University of Babylon, College of science for women
Dept. of Computer science

Evolutionary Computing

Dr. Salah Al-Obaidi

Lecture #5: Representation, Mutation, and Recombination

Fall 2024

Contents

3. Representation, Mutation, and Recombination

There are two fundamental forces that form the basis of evolutionary systems: **variation** and **selection**. In the following we discuss the EA components behind the first one. Since variation operators work at the equivalent of the genetic level, that is to say they work on the representation of solutions, rather than on solutions themselves, In the following sections, we look more closely at some commonly used representations, binary representation, as an example, and the genetic operators that might be applied to them.

3.1 Representation and the Roles of Variation Operators

The first stage of building any evolutionary algorithm is to decide on a genetic **representation** of a candidate solution to the problem. When choosing a representation, it is important to choose the right representation for the problem being solved. In many cases there will be a range of options, and getting the representation right is one of the most difficult parts of designing a good evolutionary algorithm.

Mutation is the generic name given to those variation operators that use only one parent and create one child by applying some kind of randomised change to the representation (genotype). The form taken depends on the choice of encoding used, which is often introduced to regulate the intensity or magnitude of mutation. Depending on the given implementation, this can be mutation probability, mutation rate, mutation step size, etc.

Recombination, the process whereby a new individual solution is created from the information contained within two (or more) parent solutions, is considered by many to be one of the most important features in evolutionary algorithms. A lot of research activity has focused on it as the primary mechanism for creating diversity, with mutation considered as a background search operator. Regardless of the merits of different viewpoints, the ability to combine partial solutions via recombination is certainly one of the features that most distinguishes EAs from other global optimisation algorithms.

Although the term recombination has come to be used for the more general case, early authors used the term **crossover**, motivated by the biological analogy to meiosis. Recombination operators are usually applied probabilistically according to a crossover rate $\boldsymbol{p_c}$. Usually two parents are selected and two offspring are created via recombination of the two parents with probability $\boldsymbol{p_c}$; or by simply copying the parents, with probability $\boldsymbol{1 - p_c}$.

3.2 Binary Representation

It is one of the simplest representations used. This is one of the earliest representations, and historically many genetic algorithms (GAs) have used this representation almost independently of the problem they were trying to solve. Here the genotype consists simply of a string of binary digits – a bit-string. For a particular application we have to decide how long the string should be, and how we will interpret it to produce a phenotype.

One of the problems of coding numbers in binary is that different bits have different significance, and so the effect of a single bit mutation is very variable. Using standard binary code has the disadvantage that the Hamming distance between two consecutive integers is often not equal to one. If the goal is to evolve an integer number, you would like to have equal probabilities of changing a $\mathbf{7}$ into an $\mathbf{8}$ or a $\mathbf{6}$. However, changing $\mathbf{0111}$ to $\mathbf{1000}$ requires four bit-flips while changing it to $\mathbf{0110}$ takes just one. Thus with a mutation operator that randomly, and independently, changes each allele value with probability $\boldsymbol{p_m < 0.5}$, the probability of changing $\mathbf{7}$ to $\mathbf{8}$ is much less than changing $\mathbf{7}$ to $\mathbf{6}$. This can be helped by using Gray coding, a variation on the way that integers are mapped on bit strings where consecutive integers always have **Hamming distance of one**.

3.2.1 Mutation for Binary Representation

The most common mutation operator for binary encodings considers each gene separately and allows each bit to flip (i.e., from $\mathbf{1}$ to $\mathbf{0}$ or $\mathbf{0}$ to $\mathbf{1}$) with a small probability $\boldsymbol{p_m}$. The actual number of values changed is thus not fixed, but depends on the sequence of random numbers drawn, so for an encoding of length $\boldsymbol{L}$, on average $\boldsymbol{L \cdot p_m}$ values will be changed. In Figure 3.1 this is illustrated for the case where the third, fourth, and eighth random values generated are less than the bitwise mutation rate p_m.

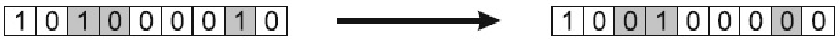

Figure 3.1: Bitwise mutation for binary encodings.

A number of studies and recommendations have been made for the choice of suitable values for the bitwise mutation rate $\boldsymbol{p_m}$. Most binary coded GAs use mutation rates in a range such that on average between one gene per generation and one gene per offspring is mutated. However, it is worth noting at the outset that the most suitable choice to use depends on the desired outcome. For example, does the application require a population in which all members have high fitness, or simply that one highly fit individual is found? The former suggests a lower mutation rate, less likely to disrupt good solutions. In the latter case one might choose a higher mutation rate if the potential benefits of ensuring good coverage of the search space outweighed the cost of disrupting copies of good solutions.

