6

The Genetic Algorithm

6.1 Introduction

The genetic algorithm is the third and most significant element of the generic evolutionary design system. This algorithm forms the core of the system, modifying alleles within chromosomes using the crossover and mutation operators, decoding genotypes to produce phenotypes, and identifying the fittest designs. The GA is the 'designer' in the system, automatically performing the entire design process by creating new conceptual designs and optimising those designs, guided by evaluation software.

As was described in Chapter Two, there are a number of types of advanced GAs, all developed to improve the abilities of evolutionary search for different types of problem. Because the problem of evolving a wide range of different solid objects from scratch is so difficult, it was necessary to introduce new 'exotic' techniques to help tackle these problems (Bentley and Wakefield, 1996c). Hence, the GA used within the system has a number of advanced features, carefully introduced during the development of the system, in response to various types of design problem. Such features include: a simple form of artificial embryology, where simple genotypes are explicitly mapped to more complex phenotypes, multiobjective optimisation, and overlapping populations.

These features all enable the evolutionary design system to successfully evolve a number of different solid object designs from scratch (Bentley and Wakefield, 1996c). However, before all of the various aspects of the system are explained, the reason why the genetic algorithm was thought to be the most suitable search algorithm for the system must be explained.

6.2 Justification of Choice

The genetic algorithm is just one of many search algorithms known in Computer Science. It is currently not possible to define exactly which of these search algorithms is best for which problem or even class of problems (Fogel, 1995). However, it is possible to identify algorithms that consistently produce improved results (compared to results produced by other techniques) for a wide range of different problems. Indeed, as was explained in detail in Chapter Two, the GA is such an algorithm, having been successfully demonstrated with hundreds of different types of problem (Holland, 1992).

Correspondingly, the evolutionary design system is intended to be generic, i.e. capable of creating good solutions to a wide variety of solid object design problems. Since there must be no limit on the type of functions that can be used to form the evaluation software, the search algorithm at the core of the system must ideally be able to find good solutions to literally any type of function. Consequently, this system demands a highly robust search algorithm - such as the genetic algorithm (Goldberg, 1989).

Although such a design system has not been demonstrated before, computers have been used to create simple conceptual designs, optimise existing designs, create artistic images and generate the shapes of 'virtual creatures'. As was shown in Chapter Two, the most common search algorithm used for all of these related systems, was the genetic algorithm. This makes the GA unique: it is the *only* type of search algorithm to have been successfully used in every type of system related to the generic evolutionary design system described in this thesis.

Moreover, some researchers claim that natural evolution and the human design process are directly comparable (Goldberg 1991b, French 1994). Indeed, Goldberg actually attempts to formally define human design in terms of evolution by the genetic algorithm (Goldberg, 1991b). Since the generic evolutionary design system is intended to automate the design process, the use of the genetic algorithm would seem appropriate in this respect, also.

Finally, although a generic automated computer design creation system has not been demonstrated before, natural evolution has been successfully creating designs for millennia (French, 1994). Indeed, conceivably the most complex and remarkable miracle of design ever created - the human brain - was generated by evolution in nature (Dawkins, 1995). This is perhaps the most conclusive demonstration of all, that evolution is the most suitable way to create new designs from scratch. Since the genetic algorithm is the closest analogue to natural evolution known in Computer Science, the use of the GA in the evolutionary design system would seem wholly justified.

To summarise, it is felt that the genetic algorithm is the most suitable form of search algorithm to use as the core of the generic evolutionary design system because:

- An algorithm capable of generating good solutions to a wide range of problems is required, and the GA is such an algorithm.
- Only the GA has been successfully used in every related type of design system.
- The human design process is comparable to and has been formally defined in terms of the GA.
- The most successful and remarkable designs known to mankind were created by natural evolution, and the GA is the closest analogue to natural evolution in Computer Science.

6.3 Beginning the Evolution of Designs

6.3.1 Population Data Structures

Before the genetic algorithm at the core of the design system can begin evolving new designs, the system must create and fill the population data structures. Populations within the GA comprise collections of 'individuals', where each individual consists of a genotype, phenotype and other related data corresponding to a single design solution. As shown by fig. 6.1, the GA uses two separate populations of individuals, named the 'internal' and 'external' populations, which are held in memory as two linked lists.

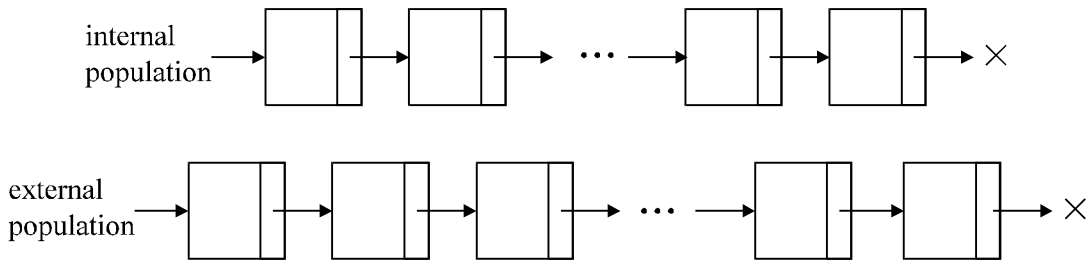

Fig. 6.1 Populations of individuals in the GA.

The internal population consists of an unordered collection of 'unborn' individuals, i.e. having freshly generated genotypes that have yet to be mapped to phenotypes and evaluated. The external population consists of individuals that have been 'born', i.e. having genotypes with corresponding phenotypes, which have been evaluated. Individuals in the external population are held in the order of their overall fitness, with the fittest at the front of the list, and the weakest at the back. Although the maximum number of individuals permitted in each population is limited, linked lists were used instead of arrays because they allow individuals to be moved or inserted very efficiently (i.e. simply by changing the value of two or three pointers).

As will be discussed later in this chapter, every generation, all individuals in the internal population are placed into the external population (removing the weakest individuals from the external population to make room). Since the internal population is normally smaller than the external one, this allows some very fit members of the external population to remain, or

'overlap' across a number of subsequent external populations, i.e. the GA allows overlapping populations, see fig. 6.2. Typical sizes for the internal and external populations used to successfully evolve designs are 160 and 200 individuals, respectively.

