27.1 Introduction To Genetic Algorithms

Genetic algorithms (GAs) attempt to computationally mimic the processes by which natural selection operates, and apply them to solve business and research problems. Developed by John Holland in the 1960s and 1970s (Holland[1]), GAs provide a framework for studying the effects of such biologically inspired factors as mate selection, reproduction, mutation, and crossover of genetic information.

In the natural world, the constraints and stresses of a particular environment force the different species (and different individuals within species) to compete to produce the fittest offspring. In the world of GAs, the fitness of various potential solutions is compared, and the fittest potential solutions evolve to produce ever more optimal solutions.

Not surprisingly, the field of GAs has borrowed heavily from genomic terminology. Each cell in our body contains the same set of *chromosomes*, strings of DNA that function as a blueprint for making one of us. Then, each chromosome can be partitioned into *genes*, which are blocks of DNA designed to encode a particular trait such as eye color. A particular instance of the gene (e.g., brown eyes) is an *allele*. Each gene is to be found at a particular *locus* on the chromosome. Recombination, or *crossover*, occurs during reproduction, where a new chromosome is formed by combining the characteristics of both parents' chromosomes. *Mutation*, the altering of a single gene in a chromosome of the offspring, may occur, randomly and relatively rarely. The offspring's *fitness* is then evaluated, either in terms of viability (living long enough to reproduce) or in the offspring's fertility.

Now, in the field of GAs, a *chromosome* refers to one of the candidate solutions to the problem, a *gene* is a single bit or digit of the candidate solution, and an *allele* is a particular instance of the bit or digit (e.g., 0 for binary-encoded solutions or the number 7 for real-valued solutions). Recall that binary numbers have base two, so that the first “decimal” place represents “ones,” the second represents “twos,” the third represents “fours,” the fourth represents “eights,” and so forth. So the binary string 10101010 represents

$$\begin{aligned}&(1 \times 128) + (0 \times 64) + (1 \times 32) + (0 \times 16) + (1 \times 8) + (0 \times 4) + (1 \times 2) + (0 \times 1)\\ &= 170\end{aligned}$$

in decimal notation.

There are three operators used by GAs, *selection*, *crossover*, and *mutation*.

1. **Selection**. The selection operator refers to the method used for selecting which chromosomes will be reproducing. The fitness function evaluates each of the chromosomes (candidate solutions), and the fitter the chromosome, the more likely it will be selected to reproduce.
2. **Crossover**. The crossover operator performs recombination, creating two new offspring by randomly selecting a locus and exchanging subsequences to the left and right of that locus between two chromosomes chosen during selection. For example, in binary representation, two strings 11111111 and 00000000 could be crossed over at the sixth locus in each to generate the two new offspring 11111000 and 00000111.
3. **Mutation**. The mutation operator randomly changes the bits or digits at a particular locus in a chromosome, usually, however with very small probability. For example, after

crossover, the 11111000 child string could be mutated at locus two to become 10111000. Mutation introduces new information to the genetic pool, and protects against converging too quickly to a local optimum.

Most GAs function by iteratively updating a collection of potential solutions, called a *population*. Each member of the population is evaluated for fitness on each cycle. A new population then replaces the old population using the above operators, with the fittest members being chosen for reproduction or cloning. The fitness function $f(x)$ is a real-valued function operating on the chromosome (potential solution), not the gene, so that the x in $f(x)$ refers to the numeric value taken by the chromosome at the time of fitness evaluation.

27.2 Basic Framework of a Genetic Algorithm

The following introductory GA framework is adapted from Mitchell[2] in her interesting book *An Introduction to Genetic Algorithms*.

