5

The Manipulation of Coded Designs *(Genotypes)*

5.1 Introduction

This chapter describes the investigation and creation of the second element of the system: the genotypes, and the genetic operators which manipulate them.

Natural evolution operates on a set of coded instructions for how organisms should be grown (known as DNA), it does not operate directly on the organisms themselves (Paton, 1994). Likewise, the genetic algorithm operates on coded parameter values and not directly on solutions (Goldberg, 1989). This means that to enable the evolutionary design system to modify designs defined by the spatial partitioning representation described in the previous chapter, these designs, or phenotypes, must be coded as genotypes.

There are a number of different ways in which a phenotype can be coded as a genotype within a GA. The values of parameters can be coded as alleles of genes by converting them to binary or other number bases (Goldberg, 1991a). Chromosomes can be constructed from lists of rigidly ordered alleles, unordered sets, or as hierarchically structured groups of alleles (Oppacher & Deugo, 1995). A genotype can consist of a single chromosome, a number of chromosomes, or even a number of pairs of chromosomes (Paton, 1994).

Genotypes are created and modified within a GA by the use of genetic operators such as crossover and mutation. These genetic operators define how the search space can be traversed. More precisely, the combination of genetic coding and operators defines exactly how new genotypes can be created from combinations of old ones, and how alleles in genotypes can be modified. Put another way, the genetic coding and operators of a GA determine how the GA moves from one solution to another in the search space, and hence determines the efficiency and effectiveness of the searching process (Davis, 1991).

5.2 Genetic Coding of Designs

5.2.1 Coding of Parameter Values

Within the generic evolutionary design system, a phenotype specifies the shape of a solid object by using a number of primitive shapes in combination. Since every primitive is defined by nine parameters, every phenotype consists of a list of *n* multiples of nine parameters (where *n* is the number of primitives in the design). In order to allow a genetic algorithm to manipulate the values of these parameters, they must be coded in some way to form genotypes. Hence, the genotypes for the system must consist of *n* multiples of nine coded parameters, or genes.

Often when parameters are coded as genes, the coded parameter values (alleles) are simply converted to an alternative number base. Perhaps the two most common forms of coding are binary coding, giving a genetic alphabet of cardinality two ('1' and '0'), or real coding, where alleles are stored in decimal or other high-cardinality codings (Goldberg, 1991a).

Real coding, using decimal numbers, is the simplest form of 'coding' used within GAs. Since parameter values are normally also stored in decimal, 'real coded' parameter values remain unchanged, meaning that real-coded genotypes are typically identical to phenotypes. However, the evolutionary design system uses a distinct mapping stage to convert genotypes to phenotypes (during which designs with overlapping primitives are corrected, and partial

designs are reflected to create symmetrical designs). This means that, for the system described in this thesis, a real-coded genotype is not identical to a phenotype.

Because of the inherent simplicity of real-coding, this method was used within early versions of the design system, and with significant success (Bentley & Wakefield, 1995a/1996a, 1995b). However, there are some known problems with real coding. Theoretical analyses suggest that alphabets with low cardinality (i.e. binary coding) will allow quicker and more effective convergence to good solutions than alphabets with higher cardinalities (Holland 1975, Goldberg, 1989). Natural evolution, the inspiration for GAs, uses a low cardinality alphabet of four (Dawkins, 1976). Moreover, Holland's Schema Theorem, which suggests how and why binary-coded GAs converge to good solutions, is unable to explain how real-coded GAs can converge to good solutions (Goldberg, 1991a).

Goldberg has attempted to overcome this limitation of the Schema Theorem by the creation of a new theory which postulates that selection dominates early GA performance and then restricts subsequent search to *virtual characters* with above average function values (Goldberg, 1991a). In other words, he suggests that the real-coded GA "turns big alphabets into little alphabets" (Goldberg, 1991a), i.e. alphabets with large cardinalities are manipulated in terms of virtual characters from much smaller virtual alphabets. However, Goldberg also states that this theory suggests that real-coded GAs can be prevented, or 'blocked', from finding global optima by certain types of problem (as well as still being susceptible to deceptive problems).

Because of these problems, it was decided quite early in the development of the system to change the genetic coding of parameters to binary. Hence, every parameter value in the system is coded as a 16-bit binary number, using sign-and-magnitude notation (i.e. the most significant bit denotes the sign of the number). The last seven bits define the fractional part of the coded value, see fig. 5.1.

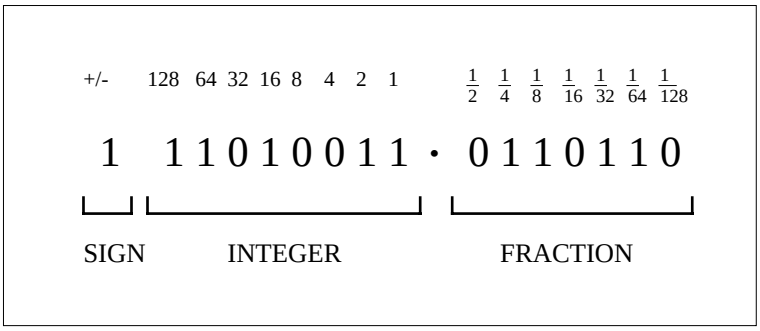

Fig. 5.1 Binary coding of parameter values in the system.

