CHAPTER 2

The Binary Genetic Algorithm

2.1 GENETIC ALGORITHMS: NATURAL SELECTION ON A COMPUTER

If the previous chapter whet your appetite for something better than the traditional optimization methods, this and the next chapter give step-by-step procedures for implementing two flavors of a GA. Both algorithms follow the same menu of modeling genetic recombination and natural selection. One represents variables as an encoded binary string and works with the binary strings to minimize the cost, while the other works with the continuous variables themselves to minimize the cost. Since GAs originated with a binary representation of the variables, the binary method is presented first.

Figure 2.1 shows the analogy between biological evolution and a binary GA. Both start with an initial population of random members. Each row of binary numbers represents selected characteristics of one of the dogs in the population. Traits associated with loud barking are encoded in the binary sequence associated with these dogs. If we are trying to breed the dog with the loudest bark, then only a few of the loudest, (in this case, four loudest) barking dogs are kept for breeding. There must be some way of determining the loudest barkers—the dogs may audition while the volume of their bark is measured. Dogs with loud barks receive low costs. From this breeding population of loud barkers, two are randomly selected to create two new puppies. The puppies have a high probability of being loud barkers because both their parents have genes that make them loud barkers. The new binary sequences of the puppies contain portions of the binary sequences of both parents. These new puppies replace two discarded dogs that didn't bark loud enough. Enough puppies are generated to bring the population back to its original size. Iterating on this process leads to a dog with a very loud bark. This natural optimization process can be applied to inanimate objects as well.

Practical Genetic Algorithms, Second Edition, by Randy L. Haupt and Sue Ellen Haupt.
ISBN 0-471-45565-2 Copyright © 2004 John Wiley & Sons, Inc.

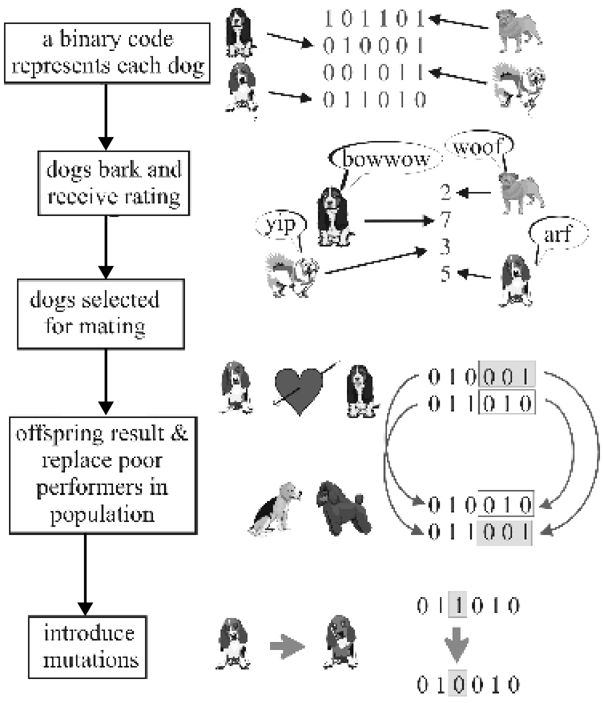


Figure 2.1 Analogy between a numerical GA and biological genetics.

2.2 COMPONENTS OF A BINARY GENETIC ALGORITHM

The GA begins, like any other optimization algorithm, by defining the optimization variables, the cost function, and the cost. It ends like other optimization algorithms too, by testing for convergence. In between, however, this algorithm is quite different. A path through the components of the GA is shown as a flowchart in Figure 2.2. Each block in this "big picture" overview is discussed in detail in this chapter.

In the previous chapter the cost function was a surface with peaks and valleys when displayed in variable space, much like a topographic map. To find a valley, an optimization algorithm searches for the minimum cost. To find a peak, an optimization algorithm searches for the maximum cost. This analogy leads to the example problem of finding the highest point in Rocky Mountain National Park. A three-dimensional plot of a portion of the park (our search space) is shown in Figure 2.3, and a crude topographical map (128×128 points) with some of the highlights is shown in Figure 2.4. Locating the top of Long's Peak (14,255 ft above sea level) is the goal. Three other interesting features in the area include Storm Peak (13,326 ft), Mount Lady Washington (13,281 ft), and Chasm Lake (11,800 ft). Since there are many peaks in the area of interest, conventional optimization techniques have difficulty finding Long's

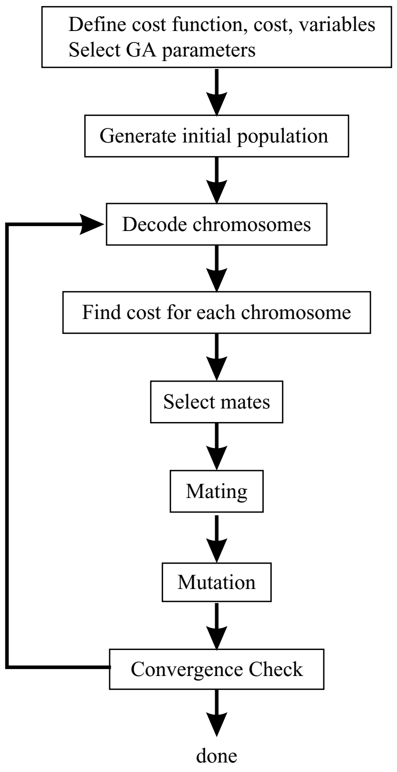

Define cost function, cost, variables
Select GA parameters
Generate initial population
Decode chromosomes
Find cost for each chromosome
Select mates
Mating
Mutation
Convergence Check
done

Figure 2.2 Flowchart of a binary GA.