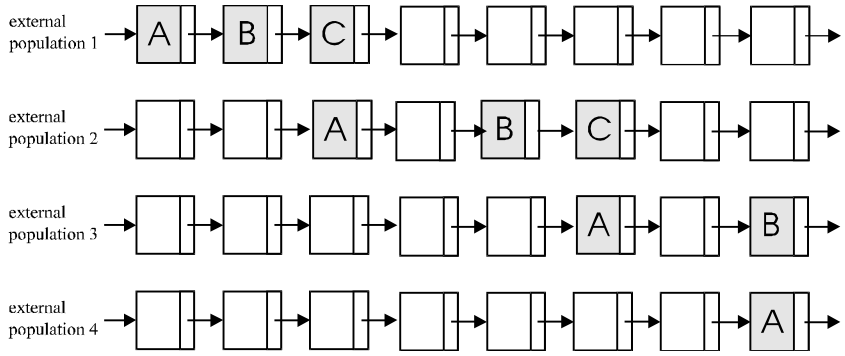

Fig. 6.2 Fit individuals can 'overlap' a number of subsequent external populations.

All individuals in the two populations have a number of separate areas in which data is stored. Every individual has a variable-sized tree-structured genotype, and a corresponding 'flat' list of decoded parameter values forming the phenotype. In addition to these, all individuals have two values specifying the number of groups of nine genes in the genotype, the number of parameters in the phenotype, and a value denoting their age (i.e. the number of generations that have passed since they were placed in the external population). Finally, individuals have a list of their fitness values for the current design problem, and a pointer to the next individual in the population linked list, see fig. 6.3.

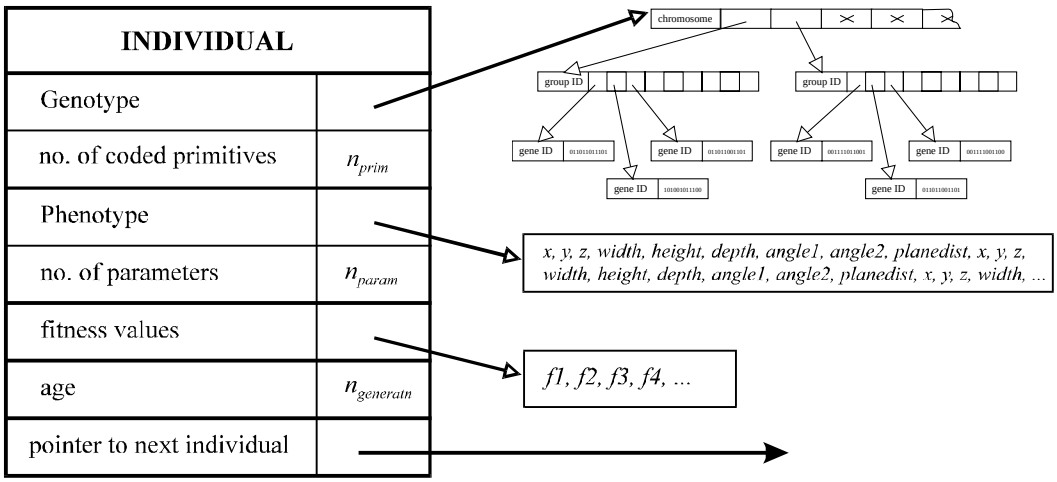

Fig. 6.3 Single individual in a population.

6.3.2 Seeding the Initial Population

Once the data structures have been allocated in memory, the GA is initialised by providing alleles for the genes of each individual in the internal population. By default, the system will initialise the genotype of every individual with random alleles for a user-defined number of primitives. When decoded, these entirely random genotypes correspond to phenotypes consisting of free-form shapes, from which the final designs must be evolved.

It is standard practice to initialise GAs with random alleles (Goldberg, 1989), however it is unusual for the corresponding phenotypes to be quite so different and remote from the required solution to the problem. For example, a typical design optimisation system will only manipulate a few selected parameters of an existing design (Parmee, 1994). Although the corresponding genes would initially be given random alleles, because the parameters only define small sections of the design that can be changed, many of the random initial designs would resemble final, acceptable designs relatively closely. However, because the generic evolutionary design system uses a spatial partitioning representation that allows genotypes to define the shape of every part of designs, the random initial designs of the system are truly random, and often do not resemble anything at all.

Although such a varied initial population of designs inevitably makes the task of evolving a design with a specific shape very difficult for the GA, this variety is vital to give the GA complete freedom to generate any shape of design that will satisfy the design criteria. In other words, by initialising the GA with random, free-form 'blobs', the GA can potentially create new and unconventional conceptual designs, from scratch.

Moreover, if desired, genotypes can be initialised with a random number of coded primitives (groups of nine alleles), which introduces another level of variability to the first population of individuals. If genotypes are initialised in this way, not only is the initial shape of designs random, but the number of primitives that help to define the shape of designs is random, i.e. designs have variable shapes and variable amounts of detail in each shape. Hence, the initial

population helps evolution to be unbounded, and can permit the GA to explore a number of different hyperspaces of solutions (Sims, 1994b).

The generic evolutionary design system does permit the user to intervene during the initialisation process. As will be fully explained at the end of Chapter Seven, the system automatically reads an initialisation file prior to initialising the GA. This file allows the user to directly specify the value of any gene, the number of primitives in designs, whether designs should be symmetrical and other special features, as well as defining which modules of evaluation software to use and specifying global parameters such as population sizes and mutation rates. In this way, should it be required, the GA can be used to optimise existing designs, parts of designs, assemble components of designs, or to design around fixed 'skeletons'. Nevertheless, the emphasis of this thesis is on the evolution of solid object designs from scratch, so the GA is usually initialised with completely random alleles.

Once individuals in the internal population have been initialised (regardless of the method used), the individuals are evaluated and then placed directly into the empty external population. However, before any individual can be evaluated, its genotype must be mapped to generate the corresponding phenotype.

6.4 Mapping Genotypes to Phenotypes

6.4.1 Explicit Mapping

Many researchers seem to overlook the mapping stage from genotypes to phenotypes in their implementations of genetic algorithms. It is not uncommon to see GAs that evaluate genotypes directly, e.g. Goldberg's simple GA (Goldberg, 1989). However, biologists have known for some time that the process of generating a phenotype from the 'instructions' held within a genotype, is perhaps the most significant element of life (Dawkins, 1995). This remarkable process of embryology allows a single molecule of DNA in a zygote to develop into a hugely more complex animal. It is then the capabilities of these animals to survive and reproduce that

determines whether the genes - that specified how they should be created - will be passed on. In other words, nature uses a highly efficient method of mapping genotypes to phenotypes of greater complexity. Nature 'evaluates' phenotypes, not genotypes.

