CHAPTER 3

The Continuous Genetic Algorithm

Now that you are convinced (perhaps) that the binary GA solves many optimization problems that stump traditional techniques, let's look a bit closer at the quantization limitation. What if you are attempting to solve a problem where the values of the variables are continuous and you want to know them to the full machine precision? In such a problem each variable requires many bits to represent it. If the number of variables is large, the size of the chromosome is also large. Of course, 1s and 0s are not the only way to represent a variable. One could, in principle, use any representation conceivable for encoding the variables. When the variables are naturally quantized, the binary GA fits nicely. However, when the variables are continuous, it is more logical to represent them by floating-point numbers. In addition, since the binary GA has its precision limited by the binary representation of variables, using floating point numbers instead easily allows representation to the machine precision. This continuous GA also has the advantage of requiring less storage than the binary GA because a single floating-point number represents the variable instead of N_{bits} integers. The continuous GA is inherently faster than the binary GA, because the chromosomes do not have to be decoded prior to the evaluation of the cost function.

The purpose of this chapter is to introduce the continuous GA. Most sources call this version of the GA a real-valued GA. We use the term continuous rather than real-valued to avoid confusion between real and complex numbers. The development here closely parallels the last chapter. We primarily dwell upon the differences in the two algorithms. The continuous example introduced in Chapter 1 is our primary example problem. This allows the reader to compare the continuous GA performance with the more traditional optimization algorithms introduced in Chapter 1.

Practical Genetic Algorithms, Second Edition, by Randy L. Haupt and Sue Ellen Haupt.
ISBN 0-471-45565-2

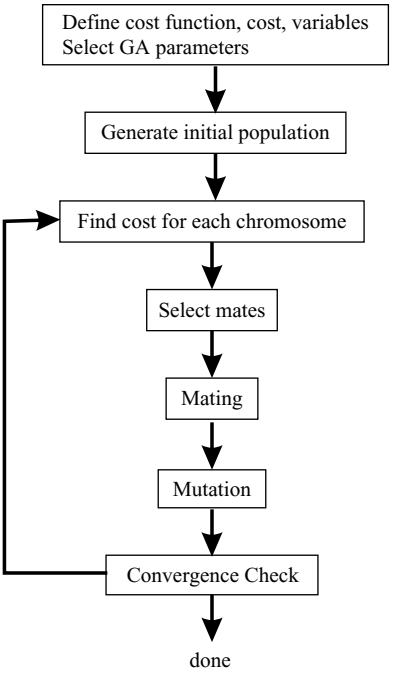

Figure 3.1 Flowchart of a continuous GA.

3.1 COMPONENTS OF A CONTINUOUS GENETIC ALGORITHM

The flowchart in Figure 3.1 provides a "big picture" overview of a continuous GA. Each block is discussed in detail in this chapter. This GA is very similar to the binary GA presented in the last chapter. The primary difference is the fact that variables are no longer represented by bits of zeros and ones, but instead by floating-point numbers over whatever range is deemed appropriate. However, this simple fact adds some nuances to the application of the technique that must be carefully considered. In particular, we will present different crossover and mutation operators.

3.1.1 The Example Variables and Cost Function

As we saw in the last chapter, the goal is to solve some optimization problem where we search for an optimal (minimum) solution in terms of the variables of the problem. Therefore we begin the process of fitting it to a GA by defining a chromosome as an array of variable values to be optimized. If the chromosome has N_{var} variables (an N-dimensional optimization problem) given by $p_1, p_2, \ldots, p_{N_{var}}$ then the chromosome is written as an array with $1 \times N_{var}$ elements so that

$$chromosome = [\, p_1, p_2, p_3, \ldots, p_{N_{var}} \,] \tag{3.1}$$

In this case, the variable values are represented as floating-point numbers. Each chromosome has a cost found by evaluating the cost function f at the variables $p_1, p_2, \ldots, p_{N_{var}}$.

$$cost = f(chromosome) = f(p_1, p_2, \ldots, p_{N_{var}}) \tag{3.2}$$

Equations (3.1) and (3.2) along with applicable constraints constitute the problem to be solved.