In decimal, this corresponds to a range of -255.992 to 255.992 in steps of 1/128. So, for example, the allele: 1 0 0 0 1 0 1 0 1 . 0 0 0 0 1 0 1

corresponds to the parameter value: - 21 . 0390625

When the performance of the evolutionary design system with binary coding was compared with the earlier version using real coding, a slight, but noticeable improvement was seen. Although typically the binary-coded GA took more generations to converge to good solutions, the evolved designs were often fitter than those produced by the real-coded GA.

As mentioned above, the genotype for an individual design must contain *n* groups of nine coded parameter values. For the sake of simplicity, every genotype consists of a single chromosome. However, because the number of primitives that define designs is variable, the number of groups of nine alleles in the chromosome is also variable. This means that the organisation of alleles within a chromosome and the creation of the crossover operator to manipulate such chromosomes is crucial to ensure that evolution can proceed.

5.2.2 Crossover and Locus-Specific Genetic Organisations

As well as encoding every parameter value as a 16-bit allele, a method of organisation is needed in the chromosome of an individual, to allow the specification of which gene an allele belongs to. In other words, the *meaning* of each allele must be stored.

The most common way of defining the meaning of alleles in a GA is by position in the chromosome (Goldberg, 1989). Typically, such GAs use rigidly ordered lists of alleles, with every locus (position) in the chromosome corresponding to a specific pre-determined gene. For example, the fifth allele in every chromosome might define the value of the gene for 'width', and the seventh allele might define the value of the gene for 'angle2'.

Whilst this is a quick and efficient method of specifying the meaning of alleles, allowing the decoding from alleles to parameter values to be trivial, it would cause significant problems in the evolutionary design system. For example, consider two individuals with locus-specific genes in their chromosomes, picked for reproduction by the GA. One has three groups of alleles (i.e. a three-primitive design), the other has had the middle group deleted by mutation, leaving two groups of alleles (i.e. a two-primitive design) and a blank space in its chromosome:

Parent 1: `abcdefghi jklmnopqr stuvwxyzα`

Parent 2: `ABCDEFGHI --------- STUVWXYZA`

If crossover was then applied to generate two offspring, with a random crossover point of, say 12, the resulting two children would both have missing alleles in the middle of their chromosomes (shown by dashes):

Child 1: `abcdefghi jkl------ STUVWXYZA`

Child 2: `ABCDEFGHI ---mnopqr stuvwxyzα`

Plainly, both offspring have only a partial specification of the second primitive of the designs, i.e. these child designs cannot be decoded or subsequently evaluated - they are meaningless.

Because every allele has its meaning defined by its position in the chromosome, a blank space in a chromosome (e.g. Parent two in the example above) cannot be removed by shifting later

alleles forwards, without the meaning of every shifted allele being changed. Moreover, even if this was done (e.g. all alleles of the genes for primitive three being changed to alleles of genes for primitive two), the chromosome would simply have its 'gap' at the end (i.e. no alleles of genes for primitive three), potentially resulting in offspring that are damaging to evolution. To illustrate this, consider Parent 2 of the previous example. The blank space in the middle of its chromosome could be removed by shifting all later alleles forward:

Parent 2: `ABCDEFGHI STUVWXYZA ---------`

However, the alleles that defined the position and shape of primitive three in the phenotype, now define the position and shape of primitive two. Whilst the actual design would appear identical, a primitive number has been changed from three to two. This means that if this genotype is picked for reproduction and crossed over with Parent 1, two problems could occur. First, if the random crossover point was, say 15, the offspring would be:

Child 1: `abcdefghi jklmnoYZA ---------`

Child 2: `ABCDEFGHI STUVWXpqr stuvwxyzα`

Although these offspring are meaningful, with Child 1 defining a two-primitive design and Child 2 defining a three-primitive design, the second primitive of both has been impaired. This is because Parent 2's second primitive was originally its third primitive (which was shifted back), meaning that the second primitive of each child has been constructed from a second primitive and a third primitive from the parents. While the GA would cope with this incompatible crossover during the early stages of evolution, at later stages when the position and shape of all primitives has been evolved to some precision, creating a new primitive out of two very differently sized and positioned primitives would usually result in a very unfit design. Indeed, it is possible that, if crossover was permitted to mix any primitives together, all the primitives in a design could eventually converge to become identical. Hence, the evolutionary design system requires that only *compatible* groups of alleles are crossed over.

The second problem with shifting alleles forward to fill a blank space in a chromosome is that the gap is simply moved to the end. For example, if two parents were chosen for reproduction, and Parent 2 had had its alleles shifted back in order to close up a gap:

Parent 1: `abcdefghi jklmnopqr stuvwxyzα`

Parent 2: `ABCDEFGHI STUVWXYZA ---------`

With a random crossover point of, say 22, the resulting offspring would be meaningless, just as in the first example:

Child 1: `abcdefghi jklmnopqr stuv-----`

Child 2: `ABCDEFGHI STUVWXYZA ----wxyzα`

Consequently, standard crossover combined with a conventional locus-specific genetic organisation is incapable of reliably producing meaningful offspring from parents with chromosomes of different lengths in a GA (Harvey, 1992).

5.2.3 Alternative Crossover Operators and Genetic Representations

There are a number of existing attempts to solve the problem of applying crossover to variable-length chromosomes. Smith used a new 'alignment' stage within his classifier system LS-1, to initially align rules at boundaries before his version of crossover could operate on individuals consisting of rule sets (Smith, 1984). His version of crossover permits a variable number of fixed-length rules in chromosomes. Since the evolutionary design system needs a variable number of fixed-length coded primitives, this idea could allow crossover to align groups of alleles in chromosomes for the system, and ensure that meaningful offspring are always created. However, this method would also permit incompatible groups of alleles to be mixed and so is not suitable for this system.

