriction coefficient [Clerc, 1999], [Eberhart and Shi, 2000], [Clerc and Kennedy, 2002]. We have used the linear velocity update in Equation (11.11) to simplify our analysis. Equation (11.11) is equivalent to the linear form of Equation (11.9) if $K = w$, and if ϕ_1 and ϕ_2 in Equation (11.11) are replaced with ϕ_1/K and ϕ_2/K respectively. To analyze this approach, we use t to denote the generation number and we write Equation (11.11) as follows:

$$\begin{aligned} v_i(t+1) &= K\left[v_i(t) + (\phi_1+\phi_2)\left(\frac{\phi_1 b_i(t) + \phi_2 h_i(t)}{\phi_1+\phi_2} - x_i(t)\right)\right] \\ &= K\left[v_i(t) + \phi_T(p_i(t) - x_i(t))\right] \end{aligned} \tag{11.12}$$

where ϕ_T and $p_i(t)$ are defined by the above equation. Now we define

$$y_i(t) = p_i(t) - x_i(t). \tag{11.13}$$

Assuming that $p_i(t)$ is constant with time, we can combine Equations (11.12) and (11.13) to get

$$\begin{aligned} v_i(t+1) &= Kv_i(t) + K\phi_T y_i(t) \\ y_i(t+1) &= p_i - x_i(t+1) \\ &= p_i - x_i(t) - v_i(t+1) \\ &= y_i(t) - Kv_i(t) - K\phi_T y_i(t) \\ &= -Kv_i(t) + (1-K\phi_T)y_i(t). \end{aligned} \tag{11.14}$$

These equations for $v_i(t+1)$ and $y_i(t+1)$ can be combined to give

$$\begin{bmatrix} v_i(t+1) \\ y_i(t+1) \end{bmatrix} = \begin{bmatrix} K & K\phi_T \\ -K & 1-K\phi_T \end{bmatrix} \begin{bmatrix} v_i(t) \\ y_i(t) \end{bmatrix}. \tag{11.15}$$

The matrix on the right side of the above equation, which governs the stability of the system, has eigenvalues

$$\begin{aligned} \lambda &= \frac{1}{2}\left[1 - K(\phi_T - 1) \pm \sqrt{1 + K^2(\phi_T-1)^2 - 2K(\phi_T+1)}\right] \\ &= \frac{1}{2}\left[1 - K(\phi_T - 1) \pm \sqrt{\Delta}\right] \end{aligned} \tag{11.16}$$

where the discriminant Δ is defined by the above equation. We denote the eigenvalues as λ_1 and λ_2:

$$\begin{aligned} \lambda_1 &= \frac{1}{2}\left[1 - K(\phi_T - 1) + \sqrt{\Delta}\right] \\ \lambda_2 &= \frac{1}{2}\left[1 - K(\phi_T - 1) - \sqrt{\Delta}\right]. \end{aligned} \tag{11.17}$$

The dynamic system of Equation (11.15) is stable if $|\lambda_1| < 1$ and $|\lambda_2| < 1$. This analysis assumes that ϕ_T is constant. In the velocity update of Equation (11.11), the ϕ_i terms are random, but in this analysis we make the simplifying assumption that each ϕ_i is constant. See Problem 11.8 for a discussion of whether to use constant K or time-varying K in PSO. In the following section, we study the behavior of λ_1 and λ_2, which determines the stability of the PSO algorithm, as a function of the constriction coefficient K.

11.3.3 PSO Stability

We can use Equation (11.16) to make the following observation.

Observation 11.1 *When $K = 0$, we obtain $\Delta = 1$, $\lambda_1 = 1$, and $\lambda_2 = 0$.*

Next we consider the values of λ_1 and λ_2 as K increases from 0. Equation (11.16) shows that

$$\begin{aligned} \lim_{|K|\to\infty} \Delta &= \infty \\ \Delta &= 0 \text{ for } K = \frac{\phi_T + 1 \pm 2\sqrt{\phi_T}}{(\phi_T - 1)^2} = \{K_1, K_2\} \\ \Delta &< 0 \text{ for } K \in (K_1, K_2) \end{aligned} \tag{11.18}$$

where K_1 and K_2 are defined by the above equation. We assume that $\phi_T > 0$ so that $\sqrt{\phi_T}$ is real. Figure 11.3 shows a plot of Δ as a function of K. This leads us to the following observation.

Observation 11.2 *λ_1 and λ_2 are real for $K < K_1$.*

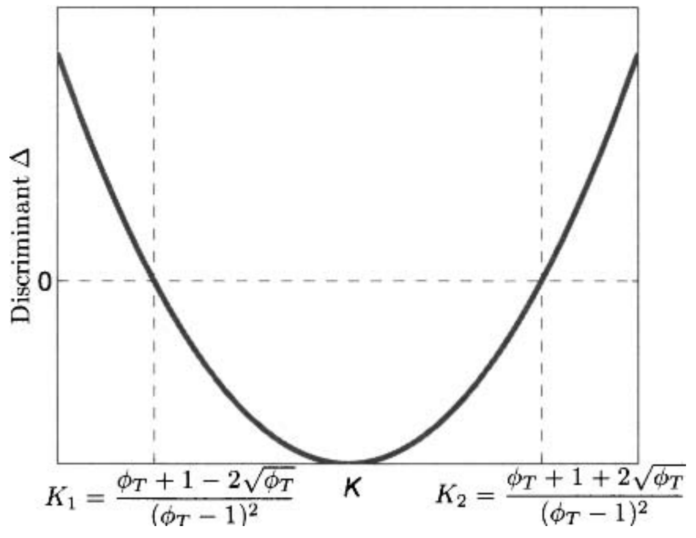

Figure 11.3 The discriminant Δ of Equation (11.16) as a function of the constriction coefficient K. $\Delta > 0$ for $K < K_1$ and $K > K_2$, which means that λ_1 and λ_2 are real. $\Delta < 0$ for $K \in (K_1, K_2)$, which means that λ_1 and λ_2 are complex.

When $K = K_1$, we see that $\Delta = 0$, which means that $\lambda_1 = \lambda_2$. In fact, it is easy to make the following observation from Equation (11.16).

Observation 11.3 *When $K = K_1$, we obtain $\lambda_1 = \lambda_2 = (1 - \sqrt{\phi_T})/(1 - \phi_T)$, which is between 0 and 1 for all $\phi_T \neq 1$.*

Now consider the behavior of λ_1 as K increases from 0 to K_1. Taking the derivative of λ_1 with respect to K, we see that

$$\frac{d\lambda_1}{dK} = \frac{1 - \phi_T}{2} - \frac{\phi_T - K(\phi_T - 1)^2 + 1}{\sqrt{K^2(\phi_T - 1)^2 - 2K(\phi_T + 1) + 1}}. \tag{11.19}$$

We can use basic algebraic manipulations to show that if $\phi_T > 1$, then this derivative is negative for $K < K_1$. We can similarly show that the derivative of λ_2 with respect to K is positive for $K < K_1$. This leads us to the following observation.

Observation 11.4 *If $\phi_T > 1$, then λ_1 and λ_2 are both between 0 and 1 for $K \in (0, K_1)$.*

Now consider the behavior of λ_1 and λ_2 as K increases from K_1. We see from Figure 11.3 that when $K \in (K_1, K_2)$, λ_1 and λ_2 are complex with the same magnitude, which can be derived as

$$|\lambda| = \frac{1}{2}\sqrt{[1 - K(\phi_T - 1)]^2 + 2K(\phi_T + 1) - 1 - K^2(\phi_T - 1)^2}. \qquad (11.20)$$

After some algebraic manipulations, this reduces to $|\lambda| = \sqrt{K}$. The derivative of this expression is positive for all $K > 0$. This leads us to the following observation.