Our primary example in this chapter is the continuous function introduced in Chapter 1. Consider the cost function

$$\begin{gathered} cost = f(x,y) = x\sin(4x) + 1.1y\sin(2y) \\ \text{Subject to the constraints:} \quad 0 \le x \le 10 \quad \text{and} \quad 0 \le y \le 10 \end{gathered} \tag{3.3}$$

Since f is a function of x and y only, the clear choice for the variables is

$$chromosome = [x, y] \tag{3.4}$$

with $N_{var} = 2$. A contour map of the cost function appears as Figure 1.4. This cost function is considerably more complex than the cost function in Chapter 2. We see that peaks and valleys dot the landscape of the cost function contour plot. The plethora of local minima overwhelms traditional minimum-seeking methods. Our goal is to find the global minimum value of $f(x, y)$.

3.1.2 Variable Encoding, Precision, and Bounds

Here is where we begin to see the differences from the prior chapter. We no longer need to consider how many bits are necessary to accurately represent a value. Instead, x and y have continuous values that fall between the bounds listed in equation (3.3). Although the values are continuous, a digital computer represents numbers by a finite number of bits. When we refer to the continuous GA, we mean the computer uses its internal precision and roundoff to define the precision of the value. Now the algorithm is limited in precision to the roundoff error of the computer.

Since the GA is a search technique, it must be limited to exploring a reasonable region of variable space. Sometimes this is done by imposing a constraint on the problem such as equation (3.3). If one does not know the initial search region, there must be enough diversity in the initial population to explore a reasonably sized variable space before focusing on the most promising regions.

3.1.3 Initial Population

To begin the GA, we define an initial population of N_{pop} chromosomes. A matrix represents the population with each row in the matrix being a $1 \times N_{var}$ array (chromosome) of continuous values. Given an initial population of N_{pop} chromosomes, the full matrix of $N_{pop} \times N_{var}$ random values is generated by

pop = rand(N_{pop}, N_{var})

All variables are normalized to have values between 0 and 1, the range of a uniform random number generator. The values of a variable are "unnormalized" in the cost function. If the range of values is between p_{lo} and p_{hi}, then the unnormalized values are given by

$$p = (p_{hi} - p_{lo})p_{norm} + p_{lo} \tag{3.5}$$

where

p_{lo} = highest number in the variable range
p_{hi} = lowest number in the variable range
p_{norm} = normalized value of variable

In our example, the unnormalized values are just $10p_{norm}$.

This society of chromosomes is not a democracy: the individual chromosomes are not all created equal. Each one's worth is assessed by the cost function. So at this point, the chromosomes are passed to the cost function for evaluation.

We begin solving (3.3) by filling a $N_{pop} \times N_{var}$ matrix with uniform random numbers between 0 and 10. Figure 3.2 shows the initial random population for the $N_{pop} = 8$ chromosomes. Population values are listed in Table 3.1. We see widely scattered population members that well sample the values of the cost function. None of the initial guesses are particularly close to the global minimum.

3.1.4 Natural Selection

Now is the time to decide which chromosomes in the initial population are fit enough to survive and possibly reproduce offspring in the next generation. As done for the binary version of the algorithm, the N_{pop} costs and associated chromosomes are ranked from lowest cost to highest cost. The rest die off. This process of natural selection must occur at each iteration of the algorithm to allow the population of chromosomes to evolve over the generations to the most fit members as defined by the cost function. Not all of the survivors are deemed fit enough to mate. Of the N_{pop} chromosomes in a given generation, only the top N_{keep} are kept for mating and the rest are discarded to make room for the new offspring.

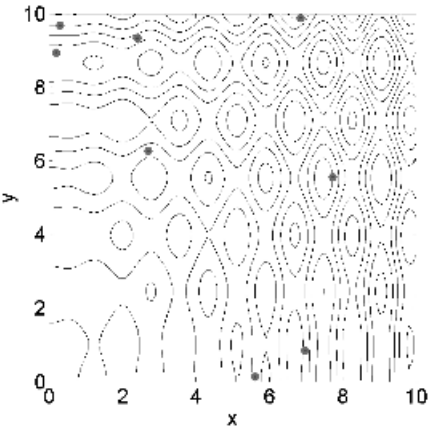

Figure 3.2 Contour plot of the cost function with the initial population ($N_{pop} = 8$) indicated by large dots.