Koza evolved programs defined as LISP expressions which are arranged in hierarchical tree-structures (Koza, 1990). Koza's crossover simply allows any branches of the two parent trees to be interchanged. Although the genotypes within the evolutionary design system can be defined hierarchically as will be shown later, they would not survive the simple methods of Koza and remain meaningful, (i.e., a tree-like genotype with too many or too few branches would define too many or too few parameter values).

Harp and Samad code neural networks as genotypes, using a fixed length group of genes to define a single layer in the network, with the number of such layers being variable (Harp & Samad, 1992). Their crossover ensures that if a group is split in one parent, another group is split in the same position in the other parent, thus ensuring that the resulting offspring is meaningful. Harvey also uses a GA to evolve neural networks, this time using a syntactic comparison technique to "minimise the difference between the swapped segments", and thus minimise the loss of meaning in offspring (Harvey, 1992). However, whilst these methods could be used to create meaningful offspring in the design system, again they do not ensure that compatible groups of alleles are crossed, and also seem unnecessarily computationally expensive.

Goldberg's Messy GAs (Deb & Goldberg, 1991) use a common technique of *labelling* every allele. When every allele is labelled, a chromosome becomes more like a set, with values in any order, yet still 'knowing' which gene they correspond to. Messy GAs use very simple 'cut' and 'splice' operators in the place of crossover, allowing an offspring to inherit random alleles from each parent. This inevitably allows individuals to have duplicate alleles or missing alleles. An arbitrary rule determines which duplicated allele should be used as the actual value for an overspecified gene (e.g. always pick the first). If there is no allele for a gene (i.e. the gene is underspecified), a 'competitive template' (similar to a look-up table) is used to set the value. Because all alleles are labelled, this method could be used successfully within the evolutionary design system, ensuring no loss of meaning and that no incompatible groups of alleles are ever mixed. However, this method does tend to produce enormously long chromosomes with many redundant alleles (Deb & Goldberg, 1991), and the fixed-length 'competitive template' would place a limit on the number of new groups of alleles that mutation could add to a chromosome.

Radcliffe also labels each allele, calling this an allelic representation (where every allele is a <gene, value> pair), then directly treats chromosomes as sets (Radcliffe, 1992). He has created a number of different versions of crossover based on set operations, e.g. random assorting recombination (RAR), random transmitting recombination (RTR) and random respectful

recombination (R^3) (Radcliffe & George, 1993). These methods could also be used successfully within the evolutionary design system, but, with every allele requiring two identifiers and the 16-bit value (i.e. <group_id, gene_id, value>), a considerable amount of unnecessary memory and computation (to find and compare the values) would be required.

For the generic evolutionary design system, the chromosome for a design defined by a number of primitives is plainly a two-level hierarchy consisting of a variable number of groups of nine alleles. However, the existing techniques mentioned above would all store and manipulate alleles regardless of this hierarchy. They would require that every allele should have one label to define which gene it belongs to, and another to define which group its gene belongs to. This is plainly nonsensical, for the very purpose of a hierarchical organisation is avoid such duplication. If a form of crossover existed which could take into account such hierarchies, then this unwanted repetition of labelling would be avoided.

Consequently, in order to overcome such deficiencies, a new crossover operator was developed for the evolutionary design system (Bentley & Wakefield, 1996d). This novel genetic operator, known as Hierarchical Crossover, removes the need for redundant duplicated allelic labels by directly manipulating hierarchically organised chromosomes. Moreover, this generic crossover operator can generate new meaningful chromosomes from variable-length chromosomes, independently of how the chromosomes are actually stored, by using the hierarchical meaning (or semantic hierarchy) of alleles, to ensure that only compatible alleles (or groups of alleles) are combined.

5.2.4 Semantic Hierarchy of Alleles

To allow hierarchical crossover to efficiently locate and identify the meaning of alleles, and thus ensure that only compatible alleles and groups of alleles are crossed, the general concept of a *semantic hierarchy* of alleles was developed for this work (Bentley & Wakefield, 1996d). The semantic hierarchy of a genotype is simply a 'tree of meaning' (i.e. a compositional or 'part_of' hierarchy). This hierarchy defines the semantics of a genotype, not the syntax, i.e. the semantic hierarchy is independent of how chromosomes are actually stored in memory.

Upon consideration, it becomes clear that all chromosomes have a semantic hierarchy of some form. For example, the traditional problem with *m* genes and *n* bits per gene, forms a hierarchy with two levels of meaning, see fig. 5.2. The problem of *l* groups of *m* genes with *n* bits per gene forms a semantic hierarchy with three levels of meaning, see fig. 5.3. What is perhaps unusual as shown in these hierarchies is the fact that the separate bits are considered as alleles, rather than collections of *n* bits being considered as alleles. However, this is done for a specific reason: by explicitly identifying the separate bits, every part of the chromosome can be made variable. In other words, hierarchical crossover allows GAs to evolve not only the value of each bit, but also the number of bits per gene (and hence the precision), the number of genes in the problem, the number of groups of genes (should the problem have a three or more level hierarchy), and so on. It should be noted that the hierarchy is of *meaning* only, i.e. the alleles shown in figures 5.2 & 5.3 are not part of the hierarchy, they are having their meaning defined by the hierarchy. Moreover, this meaning is independent of the order in which the alleles are actually stored in memory (as shown by the out-of-order nodes in the figures).