Observation 11.5 *When $K \in (K_1, K_2)$, λ_1 and λ_2 are complex and have the same magnitude, which monotonically increases with K.*

Now consider the value of λ_1 and λ_2 when $K = K_2$. From Figure 11.3, we know that λ_1 and λ_2 are real and equal when $K = K_2$. In fact, it is easy to see from Equation (11.16) that when $K = K_2$, $\lambda_1 = \lambda_2 = (1 + \sqrt{\phi_T})/(1 - \phi_T)$. This is between 0 and -1 for all $\phi_T > 4$, which we state as follows.

Observation 11.6 *When $K = K_2$, we obtain $\lambda_1 = \lambda_2 = (1 + \sqrt{\phi_T})/(1 - \phi_T)$, which is between 0 and -1 if $\phi_T > 4$.*

Now consider the values of λ_1 and λ_2 when $K > K_2$. Both λ_1 and λ_2 are real for this range of K. Equation (11.19) gives the derivative of λ_1 with respect to K when λ_1 is real. We can perform some basic algebraic manipulations of Equation (11.19) to show that if $\phi_T > 1$ and $K > K_2$, then the derivative of λ_1 is positive and the derivative of λ_2 is negative. Combining this reasoning with Observation 11.6, we see that λ_1 remains less than 1 in magnitude for all values of $K > K_2$. However, λ_2 approaches $-\infty$ as $K \to \infty$, as can be seen from Equation (11.17). This gives us the following observation.

Observation 11.7 *When $K > K_2$, λ_1 is real and negative and less than 1 in magnitude, and λ_2 is real and negative and approaches $-\infty$ as $K \to \infty$.*

The limit of λ_1 as $K \to \infty$ can be derived from Equation (11.16):

$$\begin{aligned}\lambda_1 &= \frac{1}{2}\left[1 - K(\phi_T - 1) + \sqrt{1 + K^2(\phi_T - 1)^2 - 2K(\phi_T + 1)}\right] \\ &= \frac{1/K - (\phi_T - 1) + \sqrt{1/K^2 + (\phi_T - 1)^2 - 2(\phi_T + 1)/K}}{2/K} \\ &= \frac{N(K)}{D(K)} \qquad (11.21)\end{aligned}$$

where the numerator $N(K)$ and denominator $D(K)$ are defined by the above equation. The limit of both $N(K)$ and $D(K)$ is 0 as $K \to \infty$, so we can use l'Hopital's rule to evaluate the limit as follows.[3]

[3]Thanks to Steve Szatmary for deriving $\lim_{K\to\infty} \lambda_1$.

$$
\begin{aligned}
\frac{dN(K)}{dK} &= -K^{-2} + \frac{-2K^{-3} + 2(\phi_T + 1)K^{-2}}{2\sqrt{K^{-2} + (\phi_T - 1)^2 - 2(\phi_T + 1)K^{-1}}} \\
\frac{dD(K)}{dK} &= -2K^{-2} \\
\frac{dN(K)/dK}{dD(K)/dK} &= \frac{1}{2} + \frac{K^{-1} - \phi_T - 1}{2\sqrt{K^{-2} + (\phi_T - 1)^2 - 2(\phi_T + 1)K^{-1}}} \\
\lim_{K\to\infty} \lambda_1 &= \lim_{K\to\infty} \frac{dN(K)/dK}{dD(K)/dK} \\
&= \frac{1}{2} - \frac{\phi_T + 1}{2(\phi_T - 1)} \\
&= \frac{1}{1 - \phi_T} \qquad (11.22)
\end{aligned}
$$

which is less than one in magnitude if $\phi_T > 2$. This leads us to the following observation, which is an expansion of Observations 11.6 and 11.7.

Observation 11.8 *As K increases from K_2 to ∞, λ_1 monotonically increases from $(1 + \sqrt{\phi_T})/(1 - \phi_T)$ to $1/(1 - \phi_T)$, and λ_2 monotonically decreases from $(1 + \sqrt{\phi_T})/(1 - \phi_T)$ to $-\infty$.*

Since λ_2 decreases from $(1 + \sqrt{\phi_T})/(1 - \phi_T)$, which is greater than -1, to $-\infty$, λ_2 must be equal to -1 at some value of K, which we denote as K_3. Therefore, from Equation (11.17) we have

$$
-1 = \frac{1}{2}\left[1 - K_3(\phi_T - 1) - \sqrt{1 + K^2(\phi_T - 1)^2 - 2K(\phi_T + 1)}\right]. \qquad (11.23)
$$

Solving this equation for K_3 gives $K_3 = 2/(\phi_T - 2)$. Combining this with all of the above observations gives us the following theorem.

Theorem 11.1 *If b_i and h_i are constant in the velocity update of Equation (11.11), and if $\phi_T = \phi_1 + \phi_2 > 4$, then PSO is stable for*

$$
K < \frac{2}{\phi_T - 2}. \qquad (11.24)
$$

Figure 11.4 illustrates how the eigenvalues of Equation (11.15) change in the complex plane as K increases from 0 to ∞. Figure 11.5 shows how their their magnitudes change with K.

We can write K for a stable PSO algorithm as

$$
K = \frac{2\alpha}{\phi_T - 2}\ , \text{ where } \phi_T = \phi_{1,\max} + \phi_{2,\max} \qquad (11.25)
$$

with $\alpha \in (0, 1)$, so that α indicates how close the constriction coefficient K is to its theoretically maximum value before the PSO algorithm becomes unstable. A larger α allows more exploration, while a smaller α emphasizes more exploitation.

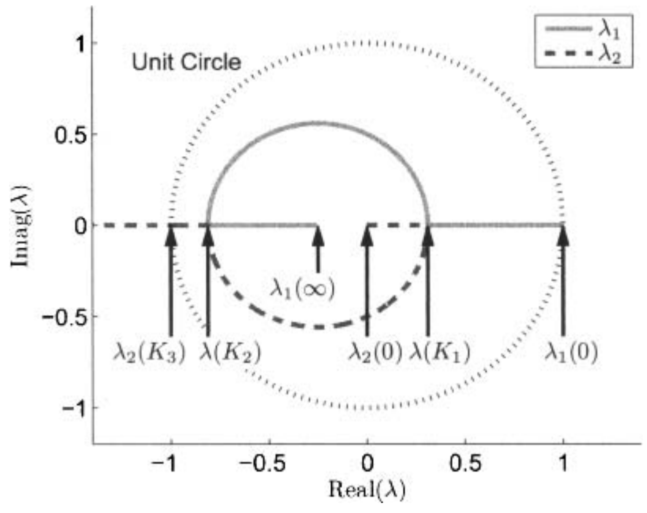

Figure 11.4 The eigenvalues of Equation (11.15) as the constriction coefficient K varies from 0 to ∞, illustrated for the case $\phi_T = 5$. At $K = 0$, $\lambda_1 = 1$ and $\lambda_2 = 0$. At $K = K_1$, $\lambda_1 = \lambda_2 > 0$. For $K \in (K_1, K_2)$, λ_1 and λ_2 are complex. At $K = K_2$, $\lambda_1 = \lambda_2 < 0$. At $K = K_3$, $\lambda_2 = -1$. As $K \to \infty$, $\lambda_1 \to 1/(1 - \phi_T)$ and $\lambda_2 \to -\infty$.