- **Step 0**. Initialization. Assume that the data are encoded in bit strings (1's and 0's). Specify a *crossover probability* or *crossover rate* p_c, and a *mutation probability* or *mutation rate* p_m. Usually, p_c is chosen to be fairly high (e.g., 0.7), and p_m is chosen to be very low (e.g., 0.001).
- **Step 1**. The population is chosen, consisting of a set of n chromosomes, each of length l.
- **Step 2**. The fitness $f(x)$ for each chromosome in the population is calculated.
- **Step 3**. Iterate through the following steps until n offspring have been generated.

  Step 3a. **Selection**. Using the values from the fitness function $f(x)$ from step 2, assign a probability of selection to each individual chromosome, with higher fitness providing a higher probability of selection. The usual term for the way these probabilities are assigned is the *roulette wheel* method. For each chromosome x_i, find the proportion of this chromosome's fitness to the total fitness summed over all the chromosomes. That is, find $f(x_i)/\sum_i f(x_i)$, and assign this proportion to be the probability of selecting that chromosome for parenthood. Each chromosome then has a proportional slice of the putative roulette wheel spun to choose the parents. Then select a pair of chromosomes to be parents, based on these probabilities. Allow the same chromosome to be potentially selected to be a parent more than once. Allowing a chromosome to pair with itself will generate three copies of that chromosome to the new generation. If the analyst is concerned about converging too quickly to a local optimum, then perhaps such pairing should not be allowed.

  Step 3b. Crossover. Select a randomly chosen locus (*crossover point*) for where to perform the crossover. Then, with probability p_c, perform crossover with the parents selected in step 3a, thereby forming two new offspring. If the crossover is not performed, clone two exact copies of the parents to be passed on to the new generation.

  Step 3c. Mutation. With probability p_m, perform mutation on each of the two offspring at each locus point. The chromosomes then take their place in the new population. If n is odd, discard one new chromosome at random.
- **Step 4**. The new population of chromosomes replaces the current population.
- **Step 5**. Check whether termination criteria have been met. For example, is the change in mean fitness from generation to generation vanishingly small? If convergence is achieved, then stop and report results; otherwise, go to step 2.

Each cycle through this algorithm is called a *generation*, with most GA applications taking from 50 to 500 generations to reach convergence. Mitchell suggests that researchers try several different runs with different random number seeds, and report the model evaluation statistics (e.g., best overall fitness) averaged over several different runs.

27.3 Simple Example of a Genetic Algorithm at Work

Let us examine a simple example of a GA at work. Suppose our task is to find the maximum value of the normal distribution with mean $\mu = 16$ and standard deviation $\sigma = 4$ (Figure 27.1). That is, we would like to find the maximum value of:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-1}{2\sigma^2}(x-\mu)^2\right) = \frac{1}{\sqrt{2\pi}(4)} \exp\left(\frac{-1}{2(4)^2}(x-16)^2\right).$$

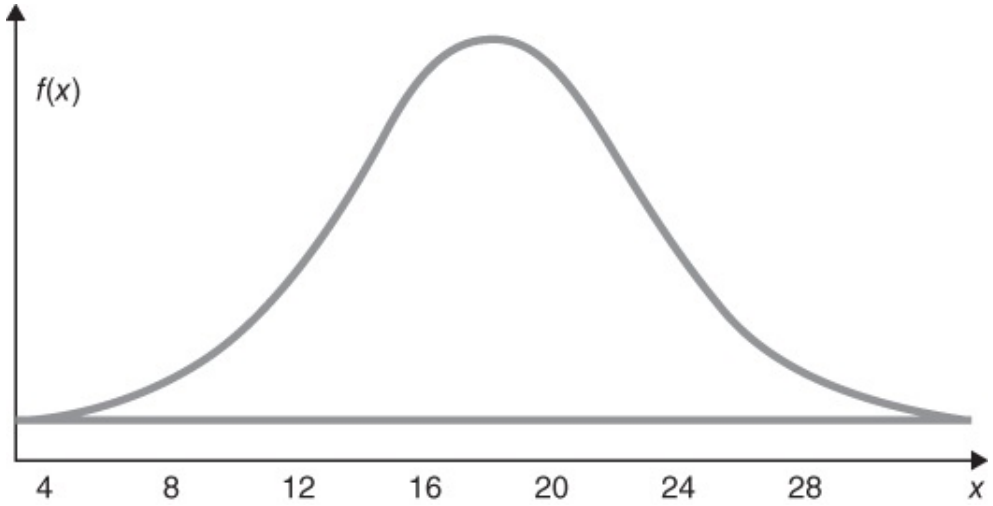

Figure 27.1 Find the maximum value of the normal(16, 4) distribution.