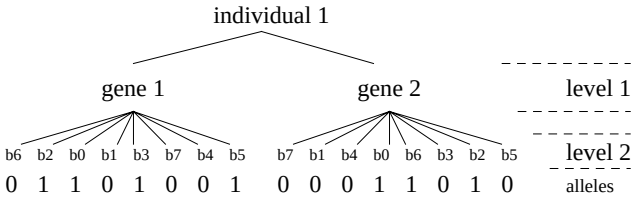

Fig. 5.2 Semantic hierarchy of a 16-bit, 2-gene chromosome.

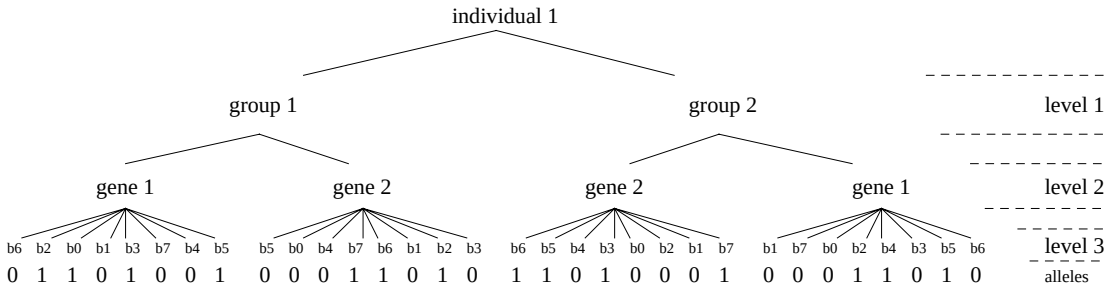

Fig. 5.3 Semantic hierarchy of a 32-bit, 4-gene, 2-group chromosome.

Any chromosome, no matter how it is stored in memory, has a semantic hierarchy and hence can be handled by a GA using hierarchical crossover. Although it is not necessary to store genotypes hierarchically, by doing so, substantial memory will be saved (especially with every bit requiring an identifier, in addition to every group of bits and every group of group of bits, and so on). Moreover, hierarchical crossover is designed to take advantage of hierarchically stored chromosomes, so the efficiency and speed of the operation is increased (i.e. as fast or faster than standard crossover) (Bentley & Wakefield, 1996d).

Finally, the previous explanation has described a problem with *m* genes and *n* bits per gene as having a semantic hierarchy with two levels of meaning. There are, in reality, two levels of meaning being ignored by such a statement: the level at which individual solutions in a population are situated, and, as is the case for many GAs, the population level. In other words, a GA could be said to operate on a semantic hierarchy of *j* populations of *k* solutions, each solution having, say *l* groups of *m* genes defined by *n* bits. However, since hierarchical (and all other types of) crossover always operates at the level of individual solutions, these higher levels of meaning can safely be ignored in this chapter.

5.2.5 Structured Hierarchical Chromosomes

Having outlined the general concept of a semantic hierarchy above, it should be clear that the evolutionary design system can be said to have a three-level hierarchy of meaning similar to the one depicted in fig. 5.3. In other words, chromosomes within the system consist of multiple groups (level 1) of nine genes (level 2) of sixteen bits (level 3). However, since the design

system only requires that the number of groups of nine genes is variable, it was decided not to permit the number of bits per gene to be variable. This means that, within the design system, alleles consist of a fixed number of sixteen bits (as described earlier), and not of single bits. Hence, every chromosome within the evolutionary design system uses a reduced semantic hierarchy of two levels, see fig. 5.4.

The hierarchy of meaning of the chromosomes can be quickly constructed, no matter how the alleles are stored in memory. This allows hierarchical crossover to efficiently find and compare alleles and groups of alleles during crossover, in chromosomes organised in any way. However, by actually storing alleles in a hierarchy that matches their semantic hierarchy, there is no need to construct it, meaning a saving of computation time. Moreover, a hierarchical organisation also saves memory (by not duplicating identifiers for alleles), and allows the crossover process itself to be speeded up (Bentley & Wakefield, 1996d).

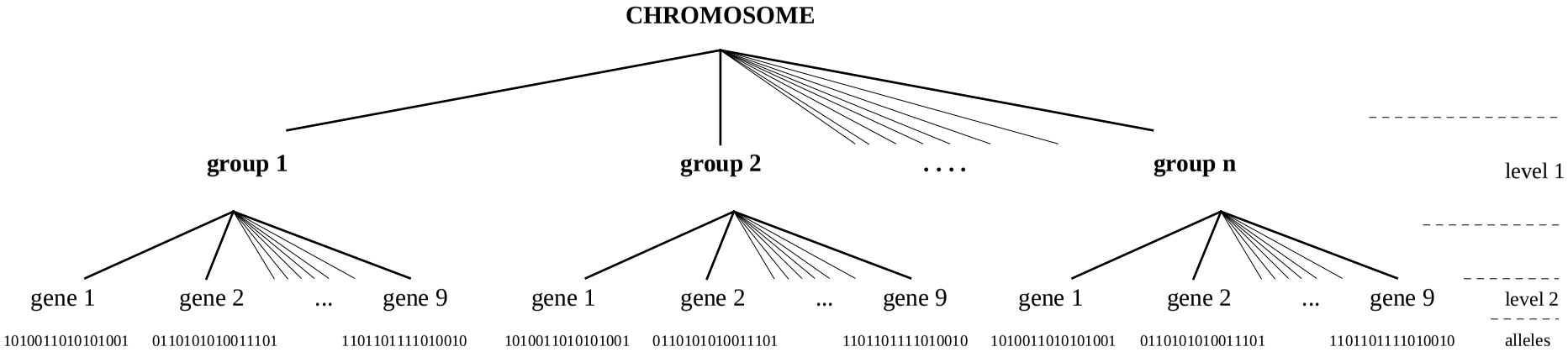

Fig 5.4 Two-level semantic hierarchy of a chromosome in the evolutionary design system.