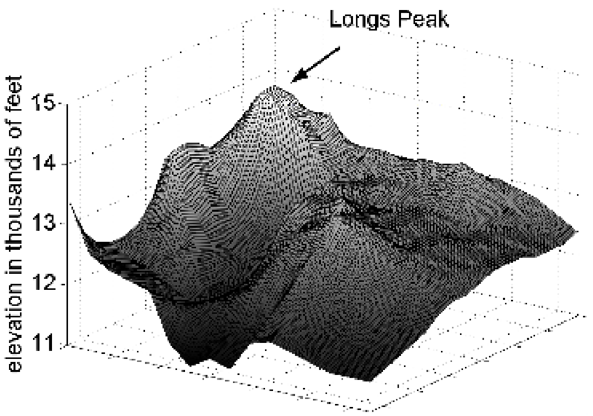


Figure 2.3 Three-dimensional view of the cost surface with a view of Long's Peak.

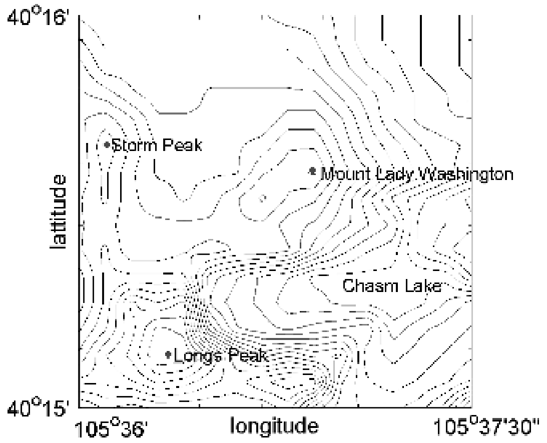


Figure 2.4 Contour plot or topographical map of the cost surface around Long's Peak.

Peak unless the starting point is in the immediate vicinity of the peak. In fact all of the methods requiring a gradient of the cost function won't work well with discrete data. The GA has no problem!

2.2.1 Selecting the Variables and the Cost Function

A cost function generates an output from a set of input variables (a chromosome). The cost function may be a mathematical function, an experiment, or a game. The object is to modify the output in some desirable fashion by finding the appropriate values for the input variables. We do this without thinking when filling a bathtub with water. The cost is the difference between the desired and actual temperatures of the water. The input variables are how much the hot and cold spigots are turned. In this case the cost function is the experimental result from sticking your hand in the water. So we see that determining an appropriate cost function and deciding which variables to use are intimately related. The term fitness is extensively used to designate the output of the objective function in the GA literature. Fitness implies a maximization problem. Although fitness has a closer association with biology than the term cost, we have adopted the term cost, since most of the optimization literature deals with minimization, hence cost. They are equivalent.

The GA begins by defining a chromosome or an array of variable values to be optimized. If the chromosome has N_{var} variables (an N_{var}-dimensional optimization problem) given by $p_1, p_2, \ldots, p_{N_{var}}$, then the chromosome is written as an N_{var} element row vector.

$$chromosome = [p_1, p_2, p_3, \ldots, p_{N_{var}}] \tag{2.1}$$

For instance, searching for the maximum elevation on a topographical map requires a cost function with input variables of longitude (x) and latitude (y)

$$chromosome = [x, y] \tag{2.2}$$

where $N_{var} = 2$. Each chromosome has a cost found by evaluating the cost function, f, at $p_1, p_2, \ldots, p_{N_{var}}$:

$$cost = f(chromosome) = f(p_1, p_2, \ldots, p_{N_{var}}) \tag{2.3}$$

Since we are trying to find the peak in Rocky Mountain National Park, the cost function is written as the negative of the elevation in order to put it into the form of a minimization algorithm:

$$f(x, y) = -elevation \text{ at } (x, y) \tag{2.4}$$

Often the cost function is quite complicated, as in maximizing the gas mileage of a car. The user must decide which variables of the problem are most important. Too many variables bog down the GA. Important variables for optimizing the gas mileage might include size of the car, size of the engine, and weight of the materials. Other variables, such as paint color and type of headlights, have little or no impact on the car gas mileage and should not be included. Sometimes the correct number and choice of variables comes from experience or trial optimization runs. Other times we have an analytical cost function. A cost function defined by $f(w, x, y, z) = 2x + 3y + z/100000 + \sqrt{w}/9876$ with all variables lying between 1 and 10 can be simplified to help the optimization algorithm. Since the w and z terms are extremely small in the region of interest, they can be discarded for most purposes. Thus the four-dimensional cost function is adequately modeled with two variables in the region of interest.

Most optimization problems require constraints or variable bounds. Allowing the weight of the car to go to zero or letting the car width be 10 meters are impractical variable values. Unconstrained variables can take any value. Constrained variables come in three brands. First, hard limits in the form of $>$, $<$, $\geq$, and $\leq$ can be imposed on the variables. When a variable exceeds a bound, then it is set equal to that bound. If x has limits of $0 \leq x \leq 10$, and the algorithm assigns $x = 11$, then x will be reassigned to the value of 10. Second, variables can be transformed into new variables that inherently include the constraints. If x has limits of $0 \leq x \leq 10$, then $x = 5\sin y + 5$ is a transformation between the constrained variable x and the unconstrained variable y. Varying y for any value is the same as varying x within its bounds. This type of transformation changes a constrained optimization problem into an unconstrained optimization problem in a smooth manner. Finally there may be a finite set of variable values from which to choose, and all values lie within the region of

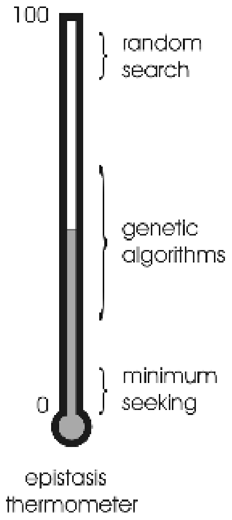


Figure 2.5 This graph of an epistasis thermometer shows that minimum seeking algorithms work best for low epistasis, while random algorithms work best for very high epistasis. GAs work best in a wide range of medium to high epistasis.

interest. Such problems come in the form of selecting parts from a limited supply.