We allow X to take on only the values described by the first five binary digits; that is, 00000 through 11111, or 0–31 in decimal notation.

27.3.1 First Iteration

- **Step 0**. Initialization. We define the *crossover rate* to be $p_c = 0.75$ and the *mutation rate* to be $p_m = 0.002$.
- **Step 1**. Our population will be a set of four chromosomes, randomly chosen from the set 00000–11111. So, $n = 4$ and $l = 5$. These are 00100 (4), 01001 (9), 11011 (27), and 11111 (31).
- **Step 2**. The fitness $f(x)$ for each chromosome in the population is calculated.
- **Step 3**. Iterate through the following steps until n offspring have been generated.

  Step 3a. Selection. We have the sum of the fitness values equal to
  $\sum_i f(x_i) = 0.001108 + 0.021569 + 0.002273 + 0.000088 = 0.025038.$

  Then, the probability that each of our chromosomes will be selected for parenthood is found by dividing their value for $f(x)$ by the sum 0.025038. These are also shown in Table 27.1. Clearly, chromosome 01001 gets a very large slice of the roulette wheel! The random selection process gets underway. Suppose that chromosomes 01001 and 11011 are selected to be the first pair of parents, because these are the two chromosomes with the highest fitness.

Table 27.1 Fitness and probability of selection for each chromosome

Chromosome	Decimal Value	Fitness	Selection Probability
00100	4	0.001108	0.04425
01001	9	0.021569	0.86145
11011	27	0.002273	0.09078
11111	31	0.000088	0.00351

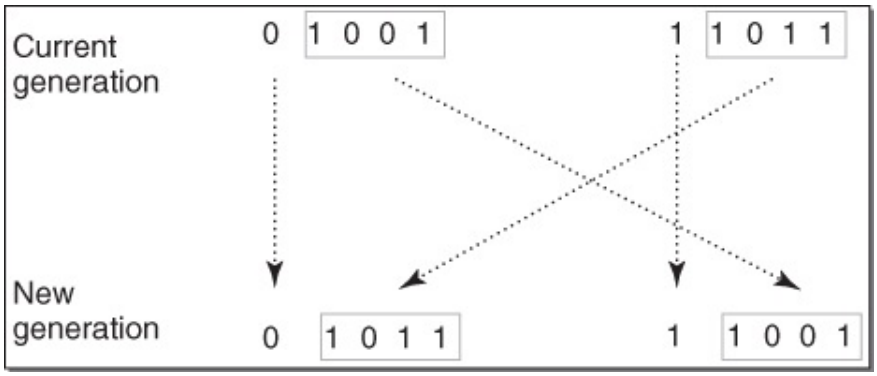

Figure 27.2 Performing crossover at locus two on the first two parents.

Step 3b. Crossover. The locus is randomly chosen to be the second position. Suppose the large crossover rate of p_c, 0.75, leads to crossover between 01001 and 11011 occurring at the second position. This is shown in Figure 27.2. Note that the strings are partitioned between the first and the second bits. Each child chromosome receives one segment from each of the parents. The two chromosomes thus formed for the new generation are 01011 (11) and 11001 (25).

Step 3c. Mutation. Because of the low mutation rate, suppose that none of the genes for 01011 or 11001 are mutated. We now have two chromosomes in our new population. We need two more, so we cycle back to step 3a.