3.2.2 Recombination for Binary Representation

Three standard forms of recombination are generally used for binary representations. They all start from two parents and create two children, although all of these have been extended to the more general case where a number of parents may be used, and there are also situations in which only one of the offspring might be considered.

One-Point Crossover

One-point crossover was the original recombination operator. It works by choosing a random number $\boldsymbol{r}$ in the range $[\mathbf{1}, \boldsymbol{l}-\mathbf{1}]$ (with $\boldsymbol{l}$ the length of the encoding), and then splitting both parents at this point and creating the two children by exchanging the tails (Figure 3.2, top). Note that by using the range $[\mathbf{1}, \boldsymbol{l}-\mathbf{1}]$ the crossover point is prevented from falling before the first position $(\boldsymbol{r}=\mathbf{0})$ or after the last position $(\boldsymbol{r}=\boldsymbol{l})$.

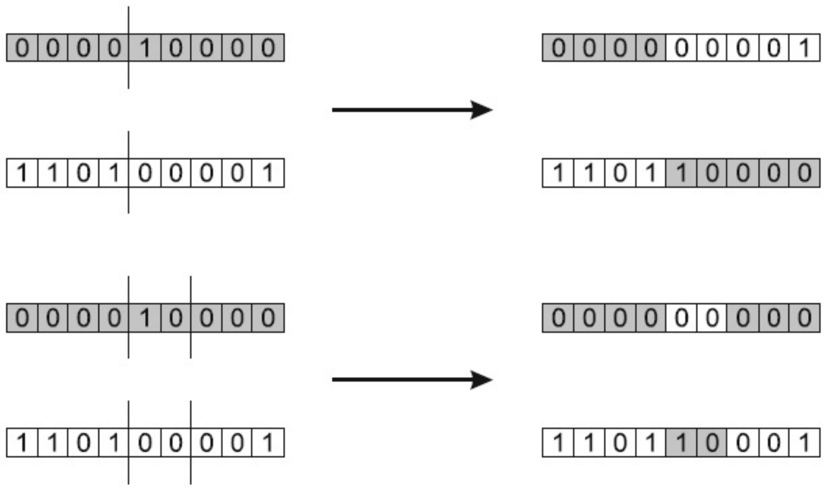

Figure 3.2: One-point crossover (top) and n-point crossover with $n = 2$ (bottom).

n-Point Crossover

One-point crossover can easily be generalised to $\boldsymbol{n}$-point crossover, where the chromosome is broken into more than two segments of contiguous genes and the offspring are created by taking alternative segments from the parents. In practice, this means choosing n random crossover points in $[\mathbf{1}, \boldsymbol{l}-\mathbf{1}]$, which is illustrated in Figure 3.2 (bottom) for $\boldsymbol{n}=\mathbf{2}$.

Uniform Crossover

The previous two operators worked by dividing the parents into a number of sections of contiguous genes and reassembling them to produce offspring. In contrast to this, uniform crossover works by treating each gene independently and making a random choice as to which parent it should be inherited from. This is implemented by generating a string of l random variables from a uniform distribution over $[0,1]$. In each position, if the value is below a parameter p (usually 0.5), the gene is inherited from the first parent; otherwise from the second. The second offspring is created using the inverse mapping. This is illustrated in Figure 3.4.

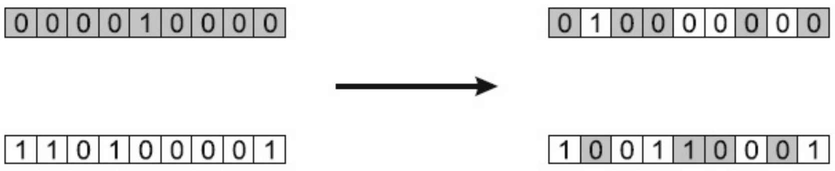

Figure 3.3: Uniform crossover. The array [0.3, 0.6, 0.1, 0.4, 0.8, 0.7, 0.3, 0.5, 0.3] of random numbers and $p = 0.5$ were used to decide inheritance for this example.