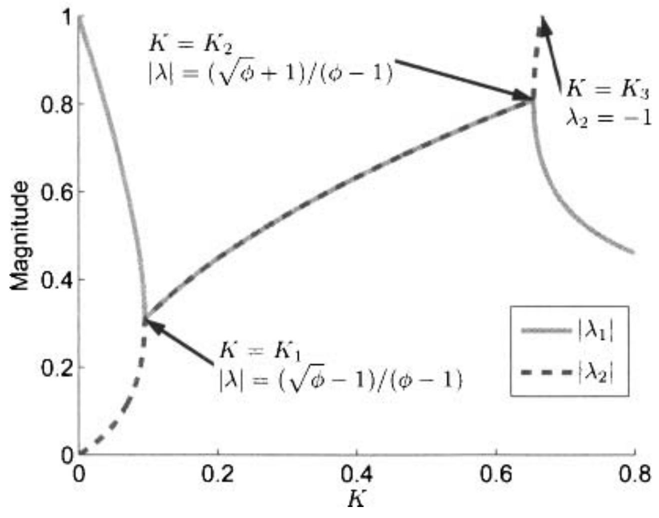

Figure 11.5 The magnitudes of the eigenvalues of Equation (11.15) as the constriction coefficient K increases from 0, illustrated for the case $\phi_T = 5$. This plot shows the magnitudes of the eigenvalues that are illustrated in Figure 11.4.

PSO algorithms are often presented in the books and research papers with the following recommendation [Carlisle and Dozier, 2001], [Clerc and Kennedy, 2002], [Eberhart and Shi, 2000], [Poli et al., 2007]:

$$\begin{aligned} \text{Common recommendation: } \phi_T &> 4 \\ K &< \frac{2}{\phi_T - 2 + \sqrt{\phi_T(\phi_T - 4)}}. \end{aligned} \tag{11.26}$$

This is equivalent to Theorem 11.1 as $\phi_T \to 4$, but Theorem 11.1 is more general for $\phi_T > 4$. Equation (11.26) does not provide any guidance for an upper bound for ϕ_T, or for how to allocate ϕ_T among $\phi_{1,\max}$ and $\phi_{2,\max}$. It is often recommended to set ϕ_T slightly larger than 4, and to allocate ϕ_T approximately equally among $\phi_{1,\max}$ and $\phi_{2,\max}$ – for example, $\phi_{1,\max} = \phi_{2,\max} = 2.05$. However, empirical results indicate there are some optimization problems for which better PSO performance can be obtained for values of ϕ_T that are much greater than 4.1, and for values of $\phi_{1,\max}$ and $\phi_{2,\max}$ that are far apart [Carlisle and Dozier, 2001]. Also note that our analysis takes only one specific approach, but other approaches with other assumptions lead to different stability conditions [Clerc and Poli, 2006].

11.4 GLOBAL VELOCITY UPDATES

One way that we can generalize the velocity update of Equation (11.11) is to write

$$v_i \leftarrow K\left[v_i + \phi_1(b_i - x_i) + \phi_2(h_i - x_i) + \phi_3(g - x_i)\right] \tag{11.27}$$

where g is the best individual found so far since the first generation. The analysis of the previous section is valid for Equation (11.27) if we define $\phi_T = \phi_{1,\max} + \phi_{2,\max} + \phi_{3,\max}$, and if we assume that $b_i + h_i + g$ is constant with time. The new term $\phi_3(g - x_i)$ adds a term to the velocity update equation that tends to drive each particle toward the best individual found so far since the first generation. This is conceptually similar to the stud EA, which uses the best individual at each generation for each recombination operation (Section 8.7.7). The difference is that g in Equation (11.27) is the best individual found since the first generation, while the stud in Section 8.7.7 is the best individual in the current generation. This similarity and difference could motivate the use of a more g-like operation in the stud EA, or the use of a more stud-like operation in the global PSO algorithm.

■ EXAMPLE 11.1

In this example we use PSO with the general velocity update of Equation (11.27) to optimize the 20-dimensional Ackley function. We use a population size of 50, an elitism parameter of 2, and a neighborhood size $\sigma = 4$. We use the nominal values

$$\begin{aligned} \phi_{1,\max} &= \phi_{2,\max} = \phi_{3,\max} = 2.1 \\ \phi_T &= \phi_{1,\max} + \phi_{2,\max} + \phi_{3,\max} \\ K &= \frac{2\alpha}{\phi_T - 2}, \quad \alpha = 0.9. \end{aligned} \tag{11.28}$$

Note that we can alternatively solve for ϕ_T in terms of K:

$$\phi_T = \frac{2(\alpha + K)}{K}. \tag{11.29}$$

Figures 11.6–11.9 show the average performance of PSO for various values of $\phi_{1,\max}$, $\phi_{2,\max}$, $\phi_{3,\max}$, and α, when the other parameters are equal to their nominal values. We see that the nominal values of Equation (11.28) are indeed approximately optimal for the 20-dimensional Ackley function.

Figures 11.6–11.8 show that when the $\phi_{\max}$ values are too small, the particles wander in an undirected manner. When they are too large, the particles are overly restricted and are unable to effectively explore the search space.

Figure 11.9 shows that when α (and hence K) is too small, the particles stagnate due to small velocities. When α (and hence K) is too large, the particles jump too aggressively through the search space.

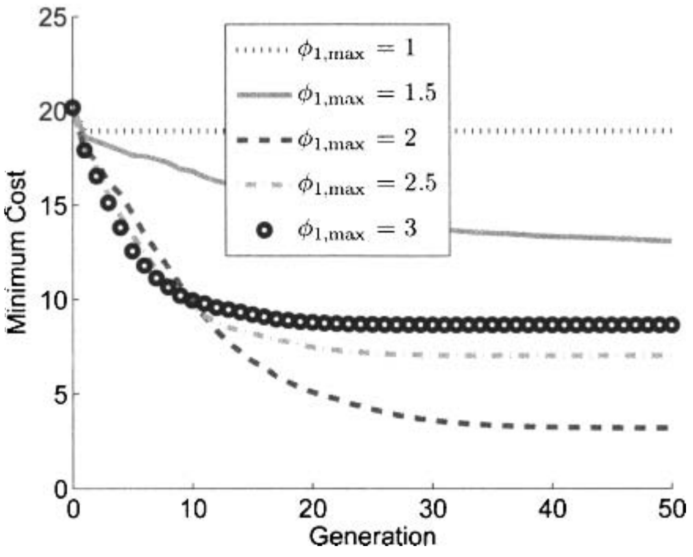

Figure 11.6 Example 11.1: Performance of PSO on the 20-dimensional Ackley function for various values of $\phi_{1,\max}$, averaged over 20 Monte Carlo simulations. $\phi_{1,\max} = 2$ is approximately optimal for this benchmark function.