Hence, the genotype of an individual solution in the population consists of a single chromosome, stored as a tree of depth two, with leaves consisting of 16-bit alleles. This tree is stored internally as a collection of nodes linked by pointers, see fig. 5.5.

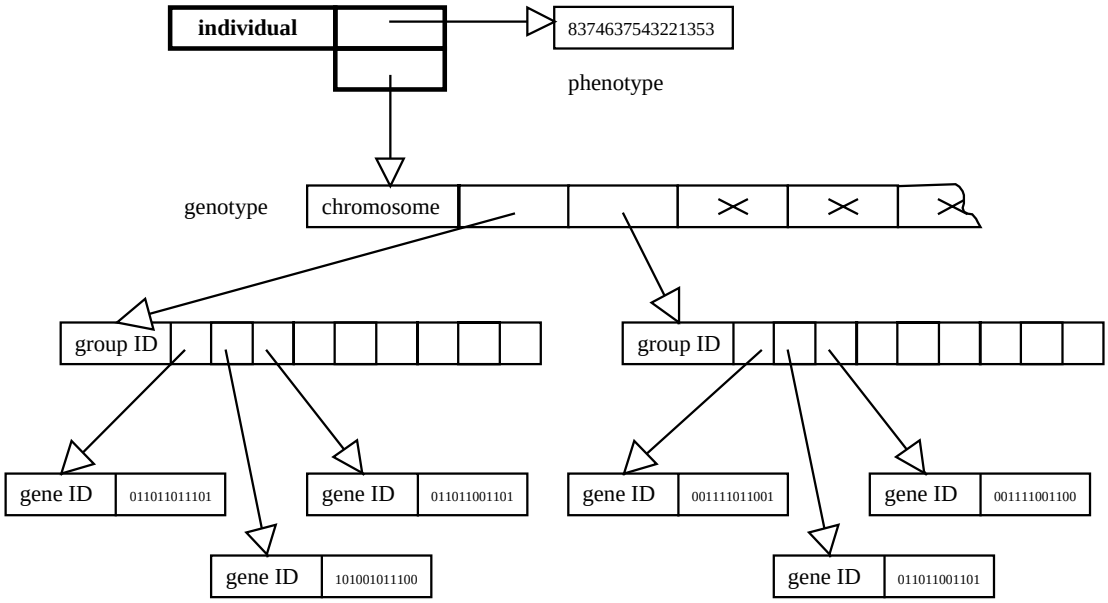

Fig. 5.5 Internal storage of a chromosome (not all alleles shown).

As shown above, chromosomes comprise an array of pointers to groups of nine pointers to alleles. The top-level array can be as large as the available memory permits, but is usually limited to 32 to help reduce execution times. This also limits the number of primitives in a phenotype to 32, assuming that symmetry has not been specified. The implementation allows the number of groups of genes (i.e. primitives in the phenotype) to be variable, and also permits the number of genes per group to be variable. However, because of time constraints, this latter feature is not investigated in this thesis. Hence, although the system can cope with more than one allele per gene (overspecification) and missing alleles for genes (underspecification) during evolution, for all the experiments performed, the number of genes per group was fixed at nine.

Consequently, as can be seen by comparing figures 5.4 and 5.5, the chromosome of an individual is internally organised in the same way as its semantic hierarchy, i.e. every allele is associated with one gene, which is associated with one group in the chromosome. This

hierarchical structure of the genotype allows the new genetic operator known as hierarchical crossover to create meaningful new chromosomes from two compatible parents' chromosomes.

5.3 Genetic Operators

5.3.1 Hierarchical Crossover

As described earlier, hierarchical crossover was developed as part of this work in order to ensure that compatible and equivalent groups of alleles are always crossed, and also to take advantage of the hierarchical semantics and organisations of chromosomes. Like all crossover operators, hierarchical crossover creates new chromosomes from fragments of existing chromosomes. In other words, new solutions to the problem are generated using random parts of existing solutions, allowing a thorough, but fast, parallelised search for good solutions to take place (Goldberg, 1989). The evolutionary design system uses hierarchical crossover to generate all new chromosomes, i.e. crossover is used with a probability of 100%.

Hierarchical crossover is based upon the same principles as the normal single-point crossover outlined in Chapter Two. It consists of a two-stage process: first, find a suitable crossover point within the two parents, and second, perform the crossover to generate two children. Of course, for the first stage, standard crossover simply picks a random position. However, when dealing with two chromosomes with potentially different of genes in a group, or groups in a chromosome, finding a suitable crossover point is more of a challenge. This problem is overcome by using the concept of a semantic hierarchy to define levels of meaning for the chromosome, allowing hierarchical crossover very quickly to traverse both chromosomes in order to locate points of similarity.