Some researchers have now discovered the benefits of an artificial embryology stage within the genetic algorithm. As was described in Chapter Four, this idea has been used to allow compact shape grammars to give instructions on how complex shapes should be 'grown' during the mapping from genotypes to phenotypes (Todd and Latham 1992, Sims 1994a, 1994b, Rosenman 1996). Schaffer has noted the value of explicitly distinguishing between genotypes and phenotypes, using problem-specific algorithms to map genotypes to phenotypes, in order to enhance evolution of good solutions for combinatorial problems (Schaffer & Eshelman, 1995).

The generic evolutionary design system also uses an explicit mapping stage between genotypes and phenotypes. As has been indicated in previous chapters, this stage is used in order to ensure that legal designs (i.e. designs with no overlapping primitive shapes) are always defined by genotypes. This simple artificial embryology process also allows partial designs to be reflected to form symmetrical designs, as well as permitting other special features, such as no intersecting planes, inflexible primitives and so on. This means that the genotype of individuals does not directly specify the shape of the phenotype, i.e. just as in nature, changing the allele of a gene may have an indirect effect on a phenotype, or even no effect at all. Nevertheless, evolution of good designs does not appear to be harmed by this indirect relationship. In a similar way to natural evolution, the GA in the design system compensates (Bentley & Wakefield, 1995a/1996a).

6.4.2 Stages of the Mapping Process

There are four stages in the process that maps the hierarchically structured alleles in a genotype to a list of parameter values in the corresponding phenotype:

1. Extract and decode an allele for every gene in the genotype.
2. If necessary, amend some values.
3. Correct the design.
4. If required, reflect and further correct the design.

First, the alleles within the genotype must be decoded from 16-bit sign-and-magnitude binary numbers to decimal parameter values. Since the number of coded primitives is stored as a separate value within each individual (see fig. 6.3), the number of parameter values in the (unreflected) phenotype must be nine times this value (nine parameters define a primitive). Hence, the alleles for each of these primitives are searched for and located in the genotype, then decoded and placed in order, in the phenotype. Should a gene be overspecified (i.e. more than one allele exists for that coded parameter), the first allele found is decoded and used. Should a gene be underspecified (i.e. no allele found for the coded parameter), a global default value is used. However, as mentioned previously, for all of the tests performed with the evolutionary design system, the number of alleles per group of alleles was not varied, so a single allele would always exist for every gene. Consequently, at the end of the first stage of the mapping process, the phenotype consists of $n_{prim}\times 9$ ordered decimal parameter values, corresponding to the $n_{prim}\times 9$ hierarchically structured alleles in the genotype (where n_{prim} is the number of coded primitives in the genotype).

In the second stage of the mapping process, specific parameter values in the partially-formed phenotype are examined. If the values for *width*, *height*, or *depth* of any primitive are negative, they are made positive (this must be explicitly done or the 'squashing' algorithm used to correct illegal designs will not work correctly). If the value for *planedist* of any primitive equals zero, it is set to a positive number of negligible size (as was explained in Chapter Four, if the distance from the centre of the primitive to the clipping plane is exactly zero, its orientation cannot be defined). Finally, if the user has requested that primitives should have no clipping planes associated with them, the *planedist* value for every primitive is set to a very high value

(i.e. every clipping plane is moved far enough away from its primitive to prevent it from ever intersecting the primitive).

The third stage of the mapping process compares every primitive in the phenotype with every other one. If any two primitives are detected as overlapping, they are 'squashed' until they touch (see Section 4.3.2). This process further amends the parameter values in the phenotype. Finally, the fourth stage of the mapping process reflects the phenotype in the plane $x = 0$, $y = 0$ or $z = 0$ if required by the user (anything from zero to three reflections can take place). For each reflection performed, the number of parameters in the phenotype doubles, and all new primitives in the phenotype are compared with all existing primitives. If any are found to overlap, they are corrected in a symmetrical manner (see Section 4.3.3). Once completed, the number of parameters in the phenotype is stored as a separate value in the individual (n_{param}).

By the end of this mapping process, a legal and possibly symmetrical phenotype that corresponds to the genotype has been created in the individual. The process guarantees that the phenotype will have non-overlapping primitives, without modifying alleles within the genotype, and so without restricting evolution. Moreover, because of symmetrical reflections, the number of parameter values in the phenotype can be substantially greater than the number of alleles in the genotype, meaning that this simple artificial embryology process does enable genotypes to specify phenotypes of greater complexity, in a similar way to nature.

6.5 Determining the Relative Fitness of a Design

6.5.1 Problems with Multiple Fitness Values

Once the phenotypes of all the individuals in the internal population have been generated by the mapping process, they are passed to the evaluation software. These user-selected modular 'fitness functions' first calculate appropriate additional information about the phenotypes, and then use this information to calculate how well each phenotype satisfies particular criteria (see Chapter Seven). The evaluation software modules then return a number of fitness values for

each phenotype, which together specify exactly how well the phenotype performs the desired function of the required design. Unfortunately, these multiple values cause a major problem, for the standard GA cannot cope with more than one fitness value per phenotype (Goldberg, 1989).

The problem arises when the GA tries to determine the overall relative fitnesses of phenotypes. With single objective problems, the genetic algorithm stores a single fitness value for every solution in the current population of solutions. This value denotes how well its corresponding solution satisfies the objective of the problem. By allocating the fitter members of the population a higher chance of producing more offspring than the less fit members (and killing the less fit members), the GA can create the next generation of (hopefully better) solutions. However, with multiobjective problems, every solution has a number of fitness values, one for each objective. This presents a problem in judging the overall fitness of the solutions. For example, one solution could have excellent fitness values for some objectives and poor values for other objectives, whilst another solution could have average fitness values for all of the objectives. The question arises: which of the two solutions is the fittest? This is a major problem, for if there is no clear way to compare the quality of different solutions, then there can be no clear way for the GA to allocate more offspring to the fitter solutions.