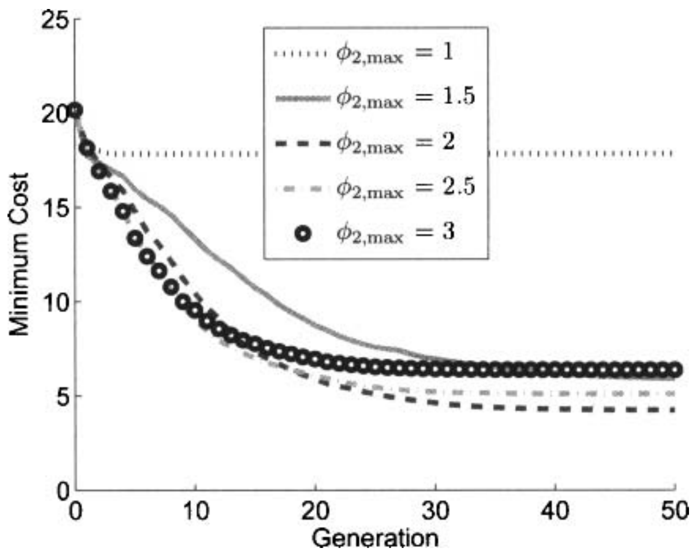

Figure 11.7 Example 11.1: Performance of PSO on the 20-dimensional Ackley function for various values of $\phi_{2,\max}$, averaged over 20 Monte Carlo simulations. $\phi_{2,\max} = 2$ is approximately optimal for this benchmark function.

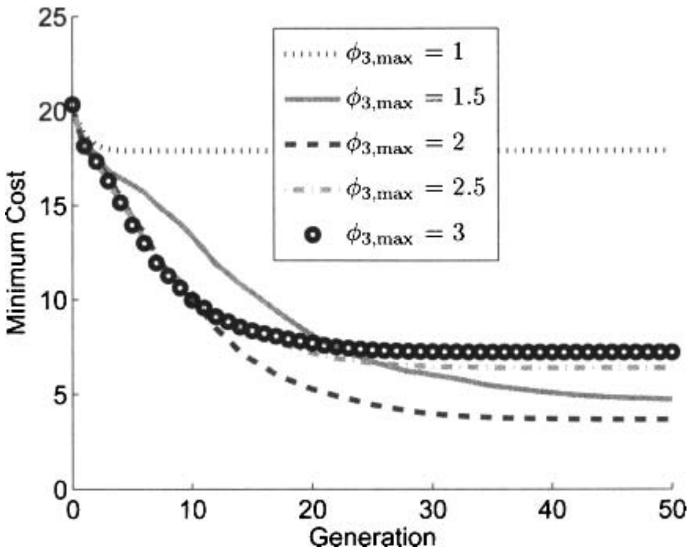

Figure 11.8 Example 11.1: Performance of PSO on the 20-dimensional Ackley function for various values of $\phi_{3,\max}$, averaged over 20 Monte Carlo simulations. $\phi_{3,\max} = 2$ is approximately optimal for this benchmark function.

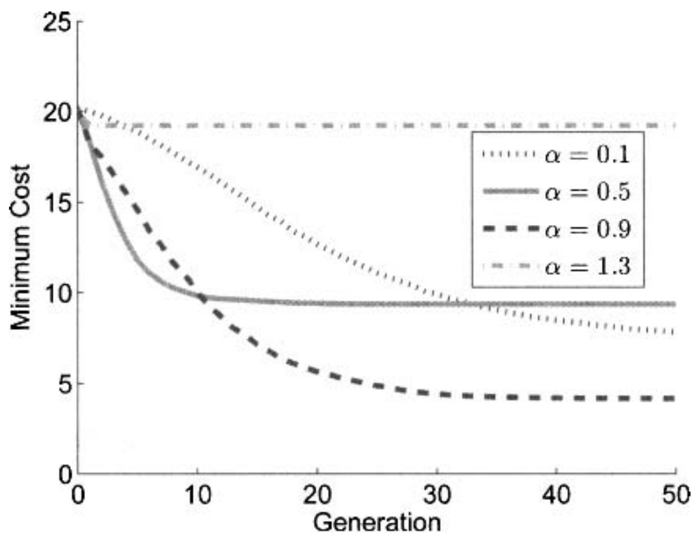

Figure 11.9 Example 11.1: Performance of PSO on the 20-dimensional Ackley function for various values of the constriction coefficient $K = \alpha K_{\text{max}}$, averaged over 20 Monte Carlo simulations. $K = 0.9K_{\text{max}}$ is approximately optimal for this benchmark function.

□

11.5 THE FULLY INFORMED PARTICLE SWARM

Equations (11.12) and (11.27) show that our most general (so far) form for the velocity update is

$$\begin{aligned} v_i(t+1) &= K\left[v_i(t) + \phi_T\left(p_i(t) - x_i(t)\right)\right] \\ \phi_T &= \phi_{1,\text{max}} + \phi_{2,\text{max}} + \phi_{3,\text{max}} \\ p_i(t) &= \frac{\phi_1 b_i(t) + \phi_2 h_i(t) + \phi_3 g(t)}{\phi_1 + \phi_2 + \phi_3}. \end{aligned} \tag{11.30}$$

We see that three particle positions contribute to the velocity update: the current individual's best position so far $b_i(t)$, the neighborhood's best current position $h_i(t)$, and the population's best position so far $g(t)$. This leads to the idea of making the velocity update more general. Why not allow every individual in the population to contribute to the velocity update? A generalization of Equation (11.30) can be written as

$$\begin{aligned} v_i(t+1) &= K\left[v_i(t) + \phi_T\left(p_i(t) - x_i(t)\right)\right] \\ \phi_T &= \frac{1}{N}\sum_{j=1}^{N} \phi_{j,\text{max}} \\ p_i(t) &= \frac{\sum_{j=1}^{N} w_{ij}\phi_j b_j(t)}{\sum_{j=1}^{N} w_{ij}\phi_j} \end{aligned} \tag{11.31}$$

where $b_j(t)$ is the best solution found so far by the j-th particle:

$$b_j(t) = \arg\min_x f(x) : x \in \{x_j(0), \cdots, x_j(t)\}. \tag{11.32}$$

Note the $1/N$ factor in the definition of ϕ_T in Equation (11.31), which is an ad-hoc approach to maintaining a reasonable balance between the contribution of $v_i(t)$ and $(p_i(t) - x_i(t))$ to the new velocity $v_i(t+1)$. The ϕ_j parameters in Equation (11.31) are random influence factors that are taken from the uniform distribution $U[0, \phi_{j,\max}]$. As indicated in Example 11.1, we often use

$$\begin{aligned} \phi_{j,\max} &\approx 2 \\ K &= 2\alpha/(3\phi_T - 2) \end{aligned} \tag{11.33}$$

where $\alpha \in (0,1)$. The factor of 3 in the value of K compensates for the fact that in Equation (11.27) ϕ_T is the sum of three $\phi_{j,\max}$ terms, while in Equation (11.31) it is the average of the $\phi_{j,\max}$ terms. The w_{ij} weights in Equation (11.31) are deterministic factors that describe the influence of the j-th particle on the velocity of the i-th particle. Sometimes we use w_{ij} = constant for all j. Other times, we want w_{ij} to be larger for values of j that correspond to better x_j particles, and also larger for values of j that correspond to x_j particles that are closer to x_i. For instance, if our problem is a minimization problem, we could use something like

$$w_{ij} = \left[\max_k f(x_k) - f(x_j)\right] + \left[\max_k |x_i - x_k| - |x_i - x_j|\right] \tag{11.34}$$

where $|\cdot|$ is a distance measurement. We might also need to weight the cost and fitness contributions appropriately so that they both contribute equal orders of magnitude to w_{ij}. For example,

$$\begin{aligned} S_i &= \frac{\max_k f(x_k) - \min_k f(x_k)}{\max_k |x_i - x_k|} \\ w_{ij} &= \left[\max_k f(x_k) - f(x_j)\right] + S_i \left[\max_k |x_i - x_k| - |x_i - x_j|\right]. \end{aligned} \tag{11.35}$$

S_i is a scale factor that makes the two terms that contribute to w_{ij} approximately equal. Since Equation (11.31) allows every particle to influence every other particle, it is called the fully informed particle swarm (FIPS) [Mendes et al., 2004]. This idea is reminiscent of global uniform recombination in EAs (Section 8.8.6).