TABLE 3.1 Example Initial Population of 8 Random Chromosomes and Their Corresponding Cost

x	*y*	Cost
6.9745	0.8342	3.4766
0.30359	9.6828	5.5408
2.402	9.3359	–2.2528
0.18758	8.9371	–8.0108
2.6974	6.2647	–2.8957
5.613	0.1289	–2.4601
7.7246	5.5655	–9.8884
6.8537	9.8784	13.752

TABLE 3.2 Surviving Chromosomes after a 50% Selection Rate

Number	*x*	*y*	Cost
1	7.7246	5.5655	–9.8884
2	0.1876	8.9371	–8.0108
3	2.6974	6.2647	–2.8957
4	5.6130	0.12885	–2.4601

In our example the mean of the cost function for the population of 8 was –0.3423 and the best cost was –9.8884. After discarding the bottom half the mean of the population is –5.8138. The natural selection results represented to five significant digits are shown in Table 3.2.

TABLE 3.3 Pairing and Mating Process of Single-Point Crossover Chromosome Family Binary String Cost

2	ma(1)	0.18758	8.9371
3	pa(1)	2.6974	6.2647
5	$offspring_1$	0.2558	6.2647
6	$offspring_2$	2.6292	8.9371
3	ma(2)	2.6974	6.2647
1	pa(2)	7.7246	5.5655
7	$offspring_3$	6.6676	5.5655
8	$offspring_4$	3.7544	6.2647

3.1.5 Pairing

The $N_{keep} = 4$ most-fit chromosomes form the mating pool. Two mothers and fathers pair in some random fashion. Each pair produces two offspring that contain traits from each parent. In addition the parents survive to be part of the next generation. The more similar the two parents, the more likely are the offspring to carry the traits of the parents. We presented some basic approaches to finding two mates in Chapter 2 and refer the reader back to that presentation rather than repeating it.

The example presented here uses rank weighting with the probabilities shown in Table 2.5. A random number generator produced the following two pairs of random numbers: (0.6710, 0.8124) and (0.7930, 0.3039). Using these random pairs and Table 2.5, the following chromosomes were randomly selected to mate:

```
ma = [2 3]
pa = [3 1]
```

Thus $chromosome_2$ mates with $chromosome_3$, and so forth. The `ma` and `pa` vectors contain the numbers corresponding to the chromosomes selected for mating. Table 3.3 summarizes the results.

3.1.6 Mating

As for the binary algorithm, two parents are chosen, and the offspring are some combination of these parents. Many different approaches have been tried for crossing over in continuous GAs. Adewuya (1996) reviews some of the methods. Several interesting methods are demonstrated by Michalewicz (1994).

The simplest methods choose one or more points in the chromosome to mark as the crossover points. Then the variables between these points are merely swapped between the two parents. For example purposes, consider the two parents to be

$$\begin{aligned} parent_1 &= [\, p_{m1}, p_{m2}, p_{m3}, p_{m4}, p_{m5}, p_{m6}, \ldots, p_{mN_{var}}] \\ parent_2 &= [\, p_{d1}, p_{d2}, p_{d3}, p_{d4}, p_{d5}, p_{d6}, \ldots, p_{dN_{var}}] \end{aligned} \tag{3.6}$$

Crossover points are randomly selected, and then the variables in between are exchanged:

$$\begin{aligned} offspring_1 &= [\, p_{m1}, p_{m2}, \uparrow p_{d3}, p_{d4}, \uparrow p_{m5}, p_{m6}, \ldots, p_{mN_{var}}] \\ offspring_2 &= [\, p_{d1}, p_{d2}, \uparrow p_{m3}, p_{m4}, \uparrow p_{d5}, p_{d6}, \ldots, p_{dN_{var}}] \end{aligned} \tag{3.7}$$

The extreme case is selecting N_{var} points and randomly choosing which of the two parents will contribute its variable at each position. Thus one goes down the line of the chromosomes and, at each variable, randomly chooses whether or not to swap information between the two parents. This method is called uniform crossover:

$$\begin{aligned} offspring_1 &= [\, p_{m1}, p_{d2}, p_{d3}, p_{d4}, p_{d5}, p_{m6}, \ldots, p_{dN_{var}}] \\ offspring_2 &= [\, p_{d1}, p_{m2}, p_{m3}, p_{m4}, p_{m5}, p_{d6}, \ldots, p_{mN_{var}}] \end{aligned} \tag{3.8}$$

The problem with these point crossover methods is that no new information is introduced: each continuous value that was randomly initiated in the initial population is propagated to the next generation, only in different combinations. Although this strategy worked fine for binary representations, there is now a continuum of values, and in this continuum we are merely interchanging two data points. These approaches totally rely on mutation to introduce new genetic material.