The concept of Pareto-optimality helps to overcome this problem of comparing solutions with multiple fitness values. A solution is Pareto-optimal (i.e., Pareto-minimal, in the Pareto-optimal range, or on the Pareto front) if it is *not dominated* by any other solution. The vector **x** *dominates* **y** when **x** is *partially less* than **y**. As stated by Goldberg, a vector **x** is partially less than **y**, or $\mathbf{x} <_p \mathbf{y}$ when: $(\mathbf{x} <_p \mathbf{y}) \Leftrightarrow (\forall_i)(\mathbf{x}_i <= \mathbf{y}_i) \wedge (\exists_i)(\mathbf{x}_i < \mathbf{y}_i)$ (Goldberg, 1989).

However, it is quite common for a large number of solutions to a problem to be Pareto-optimal (and thus be given equal fitness scores). Moreover, since a solution can be Pareto-optimal if it only has a single good fitness value, for the evolutionary design system, not all Pareto-optimal solutions are acceptable designs (Bentley and Wakefield, 1996f).

Existing literature seems to approach this fitness ranking problem using methods that can be classified in one of three ways: the aggregating approaches, the non-Pareto approaches and the Pareto approaches. Many examples of aggregation approaches exist, from simple 'weighting and summing' (Syswerda and Palmucci, 1991; Goldberg 1989) to 'multiple attribute utility analysis' (MAUA) (Horn and Nafpliotis, 1993). Of the non-Pareto approaches, perhaps the most well-known is Schaffer's VEGA (Schaffer 1984, 1985), who (as identified by Fonseca and Fleming, 1995a) does not *directly* make use of the actual definition of Pareto-optimality. Many other non-Pareto methods have been proposed (Husbands 1994, Linkens and Nyongesa, 1993; Ryan 1994; Sun and Wang, 1992). Finally the Pareto-based methods, proposed first by Goldberg (1989) have been explored by researchers such as Horn and Nafpliotis (1993) and Srinivas and Deb (1995). In addition, many researchers are now introducing 'species formation' and 'niche induction' in an attempt to allow the uniform sampling of the Pareto set (Goldberg 1989; Horn and Nafpliotis, 1993). For a comprehensive review, see Fonseca and Fleming (1995a).

Early versions of the evolutionary design system used aggregation to solve this problem, i.e. separate fitness values were simply weighted and summed to form a single, overall fitness value for each phenotype (Bentley and Wakefield 1995a/1996a, Bentley and Wakefield 1995b). However, the value of each weight required laborious and careful fine-tuning before the system was able to evolve designs successfully. This was felt to be highly unsatisfactory, for only someone who knew intimately the functions within each module of evaluation software could ever fine-tune the system accurately enough to make it work. Clearly, in order to allow a lay-person to use the system successfully, an alternative approach to aggregation was required for the GA.

6.5.2 Range Independence and Importance

In order to identify the properties that the new multiobjective ranking method should possess, the exact problems faced by a GA trying to combine multiple fitness values into a single overall value, were investigated and analysed in some depth (Bentley and Wakefield, 1996f).

Modules of evaluation software consist of different fitness functions which together produce a number of separate fitness values for a phenotype. Although the ranges of these functions may be infinite in theory, in practice the range of fitness values returned will be finite. This *effective range* of every objective function is determined not only by the function itself, but also by the domain of input values that are produced during evolution. These input values are the phenotype parameter values to be evolved by the GA and their exact values are determined initially by random, and subsequently by evolution. The values are limited still further by the genetic coding used, i.e. all alleles are limited to values between -255.992 to 255.992 (see Section 5.2.1).

In most multiobjective tasks, every separate objective function will have a different effective range (i.e. the function ranges are noncommensurable; Schaffer, 1985). This means that a bad value for one could be a reasonable or even good value for another, see Fig. 6.4. If the results from these two objective functions were simply added to produce a single fitness value for the GA, the function with the largest range would dominate evolution (a poor input value for the objective with the larger range makes the overall value much worse than a poor value for the objective with the smaller range).

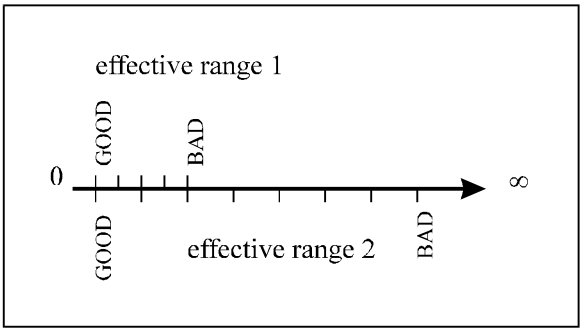

Fig. 6.4 Different effective ranges for different objective functions (to be minimised).

Thus, the only way to ensure that all objectives in a multiobjective problem are treated equally by the GA is to ensure that all the effective ranges of the objective functions are the same (i.e. to make all the objective functions commensurable), or alternatively, to ensure that no objective is directly compared to another. In other words, either the effective ranges must be converted to make them equal, and a *range-dependent* ranking method used, or a *range-independent* ranking method must be used (Bentley and Wakefield, 1996f).

The other new concept introduced by this research was *importance* (Bentley and Wakefield, 1996f). Many researchers have independently noted that with highly complex search problems, searching efficiency can be increased, and time can be reduced, by increasing the importance of a particular part, or objective(s) of that problem (Dowsland, 1995; Marett and Wright, 1995). Moreover, some problems (such as many of the design problems presented to the evolutionary design system) require that certain objectives have differing levels of importance just to allow evolution of an acceptable solution.

For example, consider the problem of packing a bag before going mountaineering. In this simplified example, the person has to choose between the amount of climbing equipment and the amount of food to be packed in the bag. How much of each should be packed? Some researchers would state that the two are non-commensurable and cannot be directly compared by a computer, and so would present a human user with a number of alternative solutions to choose from. Clearly, in reality, an acceptable solution depends on the length of time of the trip, and the difficulty of the climb. If the trip is to take two days, and will involve only a hike in some hills, then more food is required than climbing equipment. However, if the climb will involve an hour scaling a vertical cliff, then more climbing equipment is required than food. In other words, a human picks a solution based on the relative *importance* of the two objectives. Moreover, there is no good reason not to specify these relative importance values for the computer, and let the computer pick the *same solution* (without the need for a human to consider potentially hundreds of different Pareto-optimal solutions).