■ EXAMPLE 11.2

In this example, we use the fully informed particle swarm of Equation (11.31) with the weights of Equation (11.35) to optimize the 20-dimensional Ackley function. We use a population size of 40 and an elitism parameter of 2. We use the nominal values

$$\begin{aligned} \phi_{j,\max} = \phi_{\max} &= 2, \quad \text{for } j \in [1, 20] \\ K &= 2\alpha/(3\phi_{\max} - 2), \quad \alpha = 0.9. \end{aligned} \tag{11.36}$$

Figures 11.10 and 11.11 show the average performance of PSO for various values of $\phi_{\max}$ and α, when the other parameter is equal to its nominal value.

Figure 11.10 shows that when $\phi_{\max}$ is small, the swarm converges very quickly, but it converges to a poor solution. As $\phi_{\max}$ increases, initial convergence slows, but the final converged solution becomes better. This may motivate us to use an adaptive $\phi_{\max}$ that is initially small and then gradually increases over time. Figure 11.11 shows that for small values of α, convergence is very slow. Convergence is fastest for $\alpha = 0.9$, but the final solution is better for $\alpha = 0.5$.

These results are very specific. They apply for a specific benchmark function with a specific dimension, a specific elitism parameter, and a specific form for the w_{ij} weighting parameters (Equation (11.35)). Additional experimentation is needed to see if the conclusions for this example can be generalized to a wider range of problems.

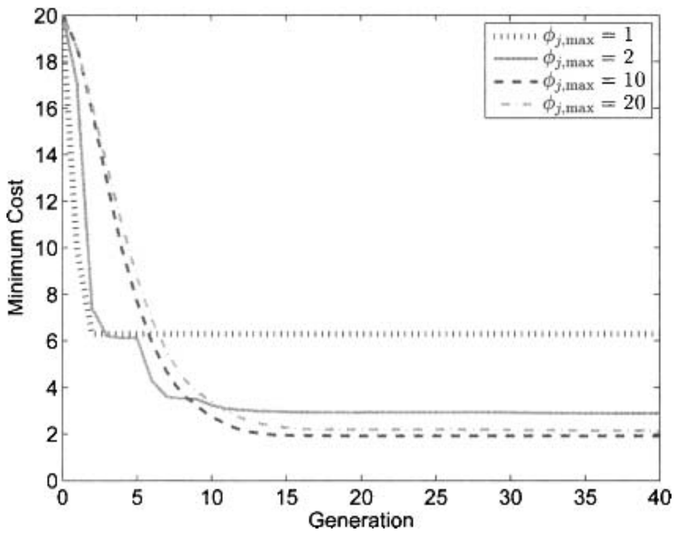

Figure 11.10 Example 11.2: Performance of the fully informed particle swarm on the 20-dimensional Ackley function for various values of $\phi_{\max}$, averaged over 20 Monte Carlo simulations. $\phi_{\max} = 1$ gives the best short-term performance, but larger values of $\phi_{\max}$ give better long-term performance.

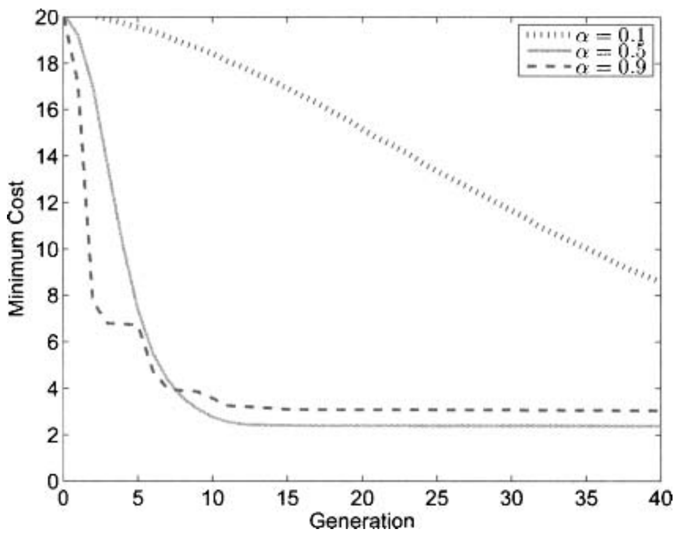

Figure 11.11 Example 11.2: Performance of the fully informed particle swarm on the 20-dimensional Ackley function for various values of α, averaged over 20 Monte Carlo simulations. $\alpha = 0.9$ gives the best short-term behavior, and $\alpha = 0.5$ gives the best long-term behavior.

□

Sometimes fully informed PSO is written differently than Equation (11.31). For example, Equation (11.31) can be replaced with the following [Poli et al., 2007]:

$$v_i(t+1) = K\left[v_i(t) + \frac{1}{n_i}\sum_{j=1}^{n_i} \phi_j \left(b_{i,j}(t) - x_i(t)\right)\right] \tag{11.37}$$

where n_i is the neighborhood size of the i-th particle, ϕ_j is taken from the uniform distribution $U[0, \phi_{\max}]$, and $b_{i,j}(t)$ is the best solution found so far by the j-th neighbor of the i-th particle. In this formulation, each particle has a certain fixed neighborhood, and each neighbor's best solution $b_{i,j}(t)$ has an equally weighted contribution (on average) to the velocity update of the i-th particle. Note that Equation (11.37) is equivalent to Equation (11.11) under certain conditions. Some papers have found that fully informed PSO performs poorly because the particles experience too many conflicting attractions, or because the search space of each particle decreases with increasing neighborhood size [de Oca and Stützle, 2008].

11.6 LEARNING FROM MISTAKES

PSO is based on the idea that biological organisms tend to repeat strategies that have proven successful in the past. This includes beneficial strategies that they have used themselves, and also beneficial strategies that they have observed in others. The basic equation for updating velocity, as we see in Equation (11.27), is

$$v_i \leftarrow K\left[v_i + \phi_1(b_i - x_i) + \phi_2(h_i - x_i) + \phi_3(g - x_i)\right] \tag{11.38}$$

where x_i and v_i are the position and velocity of the i-th particle; b_i is the previous best position of the i-th particle; h_i is the current best position of the i-th neighborhood; g is the previous best position of the entire swarm; and K, $\phi_{1,\max}$, $\phi_{2,\max}$, and $\phi_{3,\max}$ are positive tuning parameters.

However, biological organisms not only learn from successes, but also learn from mistakes. We tend to avoid strategies that have proven harmful in the past. This includes detrimental strategies that we have used ourselves, and also detrimental strategies that we have observed in others. A natural extension of PSO is to incorporate this avoidance of negative behavior in the basic PSO algorithm. This algorithm has been called "new PSO" in [Yang and Simon, 2005], [Selvakumar and Thanushkodi, 2007], but the term "new" is overused and nondescriptive, so we refer to it as "negative reinforcement PSO" (NPSO) in this section.