The blending methods remedy this problem by finding ways to combine variable values from the two parents into new variable values in the offspring. A single offspring variable value, p_{new}, comes from a combination of the two corresponding offspring variable values (Radcliff, 1991)

$$p_{new} = \beta p_{mn} + (1 - \beta) p_{dn} \tag{3.9}$$

where

β = random number on the interval [0, 1]
p_{mn} = nth variable in the mother chromosome
p_{dn} = nth variable in the father chromosome

The same variable of the second offspring is merely the complement of the first (i.e., replacing β by $1 - \beta$). If $\beta = 1$, then p_{mn} propagates in its entirety and p_{dn} dies. In contrast, if $\beta = 0$, then p_{dn} propagates in its entirety and p_{mn} dies. When $\beta = 0.5$ (Davis, 1991), the result is an average of the variables of the two parents. This method is demonstrated to work well on several interesting problems by Michalewicz (1994). Choosing which variables to blend is the next issue. Sometimes, this linear combination process is done for all variables to the right or to the left of some crossover point. Any number of points can be chosen to blend, up to N_{var} values where all variables are linear combinations of those of the two parents. The variables can be blended by using the same β for each variable or by choosing different β's for each variable. These blending methods effectively combine the information from the two parents and choose values of the variables between the values bracketed by the parents; however, they do not allow introduction of values beyond the extremes already represented in the population. To do this requires an extrapolating method. The simplest of these methods is linear crossover (Wright, 1991). In this case three offspring are generated from the two parents by

$$\begin{aligned} p_{new1} &= 0.5p_{mn} + 0.5p_{dn} \\ p_{new2} &= 1.5p_{mn} - 0.5p_{dn} \\ p_{new3} &= -0.5p_{mn} + 1.5p_{dn} \end{aligned} \tag{3.10}$$

Any variable outside the bounds is discarded in favor of the other two. Then the best two offspring are chosen to propagate. Of course, the factor 0.5 is not the only one that can be used in such a method. Heuristic crossover (Michalewicz, 1991) is a variation where some random number, β, is chosen on the interval [0, 1] and the variables of the offspring are defined by

$$p_{new} = \beta(p_{mn} - p_{dn}) + p_{mn} \tag{3.11}$$

Variations on this theme include choosing any number of variables to modify and generating different β for each variable. This method also allows generation of offspring outside of the values of the two parent variables. Sometimes values are generated outside of the allowed range. If this happens, the offspring is discarded and the algorithm tries another β. The blend crossover (BLX-α) method (Eshelman and Shaffer, 1993) begins by choosing some parameter α that determines the distance outside the bounds of the two parent variables that the offspring variable may lie. This method allows new values outside of the range of the parents without letting the algorithm stray too far. Many codes combine the various methods to use the strengths of each. New methods, such as quadratic crossover (Adewuya, 1996), do a numerical fit to the fitness function. Three parents are necessary to perform a quadratic fit.

The algorithm used in this book is a combination of an extrapolation method with a crossover method. We wanted to find a way to closely

mimic the advantages of the binary GA mating scheme. It begins by randomly selecting a variable in the first pair of parents to be the crossover point

$$\alpha = \textbf{roundup}\{\textbf{random} * N_{var}\} \tag{3.12}$$

We'll let

$$\begin{aligned} parent_1 &= [p_{m1} p_{m2} \ldots p_{m\alpha} \ldots p_{mN_{var}}] \\ parent_2 &= [p_{d1} p_{d2} \ldots p_{d\alpha} \ldots p_{dN_{var}}] \end{aligned} \tag{3.13}$$

where the *m* and *d* subscripts discriminate between the *mom* and the *dad* parent. Then the selected variables are combined to form new variables that will appear in the children:

$$\begin{aligned} p_{new1} &= p_{m\alpha} - \beta[p_{m\alpha} - p_{d\alpha}] \\ p_{new2} &= p_{d\alpha} + \beta[p_{m\alpha} - p_{d\alpha}] \end{aligned} \tag{3.14}$$

where β is also a random value between 0 and 1. The final step is to complete the crossover with the rest of the chromosome as before:

$$\begin{aligned} offspring_1 &= [p_{m1} p_{m2} \ldots p_{new1} \ldots p_{dN_{var}}] \\ offspring_2 &= [p_{d1} p_{d2} \ldots p_{new2} \ldots p_{mN_{var}}] \end{aligned} \tag{3.15}$$

If the first variable of the chromosomes is selected, then only the variables to the right of the selected variable are swapped. If the last variable of the chromosomes is selected, then only the variables to the left of the selected variable are swapped. This method does not allow offspring variables outside the bounds set by the parent unless $\beta > 1$.