Unfortunately, there is no easy way to increase the importance of one objective in relation to another, without the two objectives being directly compared to each other (Bentley & Wakefield, 1996f). Put another way, whilst it is simple to specify increased importance with a range-dependent method such as 'sum of weighted objectives' (just increase the weights), for a range-independent method such as non-dominated sorting, specifying importance is more complex. (Fonseca forces a kind of importance with his 'preference articulation' method, (Fonseca and Fleming 1995b) but this requires detailed knowledge of the ranges of the functions themselves, involves human interaction, and is not a continuous guide to evolution.)

As will be shown in Chapter Eight, the generic evolutionary design system requires the specification of importance for some types of design problems (e.g. when evolving a penta-prism, it is necessary to increase the relative importance of positioning output light rays, compared to the other criteria; Bentley & Wakefield, 1996e). In addition, since one of the intended capabilities of the system is to evolve designs without human intervention, the GA must evolve a single acceptable design, not a selection of both acceptable and unacceptable designs, from which a human selects a single acceptable design. This rules out the 'exotic' techniques favoured by some researchers, who avoid the problems of generating a single acceptable solution, by using niching and speciation within GAs to evolve a selection of Pareto-optimal solutions (Goldberg, 1989; Horn and Nafpliotis, 1993).

Consequently, through this investigation it was discovered that the GA within the design system required a *range-independent* multiobjective fitness ranking method, so that all objectives would be automatically treated equally, without laborious fine-tuning of weights. In addition, this fitness ranking method should allow the simple specification of *importance*, to allow designs with good values for some objectives to be considered fitter than designs with good values for other objectives. Finally, this ranking method must allow the evolution of a single design that is acceptable to the user, not a large collection of both unacceptable and acceptable designs.

6.5.3 Testing Six Multiobjective Ranking Methods

In an attempt to discover a suitable multiobjective technique for the evolutionary design system, three well-known existing methods were compared with three novel methods developed as part of this work (Bentley and Wakefield, 1996f). The algorithms are given in Appendix A. Briefly, these methods were:

1. Sum of Weighted Objectives (SWO).

This is an aggregation method, simply weighting and summing the separate fitness values into one overall value (Srivinas and Deb, 1995). This method supports importance, but is not range-independent.

2. Non-dominated Sorting (NDS).

This Pareto approach ranks solutions into 'non-dominated' order, with the fittest being the solutions dominated the least by others (i.e. having the fewest solutions partially less than themselves) (Goldberg, 1989). This method does not support importance, but is range-independent.

3. Weighted Maximum Ranking (WMR).

This non-Pareto approach is based upon Schaffer's VEGA (Schaffer 1984, 1985). VEGA forms lists of fitness values of each solution for each objective. The fittest *n* solutions from each list are then extracted, and random pairs are selected for reproduction. This method does not support importance, but is range independent.

4. Weighted Average Ranking (WAR).

The first of the new techniques developed, this non-Pareto method calculates the rank of each solution for each objective separately, then makes the overall fitness of each solution equal to its average rank (Bentley and Wakefield, 1996f). This method partially supports importance and is range-independent.

5. Sum of Weighted Ratios (SWR).

The second of the new techniques, this aggregation based technique converts the fitness values for every objective into ratios, using the best and worst solution in the current population for that objective every generation. They are then summed to provide a single overall fitness value for each solution (Bentley and Wakefield, 1996f). This method fully supports importance and is range-independent.

6. Sum of Weighted Global Ratios (SWGR)

This method is the third of the novel proposed ranking methods for GAs, and is a variation of SWR (method 5). Instead of the separate fitnesses for each objective in every solution being converted to a ratio using the *current* population best and worst values, the *globally* best and worst values are used (Bentley and Wakefield, 1996f). This method also fully supports importance and is range-independent.

These six multiobjective fitness ranking methods were tested with a standard GA on four established test functions, see table 6.1. The distribution of the solutions generated was analysed by running up to 10,000 test runs of the GA for each method with each test function. The ranking methods were also tested with a prototype of the evolutionary design system on an example design task with ten separate criteria. This problem consisted of creating a design of a portable set of steps, suitable for use, say, in a library (Bentley and Wakefield, 1996f). The quality of the solutions was analysed in terms of how many and to what extent the criteria were satisfied, in addition to judging the quality of the designs by eye.

FUNCTION:	DESCRIPTION:
$f_1 = x_1^2 + x_2^2 + x_3^2$	A single-objective, three parameter function with an optimal at (0,0,0).
$f_{21} = x^2$ $f_{22} = (x - 2)^2$	A simple twin-objective single parameter function with a Pareto-optimal range between 0 and 2, and a best compromise solution of 1.
$f_{31} = -x$ where $x <= 1$ $-2 + x$ where $1 < x <= 3$ $4 - x$ where $3< x <= 4$ $-4 + x$ where $4 < x$ $f_{32} = (x - 5)^2$	A twin-objective single parameter function with two disjoint Pareto-optimal ranges: 0 to 2 and 4 to 5. This has a single best compromise solution of 4.5
$f_{41} = 1 - \exp(-(x_1 - 1)^2 - (x_2 + 1)^2)$ $f_{42} = 1 - \exp(-(x_1 + 1)^2 - (x_2 - 1)^2)$	A twin-objective function this time with two parameters. This has a single Pareto-optimal range along the line (-1,1) to (1,-1), but two equal best compromise solutions at the optima of each function: (-1,1) and (1,-1).

Table 6.1 The four test functions used to compare the ranking methods.

6.5.4 Distribution and Quality of Solutions

The distributions obtained after performing the experiments with the four test functions were revealing. Figure 6.5 shows an example set of distributions for the second test function. (The complete results of all tests are given and analysed in the paper: 'An Analysis of Multiobjective Optimisation' in Appendix B.) As fig. 6.5 illustrates, although all of the multiobjective ranking methods allowed the GA to generate nothing but Pareto-optimal solutions, the distribution of these solutions in the Pareto-optimal range was highly distinct for each method. Moreover, the distributions all seemed to follow coherent and predictable patterns that were consistent for all the test functions (Bentley and Wakefield, 1996f). (It should also be noted that this appears to be the first work investigating the distribution of solutions generated by multiobjective ranking methods.)