Dependent variables present special problems for optimization algorithms because varying one variable also changes the value of the other variable. For example, size and weight of the car are dependent. Increasing the size of the car will most likely increase the weight as well (unless some other factor, such as type of material, is also changed). Independent variables, like Fourier series coefficients, do not interact with each other. If 10 coefficients are not enough to represent a function, then more can be added without having to recalculate the original 10.

In the GA literature, variable interaction is called *epistasis* (a biological term for gene interaction). When there is little to no epistasis, minimum seeking algorithms perform best. GAs shine when the epistasis is medium to high, and pure random search algorithms are champions when epistasis is very high (Figure 2.5).

2.2.2 Variable Encoding and Decoding

Since the variable values are represented in binary, there must be a way of converting continuous values into binary, and visa versa. Quantization samples a continuous range of values and categorizes the samples into nonoverlapping subranges. Then a unique discrete value is assigned to each subrange. The difference between the actual function value and the quantization level is known

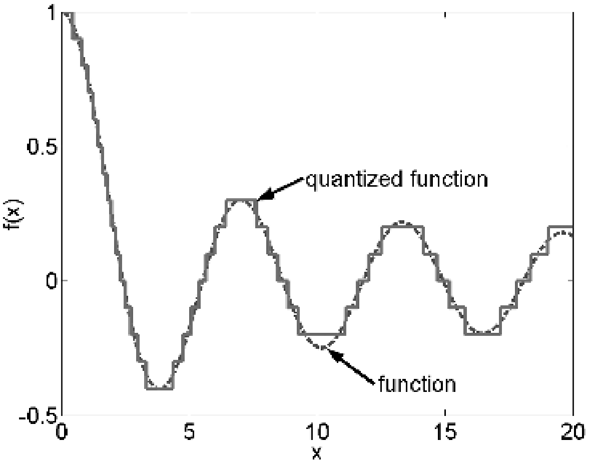
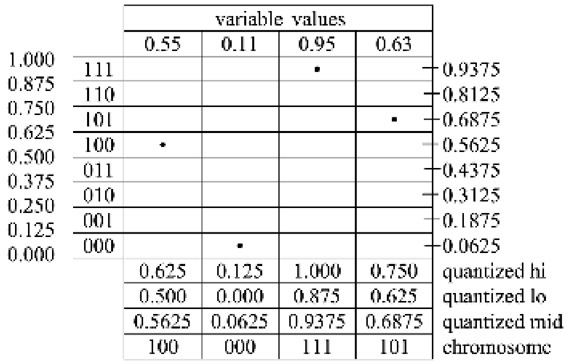


Figure 2.6 A Bessel function and a 6-bit quantized version of the same function.



		variable values				
		0.55	0.11	0.95	0.63	
1.000	111			●		0.9375
0.875	110					0.8125
0.750	101				●	0.6875
0.625	100	●				0.5625
0.500	011					0.4375
0.375	010					0.3125
0.250	001					0.1875
0.125	000		●			0.0625
0.000						
		0.625	0.125	1.000	0.750	quantized hi
		0.500	0.000	0.875	0.625	quantized lo
		0.5625	0.0625	0.9375	0.6875	quantized mid
		100	000	111	101	chromosome

Figure 2.7 Four continuous parameter values are graphed with the quantization levels shown. The corresponding gene or chromosome indicates the quantization level where the parameter value falls. Each chromosome corresponds to a low, mid, or high value in the quantization level. Normally the parameter is assigned the mid value of the quantization level.

as the quantization error. Figure 2.6 is an example of the quantization of a Bessel function ($J_0(x)$) using 4 bits. Increasing the number of bits would reduce the quantization error.

Quantization begins by sampling a function and placing the samples into equal quantization levels (Figure 2.7). Any value falling within one of the levels is set equal to the mid, high, or low value of that level. In general, setting the value to the mid value of the quantization level is best, because the largest error possible is half a level. Rounding the value to the low or high value of the level allows a maximum error equal to the quantization level. The mathematical formulas for the binary encoding and decoding of the *n*th variable, p_n, are given as follows:

For encoding,

$$p_{norm} = \frac{p_n - p_{lo}}{p_{hi} - p_{lo}} \tag{2.5}$$

$$gene[m] = \textbf{round}\left\{ p_{norm} - 2^{-m} - \sum_{p=1}^{m-1} gene[p]2^{-p} \right\} \tag{2.6}$$

For decoding,

$$p_{quant} = \sum_{m=1}^{N_{gene}} gene[m]2^{-m} + 2^{-(M+1)} \tag{2.7}$$

$$q_n = p_{quant}(p_{hi} - p_{lo}) + p_{lo} \tag{2.8}$$

In each case

p_{norm}	= normalized variable, $0 \le p_{norm} \le 1$
p_{lo}	= smallest variable value
p_{hi}	= highest variable value
$gene[m]$	= binary version of p_n
$\textbf{round}\{\cdot\}$	= round to nearest integer
p_{quant}	= quantized version of p_{norm}
q_n	= quantized version of p_n