In our discussion so far, we have suggested that in the absence of prior information, recombination worked by randomly mixing parts of the parents. However, as Figure 3.2 illustrates, n-point crossover has an inherent bias in that it tends to keep together genes that are located close to each other in the representation. Furthermore, when n is odd (e.g., **one-point crossover**), there is a strong bias against keeping together combinations of genes that are located at opposite ends of the representation. These effects are known as **positional bias**. In contrast, uniform crossover does not exhibit any positional bias. However, unlike n-point crossover, uniform crossover does have a strong tendency towards transmitting 50% of the genes from each parent and against transmitting an offspring a large number of coadapted genes from one parent. This is known as **distributional bias**.

Other representations are also used, such as Integer and Real-Valued or Floating-Point Representation, since binary representations are not always the most suitable if our problem more naturally maps onto a representation where different genes can take one of a set of values.

3.3 Integer Representation

As we mentioned before, binary representations are not always the most suitable if our problem more naturally maps onto a representation where different genes can take one of a set of values. One obvious example of when this might occur is the problem of finding the optimal values for a set of variables that all take integer values. These values might be unrestricted (i.e., any integer value is permissible), or might be restricted to a finite set: for example, if we are trying to evolve a path on a square grid, we might restrict the values to the set 0,1,2,3 representing North, East, South, West. In either case an integer encoding is probably more suitable than a binary encoding.

3.3.1 Mutation for Integer Representations

For integer encoding, there are two principal forms of mutation used, both of which mutate each gene independently with user-defined probability p_m.

Random Resetting: Here the bit-flipping mutation of binary encoding is extended to random resetting. In each position independently, with probability $\boldsymbol{p_m}$, a new value is chosen at random from the set of permissible values. This is the most suitable operator to use when the genes encode for cardinal attributes since all other gene values are equally likely to be chosen.

Creep Mutation: This scheme was designed for ordinal attributes and works by adding a small (positive or negative) value to each gene with probability $\boldsymbol{p_m}$. Usually these values are sampled randomly for each position, from a distribution that is symmetric about zero, and is more likely to generate small changes than large ones. It should be noted that creep mutation requires a number of parameters controlling the distribution from which the random numbers are drawn, and hence the size of the steps that mutation takes in the search space. Finding appropriate settings for these parameters may not be easy, and it is sometimes common to use more than one mutation operator in tandem from integer-based problems. For example, both a "big creep" and a "little creep" operator are used. Alternatively, random resetting might be used with low probability, in conjunction with a creep operator that tended to make small changes relative to the range of permissible values.

3.3.2 Recombination for Integer Representation

For representations where each gene has a finite number of possible allele values (such as integers), it is normal to use the same set of operators as for binary representations. On the one hand, these operators are valid: the offspring would not fall outside the given genotype space. On the other hand, these operators are also sufficient: it usually does not make sense to consider 'blending' allele values of this sort. For example, even if genes represent integer values, averaging an even and an odd integer yields a non-integral result.

3.4 Real-Valued or Floating-Point Representation

Often, the most sensible way to represent a candidate solution to a problem is to have a string of real values. This occurs when the values that we want to represent as genes come from a continuous rather than a discrete distribution — for example, if they represent physical quantities such as the length, width, height, or weight of some component of a design that can be specified within a tolerance smaller than integer values. An example might be if we wished to use an EA to evolve the weights on the connections between the nodes in an artificial neural network. Of course, on a computer the precision of these real values is actually limited by the implementation, so we will refer to them as floating-point numbers. The genotype for a solution with k genes is now a vector $\langle \boldsymbol{x_1}, ..., \boldsymbol{x_k} \rangle$ with $\boldsymbol{x_i} \in \mathbb{R}$.

When designing the encoding and variation operators, it is worth considering whether there are any natural relations between the possible values that an attribute can take. This might be obvious for **ordinal** attributes such as integers (2 is more like 3 than it is 389), but for **cardinal attributes** such as the compass points above, there may not be a natural ordering.

3.4.1 Mutation for Real-Valued Representation

For floating-point representations, it is normal to ignore the discretisation imposed by hardware and consider the allele values as coming from a continuous rather than a discrete distribution, so the forms of mutation described above are no longer applicable. Instead, it is common to change the allele value of each gene randomly within its domain given by a lower $\boldsymbol{L_i}$ and upper $\boldsymbol{U_i}$ bound, resulting in the following transformation:

$$\langle x_1, ..., x_n \rangle \rightarrow \langle x_1^{'}, ..., x_n^{'} \rangle, \ \ where \ \ x_i, x_i^{'} \in [L_i, U_i]$$

As with integer representations, two types can be distinguished according to the probability distribution from which the new gene values are drawn: *uniform* and *nonuniform* mutation.