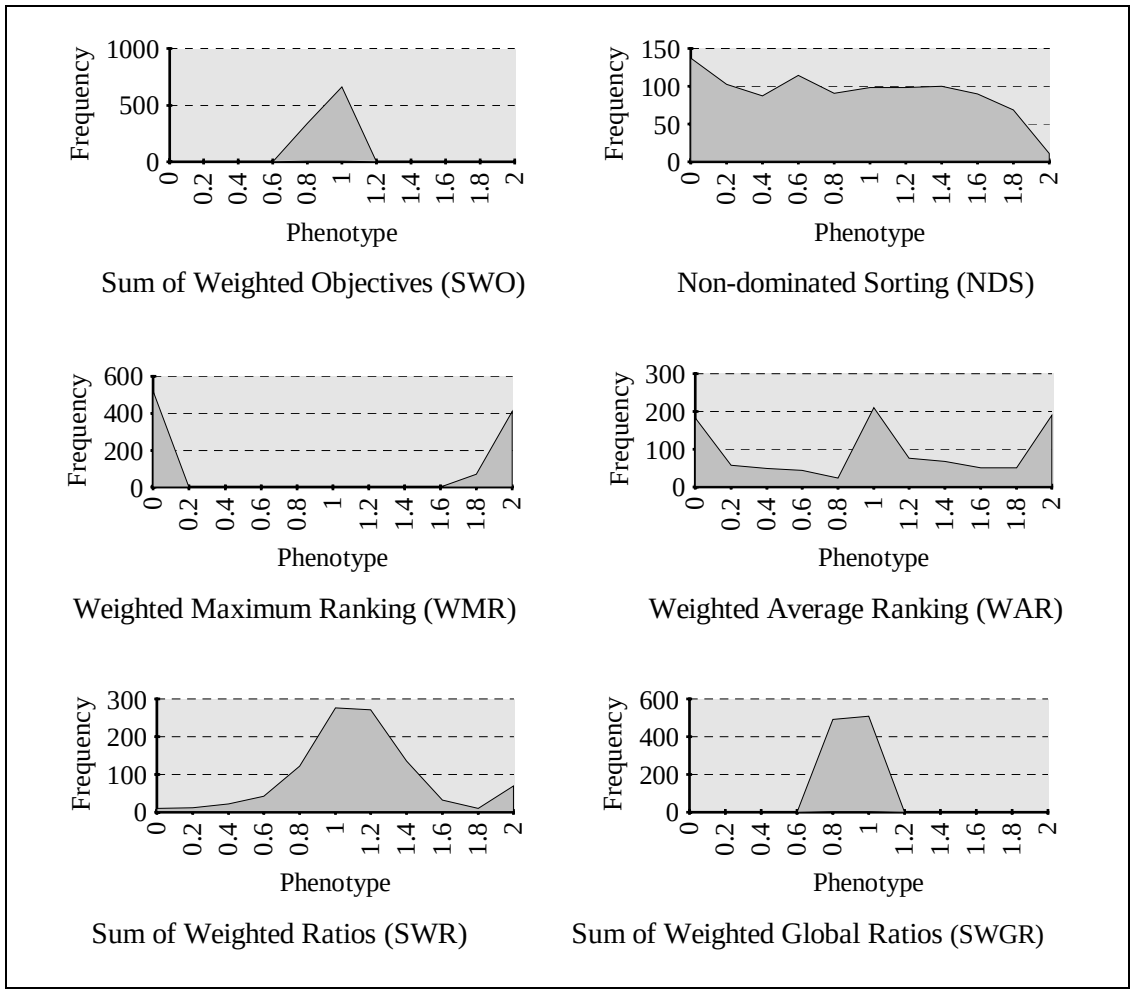

Fig. 6.5 Distributions of final solutions evolved for multiple test runs within the Pareto-optimal range for functions f_{21} & f_{22}.

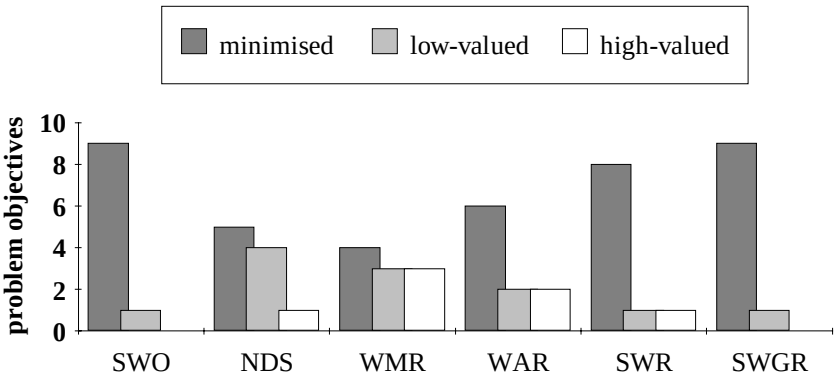

Fig. 6.6 The number of minimised, low and high valued objectives for the best designs of steps produced with each method.

The results obtained after testing each method in the evolutionary design system on the example design problem were also revealing (see Chapter Eight). Many methods produced quite unacceptable designs (despite the fact that they were pareto-optimal). The method that produced the best designs, most consistently, was 'Sum of Weighted Objectives' (after much laborious fine-tuning of weights), and the two new methods 'Sum of Weighted Ratios' and 'Sum of Weighted Global Ratios' (Bentley and Wakefield, 1996f). This is confirmed by the number of criteria that were successfully minimised, and nearly minimised for the best designs produced by each method. As fig. 6.6 shows, 'Sum of Weighted Objectives' and 'Sum of Weighted Global Ratios' both minimised nine out of ten criteria, with the last one nearly minimised. The method 'Sum of Weighted Ratios' came third, with eight minimised.

When the test results were considered together, (see Bentley & Wakefield 1996f, in Appendix B) it became clear that only one of the multiobjective ranking methods was range-independent, allowed the relative importance of objectives to be specified, and consistently produced not just Pareto-optimal solutions, but truly acceptable solutions, for all tests performed. That multiobjective technique was the 'Sum of Weighted Global Ratios' method (Bentley and Wakefield, 1996f). Not only did this method allow the GA to generate consistently good designs, but it was simple to implement and computationally inexpensive. Hence, this newly developed method was chosen for the generic evolutionary design system.

6.5.5 Sum of Weighted Global Ratios

Once the individuals in the internal population of the GA have had their genotypes mapped to phenotypes, and these phenotypes have been evaluated, the multiobjective method 'Sum of Weighted Global Ratios' (SWGR) must calculate the overall fitness ranking position of each individual.