The binary GA works with bits. The variable x has a value represented by a string of bits that is N_{gene} long. If $N_{gene} = 2$ and x has limits defined by $1 \le x \le 4$, then a gene with 2 bits has $2^{N_{gene}} = 4$ possible values. Those values are the first column of Table 2.1. The bits can represent a decimal integer, quantized values, or qualitative values as shown in columns 2 through 6 of Table 2.1. The quantized value of the gene or variable is mathematically found by multiplying the vector containing the bits by a vector containing the quantization levels:

$$q_n = gene \times Q^T \tag{2.9}$$

where

$gene = [b_1\ b_2 \ldots b_{N_{gene}}]$
N_{gene} = number bits in a gene

TABLE 2.1 Decoding a Gene

Binary Representation	Decimal Number	First Quantized x	Second Quantized x	Color	Opinion
00	0	1	1.375	Red	Excellent
01	1	2	2.125	Green	Good
10	2	3	2.875	Blue	Average
11	3	4	3.625	Yellow	Poor

b_n = binary bit = 1 or 0
Q = quantization vector = $[2^{-1}\ 2^{-2} \dots 2^{N_{gene}}]$
Q^T = transpose of Q

The first quantized representation of x includes the upper and lower bounds of x as shown in column 3 of Table 2.1. This approach has a maximum possible quantization error of 0.5. The second quantized representation of x does not include the two bounds but has a maximum error of 0.375. Increasing the number of bits decreases the quantization error. The binary representation may correspond to a nonnumerical value, such as a color or opinion that has been previously defined by the binary representation, as shown in the last two columns of Table 2.1.

The GA works with the binary encodings, but the cost function often requires continuous variables. Whenever the cost function is evaluated, the chromosome must first be decoded using (2.8). An example of a binary encoded chromosome that has N_{var} variables, each encoded with $N_{gene} = 10$ bits, is

$$chromosome = \left[\underbrace{1111001001}_{gene_1}\underbrace{0011011111}_{gene_2} \dots \underbrace{0000101001}_{gene_{N_{var}}} \right]$$

Substituting each gene in this chromosome into equation (2.8) yields an array of the quantized version of the variables. This chromosome has a total of $N_{bits} = N_{gene} \times N_{var} = 10 \times N_{var}$ bits.

As previously mentioned, the topographical map of Rocky Mountain National Park has 128 × 128 elevation points. If x and y are encoded in two genes, each with $N_{gene} = 7$ bits, then there are 2^7 possible values for x and y. These values range from $40°15' \le y \le 40°16'$ and $105°37'30'' \ge x \ge 105°36'$. The binary translations for the limits are shown in Table 2.2. The cost function translates the binary representation into a decimal value that represents the row and column in a matrix containing all the elevation values. As an example, a chromosome may have the following $N_{pop} \times N_{bits}$ binary representation:

TABLE 2.2 Binary Representations

Variable	Binary	Decimal	Value
Latitude	0000000	1	40°15′
Latitude	1111111	128	40°16′
Longitude	0000000	1	105°36′
Longitude	1111111	128	105°37′30″

$$chromosome = \left[\underbrace{1100011}_{x}\underbrace{0011001}_{y}\right]$$

This chromosome translates into matrix coordinates of [99, 25] or longitude, latitude coordinates of [105°36′50″, 40°15′29.7″]. During the optimization, the actual values of the longitude and latitude do not need to be calculated. Decoding is only necessary to interpret the results at the end.

2.2.3 The Population

The GA starts with a group of chromosomes known as the population. The population has N_{pop} chromosomes and is an $N_{pop} \times N_{bits}$ matrix filled with random ones and zeros generated using

```
pop=round(rand(Npop, Nbits));
```

where the function (N_{pop}, N_{bits}) generates a $N_{pop} \times N_{bits}$ matrix of uniform random numbers between zero and one. The function `round` rounds the numbers to the closest integer which in this case is either 0 or 1. Each row in the pop matrix is a chromosome. The chromosomes correspond to discrete values of longitude and latitude. Next the variables are passed to the cost function for evaluation. Table 2.3 shows an example of an initial population and their costs for the $N_{pop} = 8$ random chromosomes. The locations of the chromosomes are shown on the topographical map in Figure 2.8.

2.2.4 Natural Selection

Survival of the fittest translates into discarding the chromosomes with the highest cost (Figure 2.9). First, the N_{pop} costs and associated chromosomes are ranked from lowest cost to highest cost. Then, only the best are selected to continue, while the rest are deleted. The selection rate, X_{rate}, is the fraction of N_{pop} that survives for the next step of mating. The number of chromosomes that are kept each generation is

TABLE 2.3 Example Initial Population of 8 Random Chromosomes and Their Corresponding Cost

Chromosome	Cost
00101111000110	–12359
11100101100100	–11872
00110010001100	–13477
00101111001000	–12363
11001111111011	–11631
01000101111011	–12097
11101100000001	–12588
01001101110011	–11860

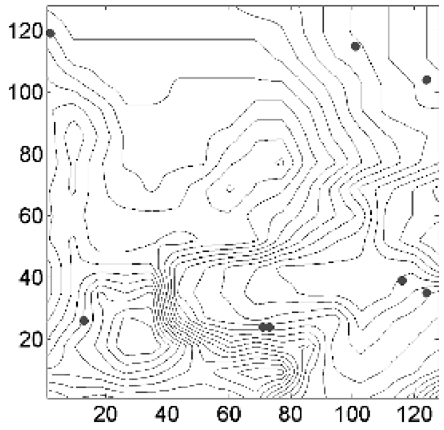


Figure 2.8 A contour map of the cost surface with the 8 initial population members indicated by large dots.

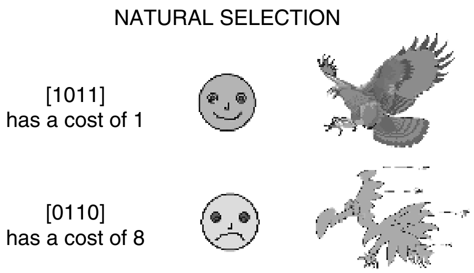


Figure 2.9 Individuals with the best traits survive. Unfit species in nature don't survive. Chromosomes with high costs in GAs are discarded.