In NPSO, each particle adjusts its velocity not only in the direction of the best position of itself and its neighbors, but also away from the direction of the worst position of itself and its neighbors. Equation (11.38) is therefore modified to

$$\begin{aligned} v_i \quad \leftarrow \quad & K\left[v_i + \phi_1(b_i - x_i) + \phi_2(h_i - x_i) + \phi_3(g - x_i)\right. \\ & \left. -\phi_4(\bar{b}_i - x_i) - \phi_5(\bar{h}_i - x_i) - \phi_6(\bar{g} - x_i)\right] \end{aligned} \tag{11.39}$$

where $\bar{b}_i$ is the previous worst position of the i-th particle; $\bar{h}_i$ is the current worst position of the i-th neighborhood; $\bar{g}$ is the previous worst position of the entire swarm; each ϕ_j is taken from a uniform distribution on $(0, \phi_{j,\max})$; and each $\phi_{j,\max}$ is a positive tuning parameter.

We have to find a balance between the velocity adjustment towards beneficial solutions that comes from standard PSO, and the velocity adjustment away from detrimental solutions that we have added to NPSO. This balance is something that we all try to find in our everyday lives. How much do we focus on success and try to emulate it, compared to how much we focus on failure and try to avoid it? Most of us agree that positive reinforcement is more effective than negative reinforcement, but most of us also agree that both types of reinforcement are important for learning.

■ EXAMPLE 11.3

In this example, we use the NPSO of Equation (11.39) to optimize the 20-dimensional Schwefel 2.26 function. We use a population size of 20 and an elitism parameter of 2. We use the nominal values

$$\begin{aligned} \phi_{1,\max} = \phi_{2,\max} = \phi_{3,\max} &= 2 \\ \phi_{4,\max} = \phi_{5,\max} = \phi_{6,\max} &= 0 \\ K = \frac{2\alpha}{\phi_{1,\max} + \phi_{2,\max} + \phi_{3,\max} - 2}, \quad & \alpha = 0.9. \end{aligned} \tag{11.40}$$

Figures 11.12–11.14 show the average performance of NPSO for various values of $\phi_{4,\max}$, $\phi_{5,\max}$, and $\phi_{6,\max}$, when the other parameters are equal to their nominal values. Figure 11.12 shows that when $\phi_{4,\max}$, which determines how much each particle avoids its previous worst position, is increased above its nominal value of 0, it can result in a large improvement in performance. Figure 11.13 shows a similar but less dramatic improvement when $\phi_{5,\max}$, which determines how much each particle avoids the current worst position of its neighborhood, is increased beyond its nominal value of 0. Finally, Figure 11.14 shows that performance also improves when $\phi_{6,\max}$, which determines how much each particle avoids the previous worst position of the entire swarm, is increased beyond its nominal value of 0 . It appears from these figures that $\phi_{6,\max}$ has the greatest effect on NPSO performance.

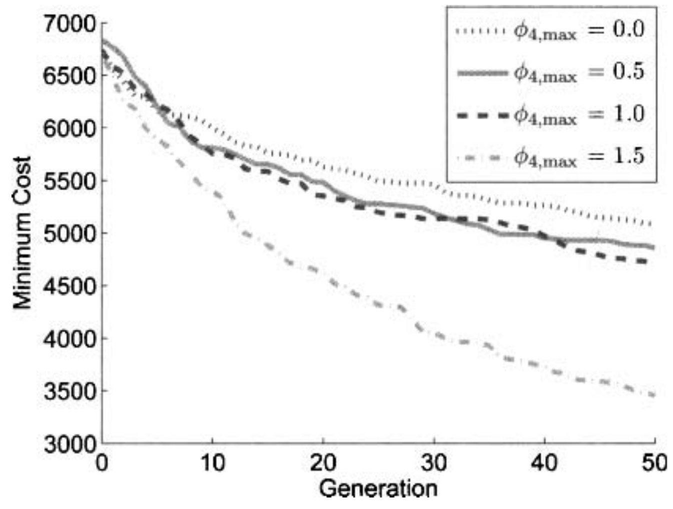

Figure 11.12 Example 11.3: Performance of NPSO on the 20-dimensional Schwefel 2.26 function for various values of $\phi_{4,\max}$, averaged over 20 Monte Carlo simulations. Particles that avoid their own previous worst position perform significantly better than particles that do not.

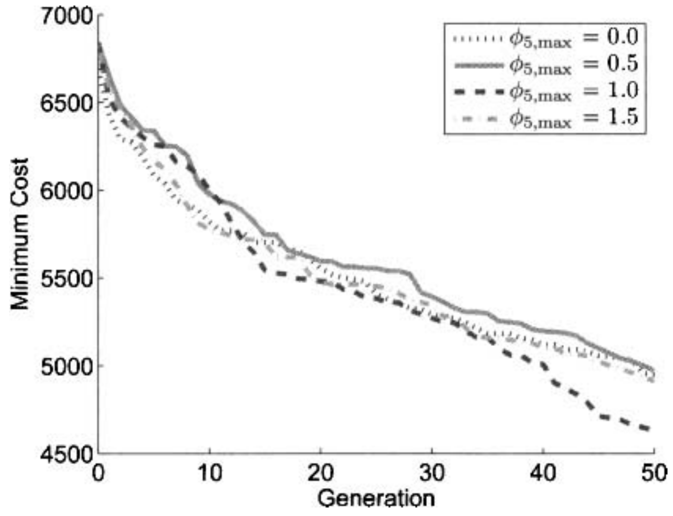

Figure 11.13 Example 11.3: Performance of NSPO on the 20-dimensional Schwefel 2.26 function for various values of $\phi_{5,\max}$, averaged over 20 Monte Carlo simulations. Particles that avoid the current worst position of their neighbors perform noticeably better than particles that do not.

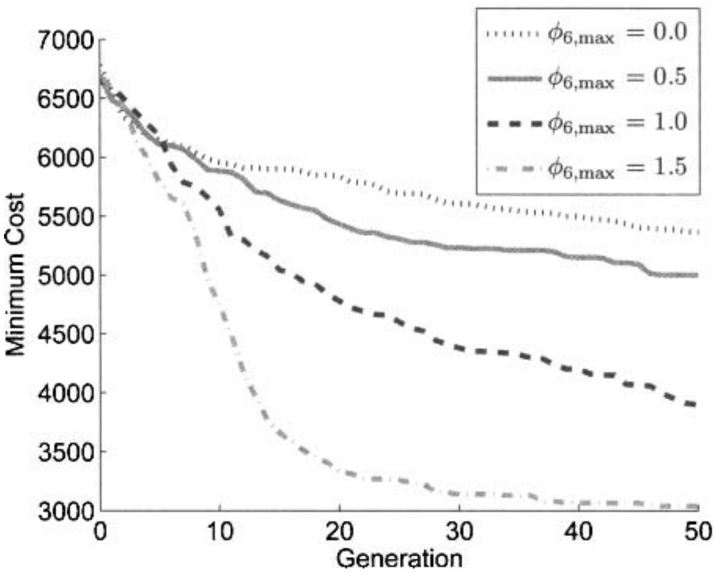

Figure 11.14 Example 11.3: Performance of NSPO on the 20-dimensional Schwefel 2.26 function for various values of $\phi_{6,\max}$, averaged over 20 Monte Carlo simulations. Particles that avoid the previous worst position of the swarm perform significantly better than particles that do not.