First, SWGR records global minimum and maximum fitness values for each of the separate fitness values in each individual. For example, if the currently selected modules of evaluation software return six separate fitness values for each individual, then SWGR maintains a list of

six corresponding minimum and maximum fitness values. These values are updated every generation by examining the fitness values of the new individuals in the internal population. If any fitness value falls below the minimum recorded, then this minimum value is updated. If any value falls above the maximum, then this maximum value is updated.

These continuously updated minimum and maximum values, produced so far by a fitness function, give a steadily improving approximation of the *effective range* of that function. SWGR uses this information to convert every fitness value of every individual in the internal population, into a fitness ratio:

$$fitness\_ratio_i = \frac{(fitness\_value_i - \min(fitness\_value))}{(\max(fitness\_value) - \min(fitness\_value))}$$

This negates the effect of having multiple criteria with different effective ranges by equalising all of the effective ranges, allowing every criteria to be treated equally. For example, consider the multicriteria design problem, where one module of evaluation software consists of a function that has returned fitness values between 500 and 5, and another module consists of a function that has returned fitness values between 20 and 2. If an individual has corresponding fitness values of 205 and 12, it would appear that the score for the second criteria is far better than that for the first (the lower the score, the higher the fitness). However, when the effective ranges are taken into consideration, and the fitness values of the individual are converted into ratios, the individual now has two fitness values of: 0.404 and 0.556. In other words, the second value is actually worse than the first, so the design needs to be improved with respect to the second criteria more than it does for the first.

Having converted all the separate fitness values of each individual into ratios, SWGR then weights the separate fitnesses by their relative importance values, as specified by the user. For example, if the user specified that the second criteria should be twice as important, all fitness ratios corresponding to the second criteria are simply multiplied by two. (If the fitness value is

made twice as bad, then the design has to be twice as good for that criteria before the fitness can be restored to its original value.) It should be noted that although importance levels could be set without equalising the effective ranges of the separate criteria, experience shows that the task is considerably more difficult (Bentley and Wakefield, 1996f). In other words, if the separate criteria are not being treated equally to begin with, determining what the weighting value should be to, say double the importance of a criteria, is complete guesswork because of the unknown effective ranges.

Finally, SWGR sums these weighted global fitness ratios, to provide a single, overall fitness value for each individual in the internal population. This is also performed for all the current individuals in the external population, for although they have not had their fitnesses recalculated, their fitness ratios and overall fitness value require updating because the global minimum and maximum fitness values may have been updated.

The sum of weighted global ratios value is used during the next stage of the GA within the generic evolutionary design system to allow the fittest individuals to reproduce more offspring, and to ensure that the weakest individuals are always 'killed' first.

6.6 Generating a New Population of Designs

6.6.1 Emptying the Internal Population

Having mapped all genotypes to phenotypes, evaluated these phenotypes, and established the overall fitnesses of each individual in the internal population, these individuals are next moved into the external population. As described previously, all individuals in the external population are held in order of their overall fitness values, with the fittest at the front of the linked list, and the weakest at the back. Since both populations have a fixed size, an individual must be removed from the external population to make room for the new individual being moved from the internal population. So, for every new individual that is moved to the external population, the weakest external individual is 'killed', i.e. removed from the back of the external population,

cleared, and placed at the end of the internal population. (Note that for the first generation, all the 'empty' or undefined individuals in the external population are treated as totally unfit, and are all quickly replaced by the first, random individuals from the internal population.) In this way, the memory used by individuals is continuously recycled, speeding up execution times by removing the need to constantly allocate and free blocks of memory, see fig. 6.7.

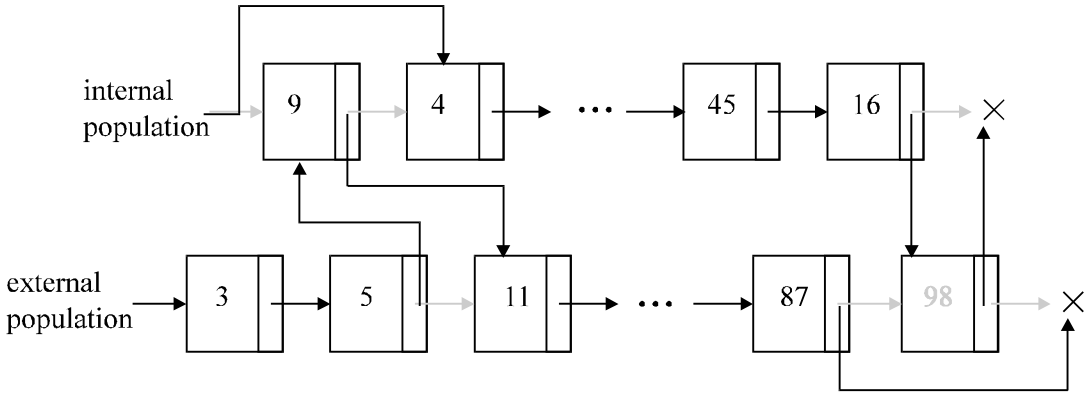

Fig. 6.7 Inserting a new individual from the internal to the external population and removing the weakest individual from the external to the internal population.

Individuals being moved to the external population are always inserted into the appropriate place in this linked list, maintaining the order. This population is kept in order of fitness to improve execution time. For example, when an ordered list of individuals is held in memory, locating the fittest *n*% is simple - they are held in the front *n*% of the list. Likewise, removing the weakest individuals is simple - they are always found at the end of the list. Hence, the practice of keeping an ordered population minimises the computation required to locate individuals. Inevitably, this increases the time needed to insert a new individual in the population, as the appropriate place must be searched for. However, since the fitnesses of new individuals are usually better than that of existing individuals, most new individuals are placed near to, or at, the front of the external population list, meaning that only a very short search is normally needed.

If the fitness of a new individual is the same as an existing individual in the population, then the new individual is placed in front of the existing one. In this way, even in a relatively static

period of evolution (where almost every individual has the same fitness), new individuals are continuously 'pushing out' old individuals from the population.

This process of removing weak individuals and inserting new individuals continues until every member of the internal population has been inserted into the external population. Because the internal population is smaller than the external population, this process always lets some of the fittest 'old' individuals remain in the external population, i.e. fit individuals can 'overlap' two or more subsequent populations, giving them an increased chance to reproduce.