TABLE 2.4 Surviving Chromosomes after a 50% Selection Rate

Chromosome	Cost
00110010001100	–13477
11101100000001	–12588
00101111001000	–12363
00101111000110	–12359

$$N_{keep} = X_{rate} N_{pop} \tag{2.10}$$

Natural selection occurs each generation or iteration of the algorithm. Of the N_{pop} chromosomes in a generation, only the top N_{keep} survive for mating, and the bottom $N_{pop} - N_{keep}$ are discarded to make room for the new offspring.

Deciding how many chromosomes to keep is somewhat arbitrary. Letting only a few chromosomes survive to the next generation limits the available genes in the offspring. Keeping too many chromosomes allows bad performers a chance to contribute their traits to the next generation. We often keep 50% ($X_{rate} = 0.5$) in the natural selection process.

In our example, $N_{pop} = 8$. With a 50% selection rate, $N_{keep} = 4$. The natural selection results are shown in Table 2.4. Note that the chromosomes of Table 2.4 have first been sorted by cost. Then the four with the lowest cost survive to the next generation and become potential parents.

Another approach to natural selection is called *thresholding*. In this approach all chromosomes that have a cost lower than some threshold survive. The threshold must allow some chromosomes to continue in order to have parents to produce offspring. Otherwise, a whole new population must be generated to find some chromosomes that pass the test. At first, only a few chromosomes may survive. In later generations, however, most of the chromosomes will survive unless the threshold is changed. An attractive feature of this technique is that the population does not have to be sorted.

2.2.5 Selection

Now it's time to play matchmaker. Two chromosomes are selected from the mating pool of N_{keep} chromosomes to produce two new offspring. Pairing takes place in the mating population until $N_{pop} - N_{keep}$ offspring are born to replace the discarded chromosomes. Pairing chromosomes in a GA can be as interesting and varied as pairing in an animal species. We'll look at a variety of selection methods, starting with the easiest.

1. Pairing from top to bottom. Start at the top of the list and pair the chromosomes two at a time until the top N_{keep} chromosomes are selected for mating. Thus, the algorithm pairs odd rows with even rows. The mother

has row numbers in the population matrix given by $ma = 1, 3, 5, \ldots$ and the father has the row numbers pa = 2, 4, 6, . . . This approach doesn't model nature well but is very simple to program. It's a good one for beginners to try.

2. Random pairing. This approach uses a uniform random number generator to select chromosomes. The row numbers of the parents are found using

```
ma=ceil(Nkeep*rand(1, Nkeep))
pa=ceil(Nkeep*rand(1, Nkeep))
```

where `ceil` rounds the value to the next highest integer.

3. Weighted random pairing. The probabilities assigned to the chromosomes in the mating pool are inversely proportional to their cost. A chromosome with the lowest cost has the greatest probability of mating, while the chromosome with the highest cost has the lowest probability of mating. A random number determines which chromosome is selected. This type of weighting is often referred to as roulette wheel weighting. There are two techniques: rank weighting and cost weighting.

   a. Rank weighting. This approach is problem independent and finds the probability from the rank, n, of the chromosome:

$$P_n = \frac{N_{keep} - n + 1}{\sum_{n=1}^{N_{keep}} n} = \frac{4 - n + 1}{1+2+3+4} = \frac{5-n}{10} \tag{2.11}$$

   Table 2.5 shows the results for the $N_{keep} = 4$ chromosomes of our example. The cumulative probabilities listed in column 4 are used in selecting the chromosome. A random number between zero and one is generated. Starting at the top of the list, the first chromosome with a cumulative probability that is greater than the random number is selected for the mating pool. For instance, if the random number is $r = 0.577$, then $0.4 < r \leq 0.7$, so $chromosome_2$ is selected. If a chromosome is paired with itself, there are several alternatives. First, let it go. It just means there are three of these chromosomes in the next generation.

TABLE 2.5 Rank Weighting

n	Chromosome	P_n	$\sum_{i=1}^{n} P_i$
1	00110010001100	0.4	0.4
2	11101100000001	0.3	0.7
3	00101111001000	0.2	0.9
4	00101111000110	0.1	1.0

Second, randomly pick another chromosome. The randomness in this approach is more indicative of nature. Third, pick another chromosome using the same weighting technique. Rank weighting is only slightly more difficult to program than the pairing from top to bottom. Small populations have a high probability of selecting the same chromosome. The probabilities only have to be calculated once. We tend to use rank weighting because the probabilities don't change each generation.

b. Cost weighting. The probability of selection is calculated from the cost of the chromosome rather than its rank in the population. A normalized cost is calculated for each chromosome by subtracting the lowest cost of the discarded chromosomes ($c_{N_{keep+1}}$) from the cost of all the chromosomes in the mating pool:

$$C_n = c_n - c_{N_{keep+1}} \tag{2.12}$$

Subtracting $c_{N_{keep+1}}$ ensures all the costs are negative. Table 2.6 lists the normalized costs assuming that $c_{N_{keep+1}} = -12097$. P_n is calculated from

$$P_n = \left| \frac{C_n}{\sum_m^{N_{keep}} C_m} \right| \tag{2.13}$$

This approach tends to weight the top chromosome more when there is a large spread in the cost between the top and bottom chromosome. On the other hand, it tends to weight the chromosomes evenly when all the chromosomes have approximately the same cost. The same issues apply as discussed above if a chromosome is selected to mate with itself. The probabilities must be recalculated each generation.

4. Tournament selection. Another approach that closely mimics mating competition in nature is to randomly pick a small subset of chromosomes (two or three) from the mating pool, and the chromosome with the lowest cost in this subset becomes a parent. The tournament repeats for

TABLE 2.6 Cost Weighting

n	Chromosome	$C_n = c_n - c_{N_{keep+1}}$	P_n	$\sum_{i=1}^{n} P_i$
1	00110010001100	−13477 + 12097 = −1380	0.575	0.575
2	11101100000001	−12588 + 12097 = −491	0.205	0.780
3	00101111001000	−12363 + 12097 = −266	0.111	0.891
4	00101111000110	−12359 + 12097 = −262	0.109	1.000

every parent needed. Thresholding and tournament selection make a nice pair, because the population never needs to be sorted. Tournament selection works best for larger population sizes because sorting becomes time-consuming for large populations.