Uniform Mutation For this operator the values of $\boldsymbol{x_i'}$ are drawn uniformly randomly from $[\boldsymbol{L_i U_i}]$. This is the most straightforward option, analogous to bit-flipping for binary encodings and the random resetting for integer encodings.

Nonuniform Mutation Perhaps the most common form of nonuniform mutation used with floating-point representations takes a form analogous to the creep mutation for integers. It is designed so that usually, but not always, the amount of change introduced is small. This is achieved by adding to the current gene value an amount drawn randomly from a Gaussian distribution with mean zero and user-specified standard deviation, and then shrinking the resulting value to the range $[\boldsymbol{L_i}, \boldsymbol{U_i}]$ if necessary. This distribution, shown in Equation 3.1, has the feature that the probability of drawing a random number with any given magnitude is a rapidly decreasing function of the standard deviation $\boldsymbol{\sigma}$. Approximately, two-thirds of the samples drawn will lie within plus or minus one standard deviation, which means that most of the changes made will be small, but there is a nonzero probability of generating very large changes since the tail of the distribution never reaches zero

$$p(\Delta x_i) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(\Delta x_i - \xi)^2}{2\sigma^2}} \tag{3.1}$$

Thus, the $\boldsymbol{\sigma}$ value is a parameter of the algorithm that determines the extent to which given values $\boldsymbol{x_i}$ are perturbed by the mutation operator. For this reason $\boldsymbol{\sigma}$ is often called the mutation step size. It is normal practice to apply this operator with probability one per gene, and instead the mutation parameter is used to control the standard deviation of the Gaussian and hence the probability distribution of the step sizes taken.

3.4.2 Recombination Operators for Real-Valued Representation

In general, we have two options for recombining two floating-point strings:

- First, using an analogous operator to those used for bit-strings, but now split between floats. In other words, an allele is one floating-point value instead of one bit. This has the disadvantage that only mutation can insert new values into the population, since recombination only gives us new combinations of existing values. Recombination operators of this type for floating-point representations are known as **discrete recombination** and have the property that if we are creating an offspring $\boldsymbol{z}$ from parents $\boldsymbol{x}$ and $\boldsymbol{y}$, then the allele value for gene $\boldsymbol{i}$ is given by $\boldsymbol{z_i = x_i}$ or $\boldsymbol{y_i}$ with equal likelihood.

- Second, using an operator that, in each gene position, creates a new allele value in the offspring that lies between those of the parents. Using the terminology above, we have $\boldsymbol{z_i = \alpha x_i + (1 - \alpha) y_i}$ for some $\boldsymbol{\alpha}$ in $\boldsymbol{[0, 1]}$. In this way, recombination is now able to create new gene material, but it has the disadvantage that as a result of the averaging process the range of the allele values in the population for each gene is reduced. Operators of this type are known as **intermediate** or **arithmetic recombination**.

There three types of arithmetic recombination. In all of these, the choice of the parameter $\boldsymbol{\alpha}$ is sometimes made at random over $[0, 1]$, but in practice it is common to use a constant value, often **0.5** (in which case we have **uniform arithmetic recombination**).

1. **Simple Arithmetic Recombination**: First pick a recombination point $\boldsymbol{k}$. Then, for child 1, take the first $\boldsymbol{k}$ floats of parent 1 and put them into the child. The rest is the arithmetic average of parent 1 and 2:

   $$Child1 : \langle x_1, ..., x_k, \alpha \cdot y_{k+1} + (1-\alpha) \cdot x_{k+1}, ..., \alpha \cdot y_n + (1-\alpha) \cdot x_n \rangle.$$

   Child 2 is analogous, with **x** and **y** reversed (Figure 3.4 top).

2. **Single Arithmetic Recombination**: Pick a random allele $\boldsymbol{k}$. At that position, take the arithmetic average of the two parents. The other points are the points from the parents, i.e.:

   $$Child1 : \langle x_1, ..., x_{k-1}, \alpha \cdot y_k + (1-\alpha) \cdot x_k, x_{k+1}, ..., x_n \rangle.$$

   The second child is created in the same way with x and y reversed (Figure 3.4, middle).