Step 3a. Selection. Suppose that this time chromosomes 01001 (9) and 00100 (4) are selected by the roulette wheel method.

Step 3b. Crossover. However, this time suppose that crossover does not take place. Thus, clones of these chromosomes become members of the new generation, 01001 and 00100. We now have $n = 4$ members in our new population.

- **Step 4**. The new population of chromosomes therefore replaces the current population.
- **Step 5**. And we iterate back to Step 2.

27.3.2 Second Iteration

- **Step 2**. The fitness $f(x)$ for each chromosome in the population is calculated, as shown in Table 27.2.

  Step 3a. Selection. The sum of the fitness values for the second generation is $\sum_i f(x_i) = 0.076274$, which means that the average fitness among the chromosomes in the second generation is three times that of the first generation. The selection probabilities are calculated, and shown in Table 27.2.

Table 27.2 Fitness and probability of selection for the second generation

Chromosome	Decimal Value	Fitness	Selection Probability
00100	4	0.001108	0.014527
01001	9	0.021569	0.282783
01011	11	0.045662	0.598657
11001	25	0.007935	0.104033

We ask you to continue this example in the exercises.

27.4 Modifications and Enhancements: Selection

For the selection operator, the analyst should be careful to balance fitness with diversity. If fitness is favored over variability, then a set of highly fit but suboptimal chromosomes will dominate the population, reducing the ability of the GA to find the global optimum. If diversity is favored over fitness, then model convergence will be too slow.

For example, in the first generation above, one particular gene 01001 (9) dominated the fitness measure, with over 86% of the selection probability. This is an example of *selection pressure*, and a potential example of the *crowding* phenomenon in GAs, where one particular chromosome that is much fitter than the others begins to reproduce, generating too many clones and similar copies of itself in future generations. By reducing the diversity of the population, crowding impairs the ability of the GA to continue to explore new regions of the search space.

A variety of techniques are available to handle crowding. De Jong[3] suggested that new generation chromosomes should replace the individual most similar to itself in the current generation. Goldberg and Richardson[4] posited a *fitness sharing* function, where a particular chromosome's fitness was decreased by the presence of other similar population members, where the more similarity, the greater the decrease. Thus, diversity was rewarded.

Changing the mating conditions can also be used to increase population diversity. Deb and Goldberg[5] showed that if mating can take place only between sufficiently similar chromosomes, then distinct "mating groups" will have a propensity to form. These groups displayed low within-group variation and high between-group variation. However, Eshelman[6] and Eshelman and Schaffer[7] investigated the opposite strategy by not allowing matings between chromosomes that were sufficiently alike. The result was to maintain high variability within the population as a whole.

Sigma scaling, proposed by Forrest,[8] maintains the selection pressure at a relatively constant rate, by scaling a chromosome's fitness by the standard deviation of the fitnesses. If a single chromosome dominates at the beginning of the run, then the variability in fitnesses will also be large, and scaling by the variability will reduce the dominance. Later in the run, when populations are typically more homogeneous, scaling by this smaller variability will allow the highly fit chromosomes to reproduce. The sigma-scaled fitness is as follows:

$$f_{\text{sigma-scaled}}(x) = 1 + \frac{f(x) - \mu_f}{\sigma_f}$$

where μ_f and σ_f refer to the mean fitness and standard deviation of the fitnesses for the current generation.

Boltzmann selection varies the selection pressure, depending on how far along in the run the generation is. Early on, it may be better to allow lower selection pressure, allowing the less-fit chromosomes to reproduce at rates similar to the fitter chromosomes, and thereby maintaining a wider exploration of the search space. Later on in the run, increasing the selection pressure will help the GA to converge more quickly to the optimal solution, hopefully the global optimum. In Boltzmann selection, a *temperature* parameter T is gradually reduced from high levels to low levels. A chromosome's adjusted fitness is then

found as follows:

$$f_{\text{Boltzmann}}(x) = \frac{\exp(f(x)/T)}{\text{Mean}(\exp(f(x)/T))}$$

As the temperature falls, the difference in expected fitness increases between high-fit and low-fit chromosomes.