For our example problem, the first set of parents are given by

$$\begin{aligned} chromosome_2 &= [0.1876, 8.9371] \\ chromosome_3 &= [2.6974, 6.2647] \end{aligned}$$

A random number generator selects p_1 as the location of the crossover. The random number selected for β is $\beta = 0.0272$. The new offspring are given by

$$\begin{aligned} offspring_1 &= [0.18758 - 0.0272 \times 0.18758 + 0.0272 \times 2.6974, 6.2647] \\ &= [0.2558, 6.2647] \\ offspring_2 &= [2.6974 + 0.0272 \times 0.18758 - 0.0272 \times 2.6974, 8.9371] \\ &= [2.6292, 8.9371] \end{aligned}$$

Continuing this process once more with a $\beta = 0.7898$. The new offspring are given by

$$\begin{aligned} offspring_3 &= [2.6974 - 0.7898 \times 2.6974 + 0.7898 \times 7.7246, 6.2647] \\ &= [6.6676, 5.5655] \end{aligned}$$

$$\begin{aligned} offspring_4 &= [7.7246 + 0.7898 \times 2.6974 - 0.7898 \times 7.7246, 8.9371] \\ &= [3.7544, 6.2647] \end{aligned}$$

3.1.7 Mutations

Here, as in the last chapter, we can sometimes find our method working too well. If care is not taken, the GA can converge too quickly into one region of the cost surface. If this area is in the region of the global minimum, that is good. However, some functions, such as the one we are modeling, have many local minima. If we do nothing to solve this tendency to converge quickly, we could end up in a local rather than a global minimum. To avoid this problem of overly fast convergence, we force the routine to explore other areas of the cost surface by randomly introducing changes, or mutations, in some of the variables. For the binary GA, this amounted to just changing a bit from a 0 to a 1, and vice versa. The basic method of mutation is not much more complicated for the continuous GA. For more complicated methods, see Michalewicz (1994).

As with the binary GA, we chose a mutation rate of 20%. Multiplying the mutation rate by the total number of variables that can be mutated in the population gives $0.20 \times 7 \times 2 \simeq 3$ mutations. Next random numbers are chosen to select the row and columns of the variables to be mutated. A mutated variable is replaced by a new random variable. The following pairs were randomly selected:

$$\text{mrow} = [4 \quad 4 \quad 7]$$
$$\text{mcol} = [1 \quad 2 \quad 1]$$

The first random pair is (4, 1). Thus the value in row 4 and column 1 of the population matrix is replaced with a uniform random number between one and ten:

$$5.6130 \Rightarrow 9.8190$$

Mutations occur two more times. The first two columns in Table 3.4 show the population after mating. The next two columns display the population after mutation. Associated costs after the mutations appear in the last column. The mutated values in Table 3.4 appear in italics. Note that the first chromosome

TABLE 3.4 Mutating the Population

Population after Mating		Population after Mutations		
x	*y*	*x*	*y*	cost
7.7246	5.5655	7.7246	5.5655	–9.8884
0.18758	8.9371	0.18758	8.9371	–8.0108
2.6974	6.2647	2.6974	6.2647	–2.8957
5.613	0.12885	*9.819*	*7.1315*	17.601
0.2558	6.2647	0.2558	6.2647	–0.03688
2.6292	8.9371	2.6292	8.9371	–10.472
6.6676	5.5655	*9.1602*	5.5655	–14.05
3.7544	6.2647	3.7544	6.2647	2.1359

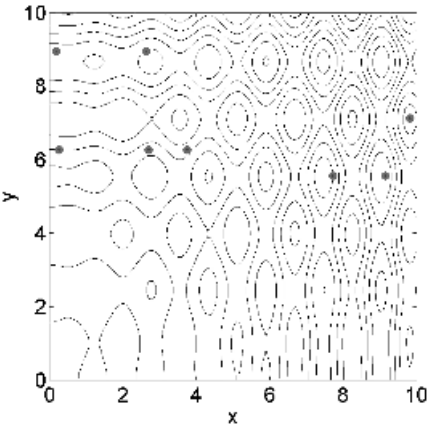

Figure 3.3 Contour plot of the cost function with the population after the first generation.

is not mutated due to elitism. The mean for this population is –3.202. The third offspring (row 7) has the best cost due to the crossover and mutation. If the *x*-value were not mutated, then the chromosome would have a cost of 0.6 and would have been eliminated in the natural selection process. Figure 3.3 shows the distribution of chromosomes after the first generation.