Each of the parent selection schemes results in a different set of parents. As such, the composition of the next generation is different for each selection scheme. Roulette wheel and tournament selection are standard for most GAs. It is very difficult to give advice on which weighting scheme works best. In this example we follow the rank-weighting parent selection procedure.

Figure 2.10 shows the probability of selection for five selection methods. Uniform selection has a constant probability for each of the eight parents. Roulette wheel rank selection and tournament selection with two chromosomes have about the same probabilities for the eight parents. Selection pressure is the ratio of the probability that the most fit chromosome is selected as a parent to the probability that the average chromosome is selected. The selection pressure increases for roulette wheel rank squared selection and tournament selection with three chromosomes, and their probability of selection for the eight parents are nearly the same. For more information on these selection methods, see Bäck (1994) and Goldberg and Deb (1991).

2.2.6 Mating

Mating is the creation of one or more offspring from the parents selected in the pairing process. The genetic makeup of the population is limited by

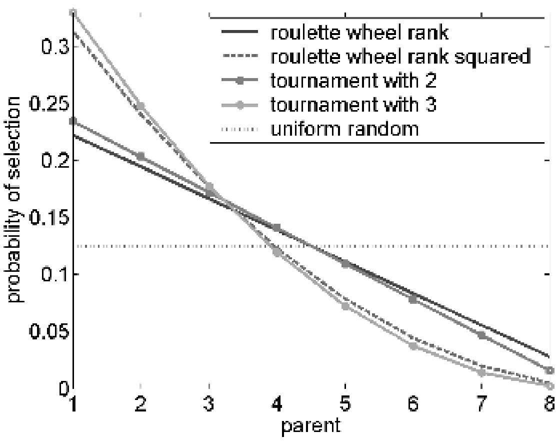


Figure 2.10 Graph of the probability of selection for 8 parents using five different methods of selection.

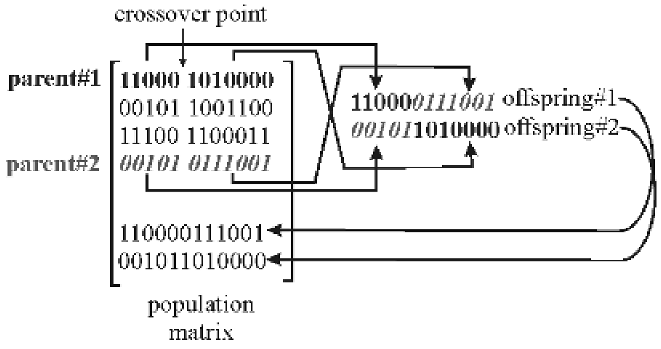


Figure 2.11 Two parents mate to produce two offspring. The offspring are placed into the population.

TABLE 2.7 Pairing and Mating Process of Single-Point Crossover

Chromosome	Family	Binary String
3	ma(1)	*00101111001000*
2	pa(1)	11101100000001
5	*offspring*$_1$	*00101*100000001
6	*offspring*$_2$	11101*111001000*
3	ma(2)	*00101111001000*
4	pa(2)	00101111000110
7	*offspring*$_3$	*0010111100*0110
8	*offspring*$_4$	0010111100*1000*

the current members of the population. The most common form of mating involves two parents that produce two offspring (see Figure 2.11). A crossover point, or kinetochore, is randomly selected between the first and last bits of the parents' chromosomes. First, *parent*$_1$ passes its binary code to the left of that crossover point to *offspring*$_1$. In a like manner, *parent*$_2$ passes its binary code to the left of the same crossover point to *offspring*$_2$. Next, the binary code to the right of the crossover point of *parent*$_1$ goes to *offspring*$_2$ and *parent*$_2$ passes its code to *offspring*$_1$. Consequently the offspring contain portions of the binary codes of both parents. The parents have produced a total of $N_{pop} - N_{keep}$ offspring, so the chromosome population is now back to N_{pop}. Table 2.7 shows the pairing and mating process for the problem at hand. The first set of parents is chromosomes 3 and 2 and has a crossover point between bits 5 and 6. The second set of parents is chromosomes 3 and 4 and has a crossover point between bits 10 and 11. This process is known as simple or single-point crossover. More complicated versions of mating are discussed in Chapter 5.

2.2.7 Mutations

Random mutations alter a certain percentage of the bits in the list of chromosomes. Mutation is the second way a GA explores a cost surface. It can introduce traits not in the original population and keeps the GA from converging too fast before sampling the entire cost surface. A single point mutation changes a 1 to a 0, and visa versa. Mutation points are randomly selected from the $N_{pop} \times N_{bits}$ total number of bits in the population matrix. Increasing the number of mutations increases the algorithm's freedom to search outside the current region of variable space. It also tends to distract the algorithm from converging on a popular solution. Mutations do not occur on the final iteration. Do we also allow mutations on the best solutions? Generally not. They are designated as *elite* solutions destined to propagate unchanged. Such elitism is very common in GAs. Why throw away a perfectly good answer?