□

Example 11.3 indicates that the NPSO can perform much better than standard PSO. Notice in Example 11.3 that we changed only one of the negative reinforcement terms at a time while leaving the other two equal to zero. We have not tried combining nonzero values of $\phi_{4,\max}$, $\phi_{5,\max}$, and $\phi_{6,\max}$, but we leave this for future research by the reader. Also note that we could combine the idea of negative reinforcement with the fully informed PSO of Equation (11.31). We also leave this extension to the reader for further research. Finally, it would be interesting to rederive the stability results of Section 11.3 for NPSO; this is another area for future research.

11.7 CONCLUSION

PSO has proven itself to be an effective EA for a variety of problems. Any investigation of a newly proposed EA should include a comparison with PSO because of its good performance. Similar to ant colony optimization, some researchers do not consider PSO as an evolutionary algorithm, but instead consider it to be a type of swarm intelligence. It is true that PSO particles do not directly share candidate solution information with each other. However, PSO does include fitness-based selection, and PSO particles do share velocity information with each other, and velocity information directly affects the solutions. Therefore, we categorize PSO as an EA in this book.

Catfish PSO is a modification that was introduced to combat stagnation in PSO [Yang et al., 2011]. In a holding tank, sardines often settle into a locally optimal behavior and location, but then become lethargic and experience rapidly degrading health. If catfish are added to the tank, the sardines experience a renewed sense of stimulation and remain healthy for a longer period of time. Catfish PSO is based on this observation, and stimulates a PSO population when it stagnates. If the best individual in the PSO population has not improved for m consecutive generations (m is often between 3 and 7), then each independent variable of the worst 10% of the population is set equal to one of the boundaries of the search domain. The reason the particles are moved to the boundaries of the search domain is to maximize the search space. Also, optimization problems with constraints often have solutions that lie on a constraint boundary [Bernstein, 2006].

All of the discussion in this chapter has focused on PSO for continuous-domain problems. PSO has been extended in several different ways for combinatorial optimization [Kennedy and Eberhart, 1997], [Yoshida et al., 2001], [Clerc, 2004]. Other current research directions include simplifying the PSO algorithm [Pedersen and Chipperfield, 2010], hybridizing it with other EAs [Niknam and Amiri, 2010], adding mutation-like operators to avoid premature convergence [Xinchao, 2010], using multiple interacting swarms [Chen and Montgomery, 2011], removing randomness from the PSO algorithm [Clerc, 1999], using dynamic and adaptive topologies [Ritscher et al., 2010], exploring initialization strategies [Gutierrez et al., 2002], and adapting PSO parameters on-line [Zhan et al., 2009]. Also note that just as we model the velocity of each PSO particle, we could also model their accelerations [Tripathi et al., 2007]. Other future work could include particle swarm behavior and convergence analysis that takes the randomness of the algorithm into account, and that takes the relationships between the particles into account.

Additional recommended reading and study in the area of PSO includes books [Kennedy and Eberhart, 2001], [Clerc, 2006], [Sun et al., 2011]; and papers [Bratton and Kennedy, 2007], [Banks et al., 2007], [Banks et al., 2008]. Useful and extensive PSO web sites include [PSC, 2012] and [Clerc, 2012a].

PROBLEMS

Written Exercises

11.1 What are some arguments for having static neighborhoods in PSO? What are some arguments for having dynamic neighborhoods?

11.2 Acceleration in PSO:

a) How could you modify the PSO algorithm of Figure 11.1 to include acceleration?

b) Given this modification of the PSO algorithm, how would Equation (11.4) change, and what would be the eigenvalues?

11.3 Suppose that $\phi_1 = 4$ in Equation (11.4).

a) What are the eigenvalues of the matrix?

b) Is the system stable?

c) Give an initial condition and input b_i that will result in x_i and v_i being bounded as $t \to \infty$.

d) Give an initial condition and input b_i that will result in x_i and v_i being unbounded as $t \to \infty$.

11.4 Equation (11.35) uses the cost and distance of x_i to calculate the weight w_{ij}. What are some other features of x_i that we might consider using as part of the w_{ij} calculation?

11.5 Assuming that $p_i(t)$ is constant in Equation (11.30), write the dynamic state-space equations for $x_i(t+1)$ and $v_i(t+1)$. What are the eigenvalues of the system?

11.6 Under what conditions are Equations (11.11) and (11.37) equivalent?

11.7 Generalize the NPSO update of Equation (11.39) to obtain a fully-informed NPSO update equation.

11.8 Equation (11.25) recommends setting the constriction coefficient as follows:

$$K = \frac{2\alpha}{\phi_T - 2}$$

where $\alpha \in (0,1)$. We can set ϕ_T to the sum of the maximum possible values of the ϕ_i terms, in which case ϕ_T is constant for the PSO algorithm; or we can set ϕ_T to the sum of the actual ϕ_i terms that are randomly computed for each velocity update, in which case ϕ_T is different for each velocity update. Assuming that we use Equation (11.27) for our velocity update, these two options can be written as follows:

$$\begin{aligned} K_1 &= \frac{2\alpha_1}{\phi_{1,\max} + \phi_{2,\max} + \phi_{3,\max} - 2} \\ K_2 &= \frac{2\alpha_2}{\phi_1 + \phi_2 + \phi_3 - 2} \end{aligned}$$

where each ϕ_i is uniformly distributed on $[0, \phi_{i,\max}]$. What value of α_2 in the above equations makes $K_1 = K_2$ on average? (See Problem 11.12 for the computer exercise counterpart to this problem.)

Computer Exercises

11.9 Neighborhood Sizes: Simulate the PSO algorithm of Figure 11.1 for 40 generations to minimize the 10-dimensional sphere function (see Appendix C.1.1 for the definition of the sphere function). Use a population size of 20, and use the global velocity update of Equation (11.27). Use $\phi_{1,\max} = \phi_{2,\max} = \phi_{3,\max} = 2$, use $v_{\max} = \infty$, and use $\alpha = 0.9$ to find the constriction coefficient K. Run 20 Monte Carlo simulations for neighborhood sizes $\sigma = 0$, 5, and 10. Plot the average performance of each Monte Carlo set as a function of generation number. What do you conclude about the importance of local neighborhoods in PSO?

11.10 Fully Informed Particle Swarm Distance Weighting: Equation (11.35) can be written as

$$\begin{aligned} w_{ij} &= w_{ij}(c) + S w_{ij}(d) \\ \text{where } w_{ij}(c) &= \max_k f(x_k) - f(x_j) \\ w_{ij}(d) &= \max_k |x_i - x_k| - |x_i - x_j|. \end{aligned}$$

$w_{ij}(c)$ is the cost contribution of x_j to w_{ij}, and $w_{ij}(d)$ is the distance contribution. The above equation can be generalized as follows:

$$w_{ij} = (w_{ij}(c) + D S w_{ij}(d))/(1 + D)$$

where D is the importance of the distance contribution relative to the cost contribution. Use this weight formula to simulate the fully informed PSO to optimize the 20-dimensional Rastrigin function (see Appendix C.1.11 for the definition of the Rastrigin function). Run 20 Monte Carlo simulations for $D = 0$, 0.5, 1, 2, and 1000. Plot the average performance of each Monte Carlo set as a function of generation number, and provide some general observations about your results.

11.11 Fully Informed Particle Swarm Neighborhood Sizes: Implement the velocity update of Equation (11.37) in a PSO simulation to minimize the 20-dimensional Rosenbrock function (see Appendix C.1.4 for the definition of the Rosenbrock function). Use a population size of 20 and a generation count limit of 40. Tune $\phi_{\max}$ and K for good performance. Run 20 Monte Carlo simulations with neighborhood sizes of 2, 5, 10, and 20. Plot the average performance of each Monte Carlo set as a function of generation number, and provide some observations about your results.