Most users of the continuous GA add a normally distributed random number to the variable selected for mutation

$$p_n' = p_n + \sigma \mathrm{N}_n(0,1) \tag{3.6}$$

where

σ = standard deviation of the normal distribution
$\mathrm{N}_n(0, 1)$ = standard normal distribution (mean = 0 and variance = 1)

We do not use this technique because a good value for σ must be chosen, the addition of the random number can cause the variable to exceed its bounds, and it takes more computer time.

3.1.8 The Next Generation

The process described is iterated until an acceptable solution is found. For our example, the starting population for the next generation is shown in Table 3.5 after ranking. The bottom four chromosomes are discarded and replaced by offspring from the top four parents. Another three random variables are selected for mutation from the bottom 7 chromosomes. The population at the end of generation 2 is shown in Table 3.6 and Figure 3.4. Table 3.7 is the ranked population at the beginning of generation 3. After mating, mutation, and ranking, the final population after three generations is shown in Table 3.8 and Figure 3.5.

TABLE 3.5 New Ranked Population at the Start of the Second Generation

x	*y*	Cost
9.1602	5.5655	–14.05
2.6292	8.9371	–10.472
7.7246	5.5655	–9.8884
0.18758	8.9371	–8.0108
2.6974	6.2647	–2.8957
0.2558	6.2647	–0.03688
3.7544	6.2647	2.1359
9.819	7.1315	17.601

TABLE 3.6 Population after Crossover and Mutation in the Second Generation

x	*y*	Cost
9.1602	5.5655	–14.05
2.6292	8.9371	–10.472
7.7246	6.4764	–1.1376
0.18758	8.9371	–8.0108
2.6292	5.8134	–7.496
9.1602	8.6892	–17.494
7.7246	8.6806	–13.339
4.4042	7.969	–6.1528

TABLE 3.7 New Ranked Population at the Start of the Third Generation

x	*y*	Cost
9.1602	8.6892	–17.494
9.1602	5.5655	–14.05
7.7246	8.6806	–13.339
2.6292	8.9371	–10.472
0.18758	8.9371	–8.0108
2.6292	5.8134	–7.496
4.4042	7.969	–6.1528
7.7246	6.4764	–1.137

TABLE 3.8 Ranking of Generation 3 from Least to Most Cost

x	*y*	Cost
9.0215	8.6806	–18.53
9.1602	8.6892	–17.494
9.1602	8.323	–15.366
9.1602	5.5655	–14.05
9.1602	8.1917	–13.618
2.6292	8.9371	–10.472
7.7246	1.8372	–4.849
7.8633	3.995	4.6471

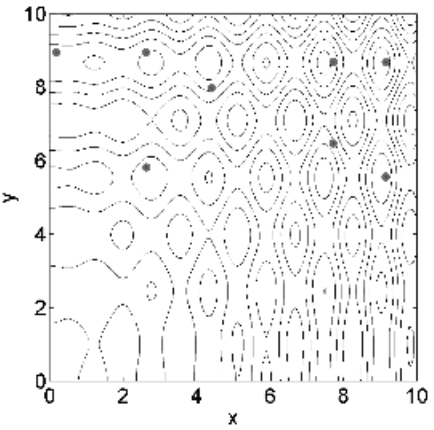

Figure 3.4 Contour plot of the cost function with the population after the second generation.

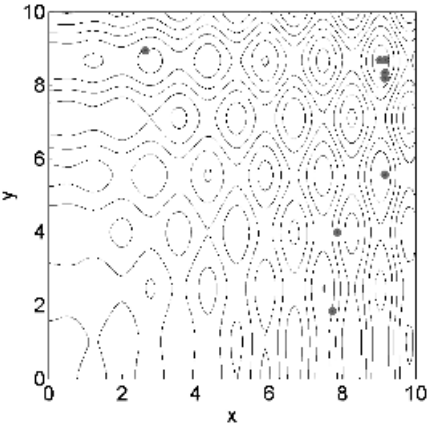

Figure 3.5 Contour plot of the cost function with the population after the third and final generation.