Elitism, developed by De Jong, refers to the selection condition requiring that the GA retain a certain number of the fittest chromosomes from one generation to the next, protecting them against destruction through crossover, mutation, or inability to reproduce. Michell, Haupt, and Haupt[9] and others report that elitism greatly improves GA performance.

Rank selection ranks the chromosomes according to fitness. Ranking avoids the selection pressure exerted by the proportional fitness method, but it also ignores the absolute differences among the chromosome fitnesses. Ranking does not take variability into account, and provides a moderate adjusted fitness measure, because the difference in probability of selection between chromosomes ranked k and $k + 1$ is the same, regardless of the absolute differences in fitness.

Tournament ranking is computationally more efficient than rank selection, while preserving the moderate selection pressure of rank selection. In tournament ranking, two chromosomes are chosen at random and with replacement from the population. Let c be a constant chosen by the user to be between zero and one (e.g., 0.67). A random number r, $0 \leq r \leq 1$, is drawn. If $r < c$, then the fitter chromosome is selected for parenthood; otherwise, the less-fit chromosome is selected.

27.5 Modifications and Enhancements: Crossover

27.5.1 Multi-Point Crossover

The single-point crossover operator that we have outlined here suffers from what is known as *positional bias*. That is, the performance of the GA depends, in part, on the order that the variables occur in the chromosome. So, genes in loci 1 and 2 will often be crossed over together, simply because of their proximity to each other, whereas genes in loci 1 and 7 will rarely cross over together. Now, if this positioning reflects natural relationships within the data and among the variables, then this is not such a concern, but such a priori knowledge is relatively rare.

The solution is to perform *multi-point crossover*, as follows. First, randomly select a set of crossover points, and split the parent chromosomes at those points. Then, to form the children, recombine the segments by alternating between the parents, as illustrated in Figure 27.3.

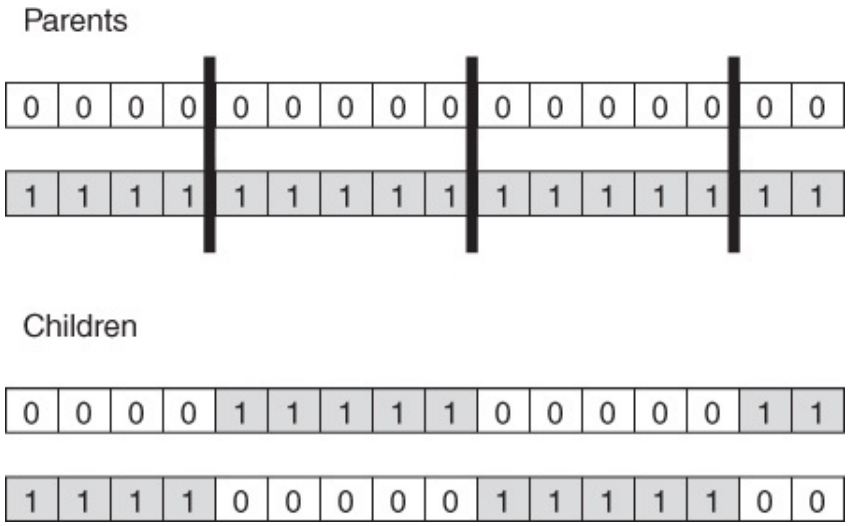

Figure 27.3 Multi-point crossover.

27.5.2 Uniform Crossover

Another alternative crossover operator is *uniform crossover*. In uniform crossover, the first child is generated as follows. Each gene is randomly assigned to be that of either one or the other parent, with 50% probability. The second child would then take the inverse of the first child. One advantage of uniform crossover is that the inherited genes are independent of position. Uniform crossover is illustrated in Figure 27.4. A modified version of uniform crossover would be to allow the probabilities to depend on the fitness of the respective parents.