11.12 Constant vs. Time-Varying Constriction: Simulate the PSO algorithm of Figure 11.1 for 50 generations to minimize the 10-dimensional Ackley function (see Appendix C.1.2 for the definition of the Ackley function). Use a population size of 20, and use the global velocity update of Equation (11.27). Use $\phi_{1,\max} = \phi_{2,\max} = \phi_{3,\max} = 2.1$, use $v_{\max} = \infty$, and use $\alpha_1 = 0.9$ to find the constriction coefficient K_1 that is defined in Problem 11.8. Run 20 Monte Carlo simulations with the constant constriction coefficient K_1, and run 20 Monte Carlo simulations with the time-varying constriction coefficient K_2 that you found in Problem 11.8. Plot the average performance of each Monte Carlo set as a function of generation number, and comment on your results.

CHAPTER 12

Differential Evolution

Compared to several existing EAs, DE is much simpler and straightforward to implement ... Simplicity of programming is important for practitioners from other fields, since they may not be experts in programming ....
—S. Das, P. Suganthan, and C. Coello Coello [Das et al., 2011]

Differential evolution (DE) was developed by Rainer Storn and Kenneth V. Price around 1995. Like many new optimization algorithms, DE was motivated by real-world problems: the solution of Chebyshev polynomial coefficients, and the optimization of digital filter coefficients. DE made a quick and impressive entrance into the world of EAs by finishing as one of the top entries at the First International Contest on Evolutionary Computation [Storn and Price, 1996] and at the Second International Contest on Evolutionary Optimization [Price, 1997]. The first DE publications were in conference proceedings [Storn, 1996a], [Storn, 1996b], and the first journal publication was a year later [Storn and Price, 1997]. However, the first DE publication that was widely read was in a non-refereed magazine [Price and Storn, 1997]. DE is a unique evolutionary algorithm because it is not biologically motivated.

Overview of the Chapter

Section 12.1 outlines a basic DE algorithm for optimization over continuous domains. After the original introduction of DE, researchers introduced many variations, and we discuss some of these variations in Section 12.2. After DE proved to be successful for continuous-domain problems, researchers extended it to discrete domains, and so we discuss DE for discrete-domain problems in Section 12.3. DE was originally introduced *not* as a separate EA, but as a genetic algorithm variation, and so we look at DE from that perspective in Section 12.4.

12.1 A BASIC DIFFERENTIAL EVOLUTION ALGORITHM

DE is a population-based algorithm that is designed to optimize functions in an n-dimensional continuous domain. Each individual in the population is an n-dimensional vector that represents a candidate solution to the problem. DE is based on the idea of taking the difference vector between two individuals, and adding a scaled version of the difference vector to a third individual to create a new candidate solution. This process is depicted in Figure 12.1.

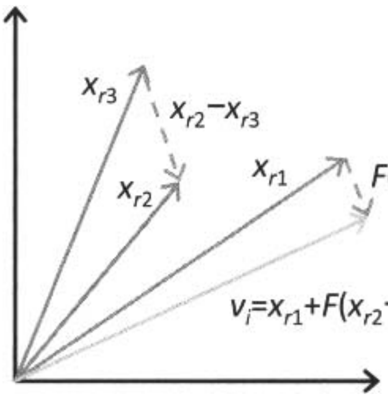

Figure 12.1 The basic idea of differential evolution, illustrated for a two-dimensional optimization problem ($n = 2$). x_{r1}, x_{r2}, and x_{r3} are candidate solutions. A scaled version of the difference between individuals x_{r2} and x_{r3} is added to x_{r1} to obtain a mutant vector v_i, which is a new candidate solution. Note that v_i is indexed with the subscript i because we generate n separate mutant vectors each generation, where n is the population size.

Figure 12.1 depicts DE in a two-dimensional search space. Two individuals, x_{r2} and x_{r3}, are randomly chosen with $r_2 \neq r_3$. A scaled version of the difference between those two individuals is added to a third randomly chosen individual, x_{r1}, where $r_1 \notin \{r_2, r_3\}$. This results in a mutant v_i that might be accepted into the population as a new candidate solution.

After the mutant vector v_i is created, it is combined (that is, crossed over) with a DE individual x_i, where $i \notin \{r_1, r_2, r_3\}$, to create a trial vector u_i. Crossover is implemented as follows:

$$u_{ij} = \begin{cases} v_{ij} & \text{if } (r_{cj} < c) \text{ or } (j = \mathcal{J}_r) \\ x_{ij} & \text{otherwise} \end{cases} \tag{12.1}$$

for $j \in [1, n]$, where n is the problem dimension and is also the dimension of u_i, v_i, and x_i; u_{ij} is the j-th component of u_i; v_{ij} is the j-th component of v_i; x_{ij} is the j-th component of individual x_i; r_{cj} is a random number taken from the uniform distribution $[0, 1]$; c is the constant crossover rate $\in [0, 1]$;[1] and $\mathcal{J}_r$ is a random integer taken from the uniform distribution $[1, n]$. We see that the trial vector u_i is a component-by-component combination of a current DE individual x_i and the mutant vector v_i. The purpose of $\mathcal{J}_r$ is to guarantee that u_i is not a clone of x_i, although this complication can be omitted for most problems (see Problem 12.3). The crossover rate c controls how likely it is that each component of u_i comes from the mutant vector v_i.

After N trial vectors u_i have been created as described above, where N is the population size, the u_i and x_i vectors are compared. The most fit vector in each (u_i, x_i) pair is kept for the next DE generation, and the least fit is discarded. The basic DE algorithm for an n-dimensional problem is summarized in Figure 12.2.

```
$F$ = stepsize parameter $\in [0.4, 0.9]$
$c$ = crossover rate $\in [0.1, 1]$
Initialize a population of candidate solutions $\{x_i\}$ for $i \in [1, N]$
While not(termination criterion)
    For each individual $x_i$, $i \in [1, N]$
        $r_1 \leftarrow$ random integer $\in [1, N] : r_1 \neq i$
        $r_2 \leftarrow$ random integer $\in [1, N] : r_2 \notin \{i, r_1\}$
        $r_3 \leftarrow$ random integer $\in [1, N] : r_3 \notin \{i, r_1, r_2\}$
        $v_i \leftarrow x_{r1} + F(x_{r2} - x_{r3})$ (mutant vector)
        $\mathcal{J}_r \leftarrow$ random integer $\in [1, n]$
        For each dimension $j \in [1, n]$
            $r_{cj} \leftarrow$ random number $\in [0, 1]$
            If $(r_{cj} < c)$ or $(j = \mathcal{J}_r)$ then
                $u_{ij} \leftarrow v_{ij}$
            else
                $u_{ij} \leftarrow x_{ij}$
            End if
        Next dimension
    Next individual
    For each population index $i \in [1, N]$
        If $f(u_i) < f(x_i)$ then $x_i \leftarrow u_i$
    Next population index
Next generation
```

Figure 12.2 A simple differential evolution (DE) algorithm for minimizing the n-dimensional function $f(x)$. This algorithm is called classic DE, or DE/rand/1/bin.

[1]Most DE literature uses the symbol Cr for the crossover rate. But two-letter symbols can be misinterpreted as separate symbols (for example, C multiplied by r), and so we use a more standard mathematical notation for the crossover rate in this chapter.