For the Rocky Mountain National Park problem, we choose to mutate 20% of the population ($\mu = 0.20$), except for the best chromosome. Thus a random number generator creates seven pairs of random integers that correspond to the rows and columns of the mutated bits. In this case the number of mutations is given by

$$\#\,mutations = \mu \times (N_{pop} - 1) \times N_{bits} = 0.2 \times 7 \times 14 = 19.6 \simeq 20 \qquad (2.14)$$

The computer code to find the rows and columns of the mutated bits is

```
nmut=ceil((Npop-1)*Nbits μ);
mrow=ceil(rand(1, μ)*(Npop-1))+1;
mcol=ceil(rand(1, μ)*Nbits);
pop(mrow,mcol)=abs(pop(mrow,mcol)-1);
```

The following pairs were randomly selected:

```
mrow =[5  7 6  3  6 6 8 4 6  7  3 4  7 4 8  6 6  4  6 7]
mcol =[6 12 5 11 13 5 5 6 4 11 10 6 13 3 4 11 5 14 10 5]
```

The first random pair is (5, 6). Thus the bit in row 5 and column 6 of the population matrix is mutated from a 1 to a ***0***:

$$00101100000001 \Rightarrow 00101\boldsymbol{\mathit{0}}00000001$$

Mutations occur 19 more times. The mutated bits in Table 2.8 appear in italics. Note that the first chromosome is not mutated due to elitism. If you look carefully, only 18 bits are mutated in Table 2.8 instead of 20. The reason is that the row column pair (6, 5) was randomly selected three times. Thus the same bit switched from a 1 to a 0 back to a 1 and finally to a 0. Locations of the chromosomes at the end of the first generation are shown in Figure 2.12.

TABLE 2.8 Mutating the Population

Population after Mating	Population after Mutations	New Cost
00110010001100	00110010001100	–13477
11101100000001	11101100000001	–12588
00101111001000	001011110*10*000	–12415
00101111000110	00*0*01*0*1100011*1*	–13482
00101100000001	00101*0*00000001	–13171
11101111001000	111*10*1110*10*0*1*0	–12146
00101111000110	0010*0*111*00100*0	–12716
00101111001000	001*10*111001000	–12103

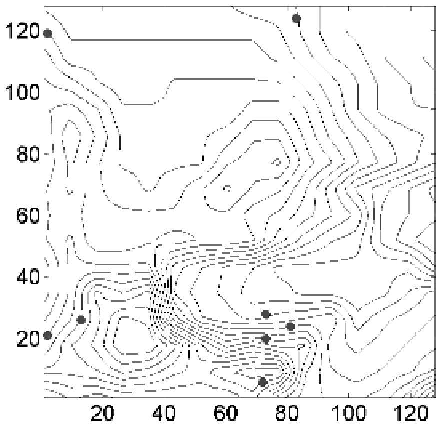


Figure 2.12 A contour map of the cost surface with the 8 members at the end of the first generation.

2.2.8 The Next Generation

After the mutations take place, the costs associated with the offspring and mutated chromosomes are calculated (third column in Table 2.8). The process described is iterated. For our example, the starting population for the next generation is shown in Table 2.9 after ranking. The bottom four chromosomes are discarded and replaced by offspring from the top four parents. Another 20 random bits are selected for mutation from the bottom 7 chromosomes. The population at the end of generation 2 is shown in Table 2.10 and Figure 2.13. Table 2.11 is the ranked population at the beginning of generation 3. After mating, mutation, and ranking, the population is shown in Table 2.12 and Figure 2.14.

TABLE 2.9 New Ranked Population at the Start of the Second Generation

Chromosome	Cost
00001011000111	–13482
00110010001100	–13477
00101000000001	–13171
00100111001000	–12716
11101100000001	–12588
00101111010000	–12415
11110111010010	–12146
00110111001000	–12103

TABLE 2.10 Population after Crossover and Mutation in the Second Generation

Chromosome	Cost
00001011000111	–13482
00110000001000	–13332
01101001000001	–12923
01100111011000	–12128
10100111000001	–12961
10100010001000	–13237
00110100001110	–13564
00100010000001	–13246

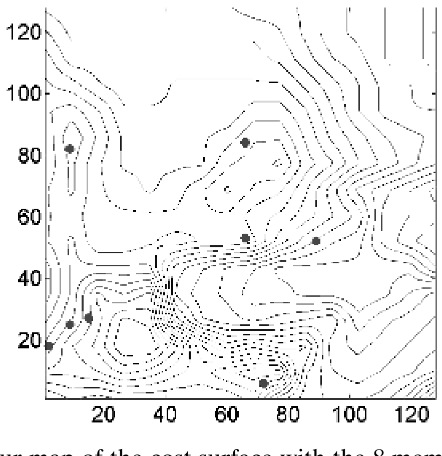


Figure 2.13 A contour map of the cost surface with the 8 members at the end of the second generation.

TABLE 2.11 New Ranked Population at the Start of the Third Generation

Chromosome	Cost
00110100001110	–13564
00001011000111	–13482
00110000001000	–13332
00100010000001	–13246
10100010001000	–13237
10100111000001	–12961
01101001000001	–12923
01100111011000	–12128

TABLE 2.12 Ranking of Generation 2 from Least to Most Cost

Chromosome	Cost
00100010100001	–14199
00110100001110	–13564
00010000001110	–13542
00100000000001	–13275
00100011010000	–12840
00001111111111	–12739
11001011000111	–12614
01111111011111	–12192

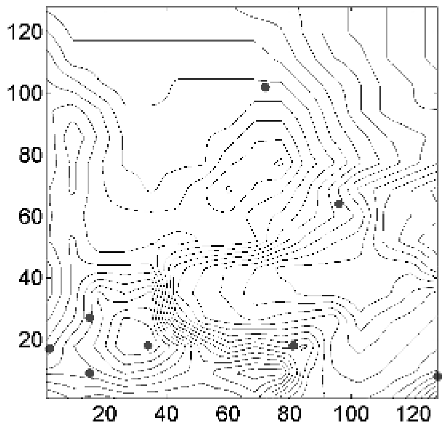


Figure 2.14 A contour map of the cost surface with the 8 members at the end of the third generation.