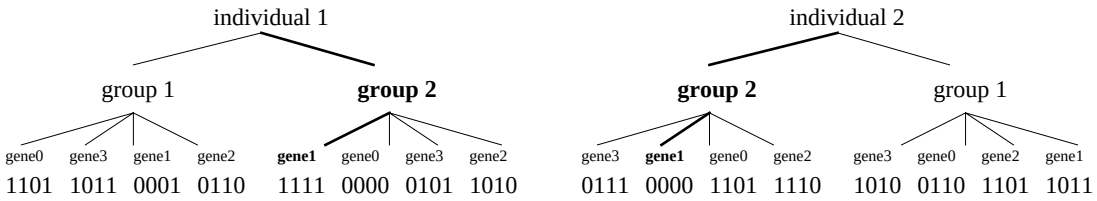

Fig. 5.6 Locating points of similarity (POS) between two parents (shown in bold).

Briefly, starting at the top level of the semantic hierarchy for each chromosome, a random node is picked from individual 1, and a corresponding node is selected from individual 2. The algorithm then traverses down these nodes, and two more nodes that correspond with each other are picked in the same way, and so on, until the leaves of the trees are reached. If at any stage, there is no corresponding node in individual 2, an alternative is randomly picked in individual 1 and a new corresponding node is searched for in individual 2. If there is still no success when all alternatives at that level have been considered, the algorithm backtracks up a level, picking a new alternative node in individual 1 at the higher level. If the algorithm backtracks right to the root, then there are no points of similarity between the two chromosomes and the crossover operation is aborted. Figure 5.6 shows the results of this process, with the points of similarity (POS) between the two individuals being randomly selected as: <group 2, gene 1>.

Once the POS (or hierarchical crossover point) has been established, the actual crossover process can begin. Again, the algorithm starts at the root of each individual. The first top-level node of meaning is then picked from individual 1, in the order of which the nodes are stored. By picking nodes in a specific order, overspecified genes can have their values determined, i.e. always use the first allele found for each gene (Bentley & Wakefield, 1996d). Next, the corresponding node is found in individual 2 (if it exists). If these nodes are *not* the same as any point of similarity found previously, they are copied (or simply moved) from both parents to the offspring. Exactly which child receives a node from which parent can be determined randomly or by comparing the node identifier with the POS node of that level in the hierarchy, to see if the current node is 'before' or 'after' the POS node. If nodes are copied randomly, this crossover will mix alleles and groups of alleles from the two parents randomly (in the same manner as Syswerda's uniform crossover; Syswerda, 1989). If nodes are copied using the order of the identifiers, the crossover will behave in an equivalent manner to standard single point crossover (Bentley & Wakefield, 1996d).

It should be noted that these 'nodes' are really abstract meanings encompassing anything from one allele to groups of alleles within the design system. Hence, when a node is copied from a parent to a child, in reality, all alleles that are defined by the node are copied. Put another way, if a group node was being copied from a parent to a child, then all genes in that group and all alleles for those genes would be copied together. Indeed, this is one of the features of hierarchical crossover that can speed up the process, for when chromosomes are internally stored in the same hierarchy as their semantic hierarchy, a parent's node can be literally *moved* to the child. In this way the majority of the crossover process can be achieved by simply changing one or two pointer values, and rather attractively, children become literally composed of their parents genes. This in itself is a highly useful property, as it means that memory need not be continuously allocated and destroyed - the two children completely re-use the memory taken up by their parents in the computer. Having said this, the evolutionary design system does not move nodes from parents to children, it simply copies the information. This is because the GA used in the system does not replace parents with children, so the parents must not be destroyed by the reproduction process.

Returning to the algorithm once more, the copying process continues until all the parents' nodes at the current level have been copied or moved, except the node listed as a point of similarity. Figure 5.7 shows the slightly denuded parents from fig. 5.6 and the partially formed children at this point, with group 2 being the current POS (with the nodes in this example being moved rather than copied, from parents to children).

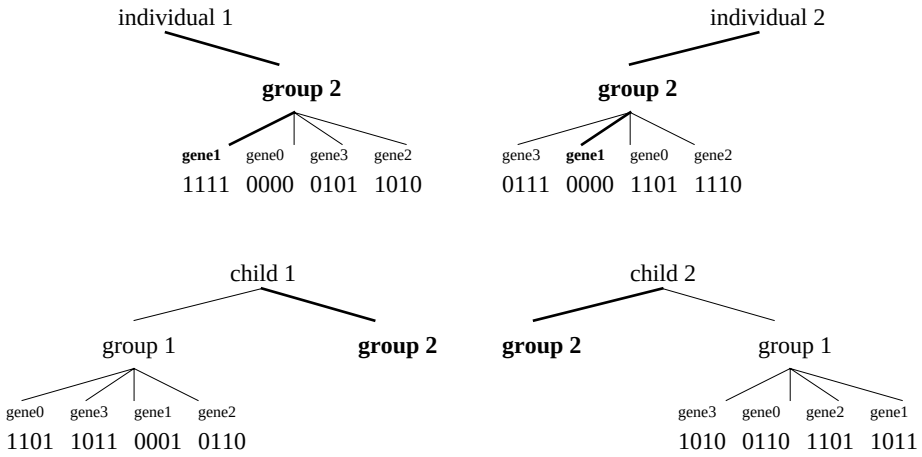

Fig. 5.7 Partially formed children, halfway through hierarchical crossover.