3. **Whole Arithmetic Recombination**: This is the most commonly used operator and works by taking the weighted sum of the two parental alleles for each gene, i.e.:

   $$Child1 = \alpha \cdot x + (1-\alpha) \cdot y, \quad Child2 = \alpha \cdot y + (1-\alpha) \cdot x.$$

   This is illustrated in Figure 3.4, bottom. As the example shows, if $\boldsymbol{\alpha = 1/2}$ the two offspring will be identical for this operator.

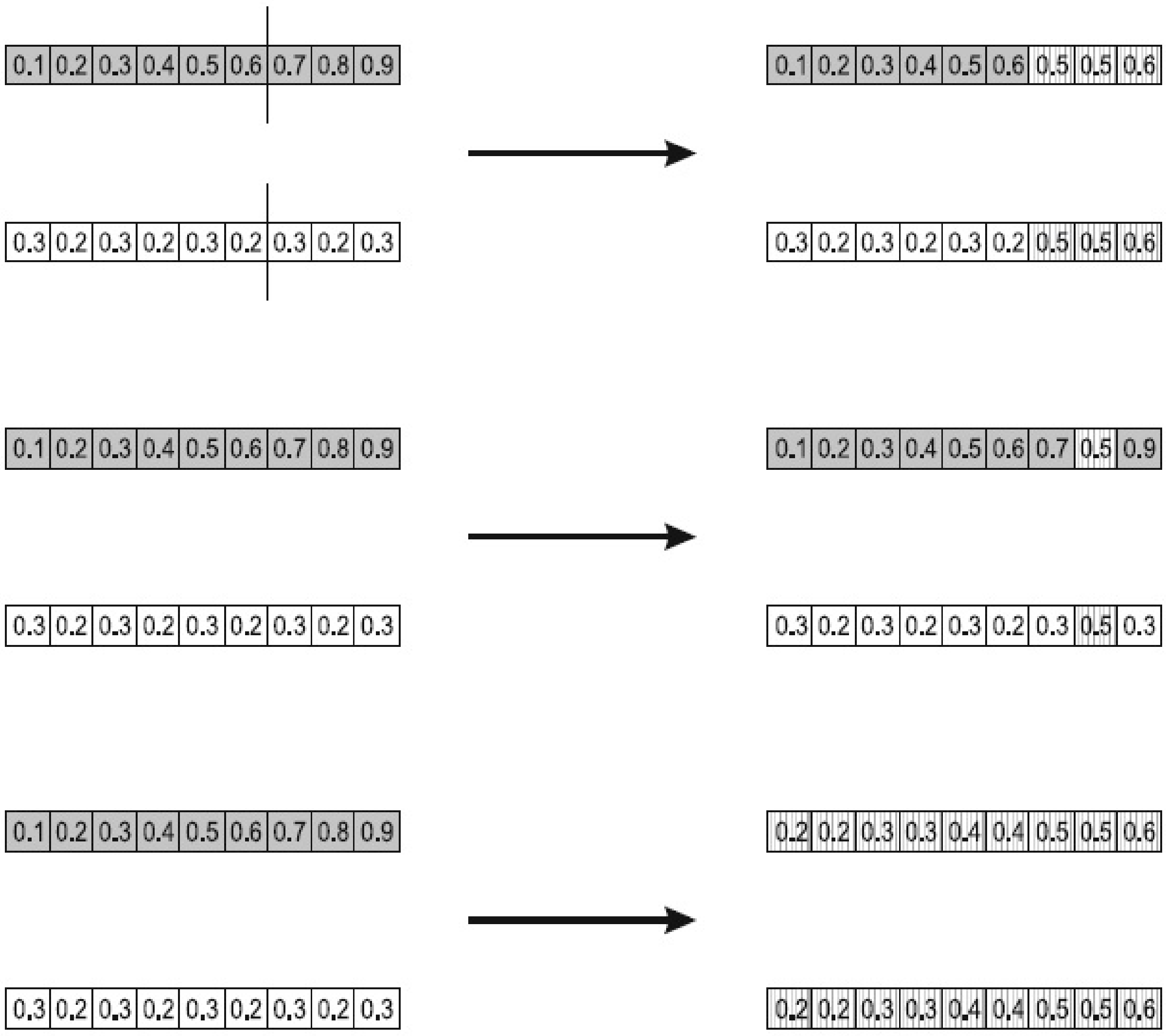

Figure 3.4: Simple arithmetic recombination with $k = 6$, $\alpha = 1/2$ (top), single arithmetic recombination with $k = 8$, $\alpha = 1/2$ (middle), whole arithmetic recombination with $\alpha = 1/2$ (bottom).