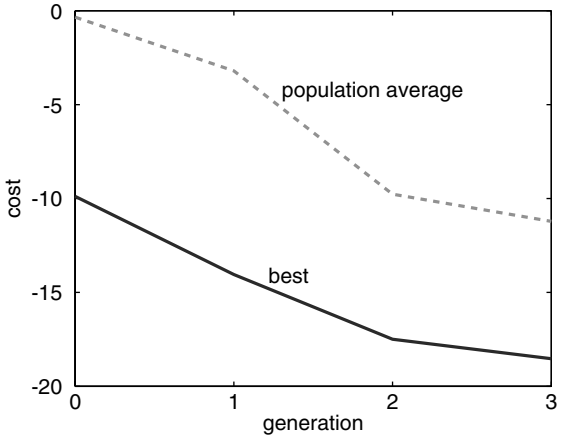

Figure 3.6 Plot of the minimum and mean costs as a function of generation. The algorithm converged in three generations.

3.1.9 Convergence

This run of the algorithm found the minimum cost (–18.53) in three generations. Members of the population are shown as large dots on the cost surface contour plot in Figures 3.2 to 3.5. By the end of the second generation, chromosomes are in the basins of the four lowest minima on the cost surface. The global minimum of –18.5 is found in generation 3. All but two of the population members are in the valley of the global minimum in the final generation. Figure 3.6 is a plot of the mean and minimum cost for each generation. The GA was able to find the global minimum, unlike the Nelder-Mead and BFGS algorithms presented in Chapter 1.

3.2 A PARTING LOOK

The binary GA could have been used in this example as well as a continuous GA. Since the problem used continuous variables, it seemed more natural to use the continuous GA. The next chapter presents some practical optimization problems for both the binary and continuous GAs. Selecting the various GA parameters, such as mutation rate and type of crossover, is still more of an art than a science and will be discussed in Chapter 5.

BIBLIOGRAPHY

Adewuya, A. A. 1996. New methods in genetic search with real-valued chromosomes. Master's thesis. Massachusetts Institute of Technology, Cambridge.

Davis, L. 1991. Hybridization and numerical representation. In L. Davis (ed.), *The Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold, pp. 61–71.

Eshelman, L. J., and D. J. Shaffer. 1993. Real-coded genetic algorithms and interval-schemata. In D. L. Whitley (ed.), *Foundations of Genetic Algorithms* 2. San Mateo, CA: Morgan Kaufman, pp. 187–202.

Michalewicz, Z. 1994. *Genetic Algorithms + Data Structures = Evolution Programs*, 2nd ed. New York: Springer-Verlag.

Radcliff, N. J. 1991. Forma analysis and random respectful recombination. In *Proc. 4th Int. Conf. on Genetic Algorithms*, San Mateo, CA: Morgan Kauffman.

Wright, A. 1991. Genetic algorithms for real parameter optimization. In G. J. E. Rawlins (ed.), *Foundations of Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann, pp. 205–218.

EXERCISES

1. Write a continuous GA that uses the following crossover:

- **a.** (3.7)
- **b.** (3.8)
- **c.** (3.9)
- **d.** (3.10)
- **e.** (3.14)

2. Write a continuous GA that uses:

- **a.** Pairing parents from top to bottom
- **b.** Random pairing
- **c.** Pairing based on cost
- **d.** Roulette wheel rank weighting
- **e.** Tournament selection

3. Find the minimum of _____ (from Appendix I) using your continuous GA.
4. Experiment with different population sizes and mutation rates. Which combination seems to work best for you? Explain.
5. Compare your GA with one of the following local optimizers:
   a. Nelder-Mead downhill simplex
   b. BFGS
   c. DFP
   d. Steepest descent
   e. Random search
6. Since the GA has many random components, it is important to average the results of several runs. Write a program that will average the results of several GA runs. Now, do another one of the exercises and compare results.
7. Plot the convergence of the GA. Which GA parameters have the most effect on convergence?
8. Compare the performance of the binary and continuous GAs. Which do you prefer and why? Does the type of problem make a difference?