The algorithm then traverses down both points of similarity in both parents, and repeats the same copying process with the nodes at this level, again omitting the POS (gene 1 in fig. 5.7). The algorithm traverses down again, or if it has reached a leaf, as in the example of Fig. 5.7, the last two remaining alleles of the parents are randomly crossed over using standard crossover to generate new alleles for both children. Figure 5.8 shows how the two children look after the completion of hierarchical crossover.

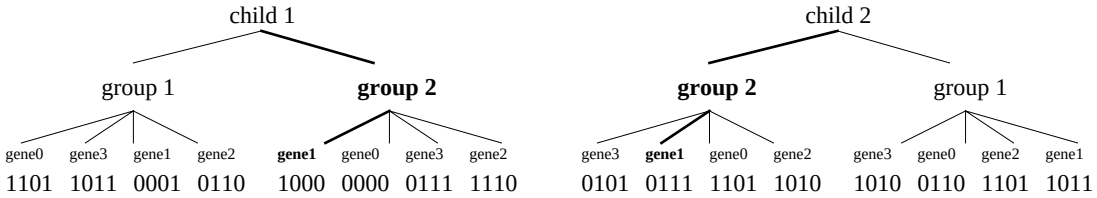

Fig. 5.8 Children produced by hierarchical crossover (from parents shown in fig. 5.6).

Upon careful examination of fig. 5.8 (taking account of the fact that the genes and groups are stored out of order in this example), it should be clear that hierarchical crossover has behaved exactly as normal crossover would (with a crossover point of 21) in this case. However, unlike standard crossover, hierarchical crossover can also deal with variable numbers of anything, in each chromosome.

So far, only an example of crossover between parents with chromosomes of identical lengths has been given. Chromosomes of different numbers of nodes (e.g. groups of alleles) are detected by hierarchical crossover when corresponding nodes are being searched for (during the copying stage) in the second individual. For example, if individual 1 has three genes: A, B, C and individual 2 has only two: A, C, the algorithm will find no corresponding gene 'B' in individual 2, hence one child will receive gene 'B' of individual 1, and the other child will not receive a gene. Equally, the algorithm checks for the reverse: if individual 1 has two genes: A, B and individual 2 has an extra gene: A, B, X, then one child receives gene 'X', and the other child receives no gene. To illustrate further, consider two parents, one with a three group, nine gene semantic hierarchy, the other identical except that it is missing group 2 (the semantics are shown by brackets and position).

Parent 1: `((abcdefghi) (jklmnopqr) (stuvwxyzα))`

Parent 2: `((ABCDEFGHI) (STUVWXYZA))`

For this example, a random point of similarity (POS) was chosen as: < group 1, gene 5 >
(i.e., the allele `e` in parent 1 and `E` in parent 2). If a random choice was made to determine which node is copied to each child, the crossover would produce children similar to:

Child 1: `((AbcDefgHi) (stuvwxyzα))`

Child 2: `((aBCdEFGhI) (jklmnopqr) (STUVWXYZA))`

It should be clear that, unlike conventional crossover, this crossover operator has not created any overspecification or underspecification of genes within the children. In other words, as long as the parents are meaningful, hierarchical crossover will always ensure that the children are meaningful. Although Parent 2 was missing group 2, this would never cause hierarchical crossover to corrupt the offspring. The fact that no corresponding group 2 can be found in Parent 2, prevents hierarchical crossover from selecting group 2 as a POS, and hence prevents a damaging cross (Bentley & Wakefield, 1996d).

The implementation of hierarchical crossover in the evolutionary design system uses random choice to determine which child receives a node from which parent. As will be shown by the evolved designs in Chapter Eight, this uniform method (Syswerda, 1989) has been found to be very effective during evolution.

In conclusion, hierarchical crossover solves the problem of generating new chromosomes from parent chromosomes of different lengths. Using the new concept of a semantic hierarchy to allow the efficient traversal of chromosomes, points of similarity between the two parents are located. The crossover operator then passes compatible and equivalent groups of alleles and alleles to the two offspring, ensuring that the meaning of the chromosomes is maintained. The full algorithm for hierarchical crossover is given in Appendix A.

5.3.2 Mutation

In addition to the use of crossover to generate new offspring from existing solutions in the population, the design system also employs mutation. Unlike crossover, which can be thought of as 'jumping' to a new solution that is related to two existing solutions in the design space, mutation is used to allow a GA to change alleles slightly in order to explore new solutions close to a current solution in the space. In the final stages of evolution using GAs (and in nature), when populations of solutions have usually converged to become almost identical, mutation is vital to allow the solutions to be slowly fine-tuned.

The system uses two forms of mutation: mutation of single alleles and mutation of groups of alleles. The first type of mutation, in which a single allele is mutated, is performed by the following algorithm:

randomly pick a group of genes in the hierarchical chromosome

nine times out of ten:

    randomly pick one of the nine genes in the group

    locate the corresponding allele

    if the allele is not fixed then flip a random bit of the allele

one time out of ten:

    locate the two alleles for the genes: angle1 and angle2

    if the alleles are not fixed then flip a random bit in each allele

Hence, every bit in every allele in every group of alleles has the same probability of being flipped (i.e. changed from '1' to '0', or from '0' to '1'), except the bits in the alleles for the genes: angle1 and angle2. This is because the alleles for these two genes define the parameter values of the two angles that specify the orientation of the clipping plane for a primitive (see Chapter Four). Changing the value of angle1 alters the orientation in the X-Y plane, and changing the value of angle2 alters the orientation in the X-Z plane. However, changing the value of both angle1 and angle2 together, alters the orientation in the Y-Z plane. So by allowing the values for both angles to be mutated together, in effect a tenth parameter value (i.e. a virtual 'angle3') is being changed. This is why mutation modifies one of the nine alleles in a group nine times out of ten, and modifies alleles for both angle1 and angle2, one time out of ten.