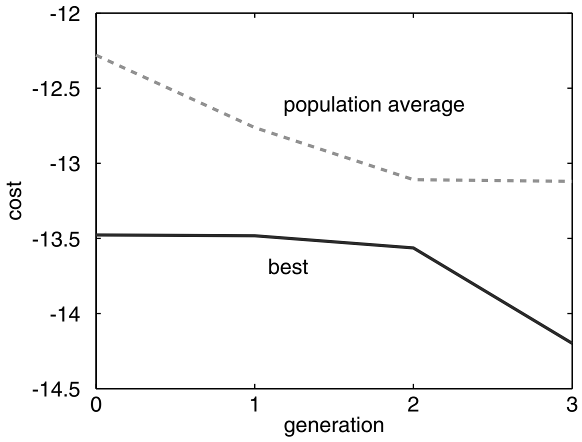


Figure 2.15 Graph of the mean cost and minimum cost for each generation.

2.2.9 Convergence

The number of generations that evolve depends on whether an acceptable solution is reached or a set number of iterations is exceeded. After a while all the chromosomes and associated costs would become the same if it were not for mutations. At this point the algorithm should be stopped.

Most GAs keep track of the population statistics in the form of population mean and minimum cost. For our example, after three generations the global minimum is found to be –14199. This minimum was found in

$$\underbrace{8}_{\text{initial population}} + \underbrace{7}_{\substack{\text{max cost evaluations}\\ \text{per generation}}} \times \underbrace{3}_{\text{generations}} = 29 \tag{2.15}$$

cost function evaluations or checking 29/(128 × 128) × 100 = 0.18% of the population. The final population is shown in Figure 2.14, where four of the members are close to Long’s Peak. Figure 2.15 shows a plot of the algorithm convergence in terms of the minimum and mean cost of each generation. Long’s Peak is actually 14,255 ft above sea level, but the quantization error (due to gridding) produced a maximum of 14,199.

2.3 A PARTING LOOK

We’ve managed to find the highest point in Rocky Mountain National Park with a GA. This may have seemed like a trivial problem—the peak could have easily been found through an exhaustive search or by looking at a topographical map. True. But try using a conventional numerical optimization routine to find the peak in these data. Such routines don’t work very well.

Many can't even be adapted to apply to this simple problem. We'll present some much more difficult problems in Chapters 4 and 6 where the utility of the GA becomes even more apparent. For now you should be comfortable with the workings of a simple GA.

```
% This is a simple GA written in MATLAB
%  costfunction.m calculates a cost for each row or
%  chromosome in pop. This function must be provided
%  by the user.

N=200;     % number of bits in a chromosome
M=8;     % number of chromosomes must be even
last=50; % number of generations
sel=0.5; % selection rate
M2=2*ceil(sel*M/2);    % number of chromosomes kept
mutrate=0.01;           % mutation rate
nmuts=mutrate*N*(M-1); % number of mutations

% creates M random chromosomes with N bits
pop=round(rand(M,N)); % initial population

for ib=1:last

   cost=costfunction(pop);   % cost function
   % ranks results and chromosomes
   [cost,ind]=sort(cost);
   pop=pop(ind(1:M2),:);
   [ib cost(1)]

   %mate
   cross=ceil((N-1)*rand(M2,1));

   % pairs chromosomes and performs crossover
   for ic=1:2:M2
    pop(ceil(M2*rand),1:cross)=pop(ic,1:cross);
    pop(ceil(M2*rand),cross+1:N)=pop(ic+1,cross+1:N);
    pop(ceil(M2*rand),1:cross)=pop(ic+1,1:cross);
    pop(ceil(M2*rand),cross+1:N)=pop(ic,cross+1:N);
   end

   %mutate
for ic=1:nmuts
   ix=ceil(M*rand);
   iy=ceil(N*rand);
   pop(ix,iy)=1-pop(ix,iy);
end %ic

end %ib
```

Figure 2.16 MATLAB code for a very simple GA.

It's very simple to program a GA. An extremely short GA in MATLAB is shown in Figure 2.16. This GA uses pairing from top to bottom when selecting mates. The cost function must be provided by the user and converts the binary strings into usable variable values.

BIBLIOGRAPHY

Angeline, P. J. 1995. Evolution revolution: An introduction to the special track on genetic and evolutionary programming. *IEEE Exp. Intell. Syst. Appl.* **10**:6–10.

Bäck, T. 1994. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In Proc. 1st IEEE Conf. on Evolutionary Computation. Piscataway, NJ: IEEE Press, pp. 57–62.

Goldberg, D. E., and K. Deb. 1991. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann, pp. 69–93.

Goldberg, D. E. 1993. Making genetic algorithms fly: A lesson from the Wright brothers. *Adv. Technol. Dev.* **2**:1–8.

Holland, J. H. 1992. Genetic algorithms. *Sci. Am.* **267**:66–72.

Janikow, C. Z., and D. St. Clair. 1995. Genetic algorithms simulating nature's methods of evolving the best design solution. *IEEE Potentials* **14**:31–35.

Malasri, S., J. R. Martin, and L. Y. Lin. 1995. Hands-on software for teaching genetic algorithms. *Comput. Educ. J.* **6**:42–47.

EXERCISES

1. Write a binary GA that uses:

- **a.** Single-point crossover
- **b.** Double-point crossover
- **c.** Uniform crossover

2. Write a binary 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 binary GA.

4. Experiment with different population sizes and mutation rates. Which combination seems to work best for you? Explain.

5. Compare your binary GA with 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 over multiple runs. Write a program that will average the results of your GA. Then do another one of the exercises and compare results.

7. Plot the convergence of the GA. Do a sensitivity analysis on parameters such as μ and N_{pop}. Which GA parameters have the most effect on convergence? A convergence plot could be: best minimum in the population versus the number of function calls or the best minimum in the population versus generation. Which method is better?