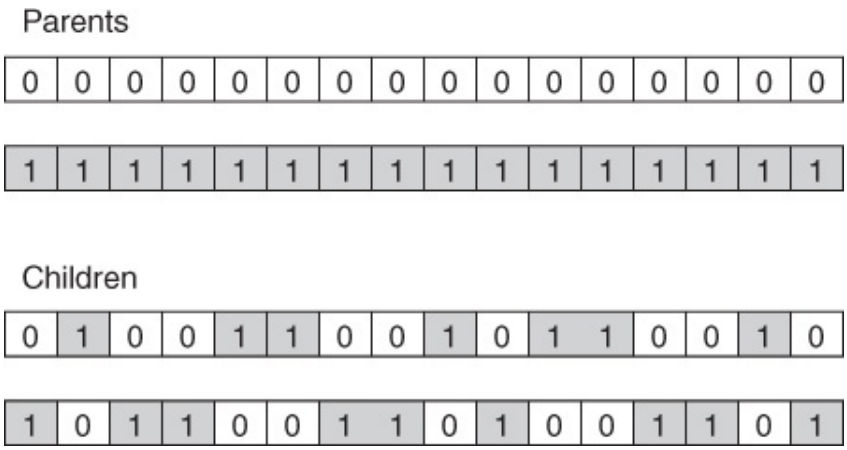

Figure 27.4 Uniform crossover.

Eiben and Smith[10] discuss the roles of crossover and mutation, and the cooperation and competition between them with respect to the search space. They describe crossover as *explorative*, discovering promising new regions in the search space by making a large jump to a region between the two parent areas. And they describe mutation as *exploitative*, optimizing present information within an already discovered promising region, creating small random deviations and thereby not wandering far from the parents. Crossover and mutation complement each other, because only crossover can bring together information from both parents, and only mutation can introduce completely new information.

27.6 Genetic Algorithms for Real-Valued Variables

The original framework for GAs was developed for binary-encoded data, because the operations of crossover and mutation worked naturally and well with such data. However, most data mining data come in the form of real numbers, often with many decimals worth of precision.

Some analysts have tried quantizing the real-valued (continuous) data into binary form. However, to re-express the real-valued data in binary terms will necessarily result in a loss of information, due to the degradation in precision caused by rounding to the nearest binary digit. To combat this loss in precision, each binary chromosome would need to be made longer, adding digits that will inevitably impair the speed of the algorithm.

Therefore, methods for applying GAs directly to real-valued data have been investigated. Eiben and Smith suggest the following methods for performing the crossover operation.

27.6.1 Single Arithmetic Crossover

Let the parents be $\langle x_1, x_2, \cdots, x_n \rangle$ and $\langle y_1, y_2, \cdots, y_n \rangle$. Pick the *k*th gene at random. Then, let the first child be of the form $\langle x_1, x_2, \cdots, \alpha \cdot y_k + (1-\alpha) \cdot x_k, \cdots, x_n \rangle$, and the second child be of the form $\langle y_1, y_2, \cdots, \alpha \cdot x_k + (1-\alpha) \cdot y_k, \cdots, y_n \rangle$, for $0 \le \alpha \le 1$.

For example, let the parents be $\langle 0.5, 1.0, 1.5, 2.0 \rangle$ and $\langle 0.2, 0.7, 0.2, 0.7 \rangle$, let $\alpha = 0.4$, and select the third gene at random. Then, single arithmetic crossover would produce the first child to be

$$\langle 0.5, 1.0, (0.4) \cdot (0.2) + (0.6) \cdot (1.5), 2.0 \rangle = \langle 0.5, 1.0, 0.98, 2.0 \rangle,$$

and the second child to be

$$\langle 0.2, 0.7, (0.4) \cdot (1.5) + (0.6) \cdot (0.2), 0.7 \rangle = \langle 0.2, 0.7, 0.72, 0.7 \rangle.$$