The probability of a single binary digit in an allele mutating is set by the user of the design system. Typical probabilities used by other researchers range from 0.001 to 0.01 (Bäck, 1993). The default value of the system is 0.001, although the precise value does not appear critical, and does not seem to have a significant effect on evolution when changed.

The second type of mutation used by the system is mutation of whole groups of alleles (i.e. coded primitives). As was described in Chapter Four, this mutation allows primitives to be added or removed from phenotypes, and consequently is responsible for changing the length of genotypes. The following algorithm is used to perform this type of mutation:

randomly pick a group of genes that are MUTATABLE in the hierarchical chromosome

randomly decide whether to delete or split the coded primitive (50:50 chance)

if deleting :- remove entire group of genes and corresponding alleles from chromosome

if splitting :- randomly decide which direction to split coded primitive

calculate the new values (see section 4.3.4) for the alleles of the current group and for a new group

add the new group of alleles to the chromosome.

The probability of each primitive being split or deleted is also set by the user of the evolutionary design system. The default value used is 0.01, although again, the precise value does not appear to be critical.

5.3.3 Non-Mutatable Alleles and Groups of Alleles

As was described at the end of the previous chapter, the primitives defined in phenotypes can be given a number of 'special properties' such as symmetry, inflexibility, and two-dimensionality. In the same way, the alleles (and groups of alleles) defined in genotypes can also be given 'special properties'. There are two such properties: the allele for a gene can be *fixed*, or a group of alleles (i.e. a coded primitive) can be *not mutatable*.

When the value of a gene is fixed, this means that it cannot be changed by mutation. Usually the value is also specified by the user and used to seed the initial population. Hence, every individual in the population can have predetermined values set for specific genes, which remain unchanged throughout evolution. For example, if every gene 3 of group 2 was given a value of 7 and was fixed, then crossover would not be able to change this value. Since mutation of fixed alleles is forbidden, the value of gene 3 would remain unchanged.

This feature is useful when the system is required to evolve around fixed 'skeletons', or when only part of a design is required to be evolved. Any gene can have its value fixed by the user.

Moreover, when the user specifies that phenotypes should be two-dimensional, all genes for z-position, angle2 and depth are initialised and automatically fixed by the system.

The other 'special property' defines whether a group of alleles (i.e. a coded primitive) in a chromosome can be deleted or split into two by the second type of mutation. If a coded primitive is 'not mutatable', then it cannot be affected by this mutation operator (however, the mutation of individual alleles remains unaffected). This characteristic is automatically given to a coded primitive with any fixed alleles, and to any coded primitive which has been defined as being 'inflexible'. The reason for preventing coded primitives with fixed alleles from being altered by mutation is simply that it is impossible to split a primitive with fixed dimensions or position, without changing the fixed values. Since the values must be changed during the splitting process, and fixed values are forbidden from being changed, the mutation is not permitted. Likewise, 'inflexible' primitives are intended to be used to form rigid, 'unsquashable' skeletons, around which designs are to be evolved, so splitting or deleting them is undesirable.

5.4 Summary

Genetic algorithms do not alter the values of parameters directly, instead they manipulate the alleles of genes which are then decoded to give parameter values. This chapter has described the coding and genetic operators used within the generic evolutionary design system to allow the GA at the core of the system to alter the shape of designs.

Every design, or phenotype, is decoded from a corresponding genotype. Every genotype within the system consists of a single chromosome, which in turn comprises a variable number of groups of nine alleles. Because low-cardinality genetic representations theoretically make GAs better able to converge to good solutions (Holland 1975, Goldberg 1989), binary coding was used, with alleles being stored as 16-bit sign-and-magnitude binary numbers with a range of -255.992 to 255.992.

Using the position of alleles within a chromosome to determine which allele belongs with which gene can cause standard crossover to produce meaningless or corrupted offspring from parents with chromosomes of different lengths. However, using 'allelic coding' (Radcliffe & George, 1993) where every allele is associated with identifiers, can waste memory because of unnecessary duplicated identifiers. To overcome these problems, the concept of a semantic hierarchy (tree of meaning) for a chromosome was introduced. This allows the meaning of alleles within a chromosome to be efficiently identified. Moreover, to reduce the memory needed to store genetic identifiers for each allele, chromosomes were internally organised in hierarchical tree-structures matching their semantic hierarchies.

These hierarchical chromosomes are manipulated by a new crossover operator created for this work, known as hierarchical crossover. This genetic operator uses the concept of a semantic hierarchy (and the fact that chromosomes are internally stored as hierarchies) to ensure that compatible fragments of chromosomes are always combined, and thus ensures that offspring are always meaningful.

Finally, the two mutation operators used by the system were described. These genetic operators randomly modify single alleles, or groups of alleles. However, the design system does allow the user to define alleles as being 'fixed', or groups of alleles as being 'not mutatable', in which case these operators are disabled.