27.6.2 Simple Arithmetic Crossover

Let the parents be $\langle x_1, x_2, \cdots, x_n \rangle$ and $\langle y_1, y_2, \cdots, y_n \rangle$. Pick the *k*th gene at random, and mix values for all genes at this point and beyond. That is, let the first child be of the form $\langle x_1, x_2, \cdots, \alpha \cdot y_k + (1-\alpha) \cdot x_k, \cdots, \alpha \cdot y_n + (1-\alpha) \cdot x_n \rangle$, and the second child be of the form $\langle y_1, y_2, \cdots, \alpha \cdot x_k + (1-\alpha) \cdot y_k, \cdots, \alpha \cdot x_n + (1-\alpha) \cdot y_n \rangle$, for $0 \le \alpha \le 1$.

For example, let the parents be $\langle 0.5, 1.0, 1.5, 2.0 \rangle$ and $\langle 0.2, 0.7, 0.2, 0.7 \rangle$, let $\alpha = 0.4$, and select the third gene at random. Then, simple arithmetic crossover would produce the first child to be

$$\begin{aligned} &\langle 0.5, 1.0, (0.4) \cdot (0.2) + (0.6) \cdot (1.5), (0.4) \cdot (0.7) + (0.6) \cdot (2.0) \rangle \\ &\quad = \langle 0.5, 1.0, 0.98, 1.48 \rangle, \end{aligned}$$

and the second child to be

$$\begin{aligned} &\langle 0.2, 0.7, (0.4) \cdot (1.5) + (0.6) \cdot (0.2), (0.4) \cdot (2.0) + (0.6) \cdot (0.7) \rangle \\ &\quad = \langle 0.2, 0.7, 0.72, 1.22 \rangle \end{aligned}$$

27.6.3 Whole Arithmetic Crossover

Let the parents be $\langle x_1, x_2, \cdots, x_n \rangle$ and $\langle y_1, y_2, \cdots, y_n \rangle$. Perform the above mixture to the entire

vector for each parent. The calculation of the child vectors is left as an exercise. Note that, for each of these arithmetic crossover techniques, the affected genes represent intermediate points between the parents' values, with $\alpha = 0.5$ generating the mean of the parents' values.

27.6.4 Discrete Crossover

Here, each gene in the child chromosome is chosen with uniform probability to be the gene of one or the other of the parents' chromosomes. For example, let the parents be $\langle 0.5,\ 1.0,\ 1.5,\ 2.0 \rangle$ and $\langle 0.2,\ 0.7,\ 0.2,\ 0.7 \rangle$, one possible child could be $\langle 0.2,\ 0.7,\ 1.5,\ 0.7 \rangle$, with the third gene coming directly from the first parent and the others coming from the second parent.

27.6.5 Normally Distributed Mutation

To avoid converging too quickly toward a local optimum, a normally distributed “random shock” may be added to each variable. The distribution should be normal, with a mean of zero, and a standard deviation of σ, which controls the amount of change (as most random shocks will lie within one σ of the original variable value). If the resulting mutated variable lies outside the allowable range, then its value should be reset so that it lies within the range. If all variables are mutated, then clearly $p_{\mathrm{m}} = 1$ in this case.

For example, suppose the mutation distribution is $\mathrm{Normal}(\mu = 0,\ \sigma = 0.1)$, and that we wish to apply the mutation to the child chromosome from the discrete crossover example, $\langle 0.2,\ 0.7,\ 1.5,\ 0.7 \rangle$. Assume that the four random shocks generated from this distribution are 0.05, −0.17, −0.03, and 0.08. Then, the child chromosome becomes $\langle 0.2 + 0.05,\ 0.7 - 0.17,\ 1.5 - 0.03,\ 0.7 + 0.08 \rangle = \langle 0.25,\ 0.53,\ 1.47,\ 0.78 \rangle$.