Consequently, the GA used within the generic evolutionary design system employs a more direct emulation of Darwin's 'survival of the fittest' (Darwin, 1859), known as 'steady-state' replacement (Syswerda, 1989). Just as in nature, individuals must exist on limited resources in the environment (i.e., the fixed size of the external population). As competition of these resources increases, the weakest individuals die (i.e., when the external population is full, weak individuals are removed to make room for fitter individuals). This applies to new offspring as well as 'old' individuals - if a new individual is weak, it will be replaced by any fitter individuals that follow it (i.e., the weakest individual is always placed at the back of the list, no matter how old it is). The fitter an individual is, the longer it 'lives', and thus the greater its chances of having offspring (i.e., fit individuals can 'overlap' a number of subsequent external populations).

6.6.2 Advantages of Steady-State Replacement

There were a number of reasons why such an improved analogy with nature in the GA was thought to be advantageous compared to the simple GA that was used in the early prototypes of the system. First, the traditional GA can suffer from certain problems. Because every individual in the population is replaced with offspring every generation, some good solutions never get a chance to reproduce and are lost. Second, mutation and crossover can damage the best genotypes, destroying these good solutions. In some cases, this can result in evolution failing, with solutions actually getting worse (Davis, 1991). Indeed, these frustrating effects

were observed during the evolution of designs by early prototypes of the system (Bentley & Wakefield, 1996c). It was primarily for these reasons that the simple GA used initially within the system was felt to be unsatisfactory.

Steady state replacement is a solution to these problems (Davis, 1991), so it was decided to change the GA in the system and use a similar method. The replacement method used by the GA is slightly different from steady-state, in that this GA inserts a whole sub-population of new individuals (the internal population) into the main population (external population), every generation, whereas 'steady-state' practitioners typically only create and insert one or two children at a time (Davis, 1991). However, the results are much the same, except for a slightly increased selection pressure (i.e. individuals have slightly less time to reproduce, compared to a standard 'steady-state' GA). The fact that unfit individuals are nearly always replaced with fitter individuals, and that the very fittest individuals can remain in the population, remains the same as with a typical steady-state GA. This means that the problem of solutions getting worse during evolution is largely removed. Indeed, this unwanted effect has not been observed since the new GA was installed in the design system.

There are additional benefits when using a replacement method such as steady-state within GAs. As described by Davis, comparisons have shown that steady-state GAs can have an improved ability to evolve good solutions (Davis, 1991). In addition, by increasing crossover and mutation rates, this performance can be improved further (Davis, 1991). In an attempt to harness such benefits, Hierarchical Crossover is used 100% of the time (compared to the more normal 60% to 95% of the time; Bäck, 1993) to generate new offspring in the system.

A possible explanation for the benefits of increasing crossover rates with steady-state GAs can be conceived. In a standard GA with typical crossover rates, where every member of the population is replaced with offspring every generation, many of these offspring will be identical copies of their parents. This is because offspring are generated using crossover 60% to 95% of the time, and hence are simply duplicated from a single parent for 40% to 5% of the time. In

other words, both a standard GA with low crossover rates, and a steady-state GA with high crossover rates, leave some individuals unchanged in a population. However, using the steady-state GA with high crossover rates will ensure that these 'overlapping' individuals are always the fittest members of the population. The standard GA has no such mechanism - some individuals will randomly 'overlap' populations, regardless of their fitnesses. Hence the standard GA effectively gives some random solutions a better chance of reproducing than others, compared to the steady-state GA which only allocates *fitter* solutions a better chance of reproducing.

One final advantage of the steady-state GA used in the evolutionary design system is that fewer evaluations need to take place in a generation. The fitness of every individual is calculated whilst still in the internal population; external individuals can have their overall fitness values changed by updating their fitness ratios (see previous section), but they are never re-evaluated. As the internal population is smaller than the external population, this means that fewer evaluations are required per generation. This is a distinct advantage, for the evaluation of designs is the most computationally expensive part of the system, so any reduction of the number of designs that need to be evaluated will increase the speed of the system. However, steady-state GAs usually take more generations to converge to good solutions than traditional GAs (Syswerda, 1989). This is why the GA in the design system has a slightly increased selection pressure (caused by inserting many individuals every generation instead of one or two) in an attempt to reduce the number of generations required.

Consequently, the GA uses a technique similar to steady-state replacement when individuals are moved from the internal population to the external population. Natural selection is used to remove the weakest individuals and leave the fittest. The advantages of using this method are that it is computationally inexpensive, good solutions are never lost, evolution always improves solutions, steady-state GAs with high crossover rates can have improved performances, and the number of evaluations of phenotypes per generation is reduced.

6.6.3 Picking Parents and Creating Offspring

Once all the individuals have been moved from the internal to the external population, the genetic algorithm must then generate a new internal population of offspring. In order to achieve this, 'parent' individuals must be picked from the external population. (The GA treats individuals in the internal population as 'immature' new individuals, whilst individuals in the external population are considered 'mature' and able to reproduce.)

Since natural selection is performed by the steady-state replacement method described previously, there is no need to allocate fitter individuals in the external population a greater chance of being picked as parents. However, because it is highly desirable to minimise the number of generations needed by the GA to evolve good solutions (and hence reduce the time needed to evolve designs), the GA does preferentially select the fittest individuals.

So, parents are randomly picked from a pre-defined top percentage of the external population. By making this percentage small, only a very few of the elite individuals ever reproduce, resulting in very fast convergence of solutions within the GA. On the other hand, if this figure is made very large (or even simply 100%), then convergence is much slower, with most (or all) of the natural selection being performed by the steady-state replacement. Although a fast convergence by the GA certainly speeds up the system, it does present a very real problem of premature convergence (Levine, 1994), see Chapter Two. Experience after some trial and error, indicates that the lowest percentage that can be used without detrimentally affecting evolution, seems to be 60%. The default value is 80%, i.e. the system uses a small degree of preferential selection by default. (However, one parent in a hundred is picked entirely randomly, to ensure that it is still remotely possible for the few good genes in a very unfit individual to be passed on.)

From the genotypes of each of the two parents that are picked, two new genotypes are generated using the Hierarchical Crossover operator. Mutation is used to modify a bit in an allele with a default probability of 0.001. Mutation is used to either split or delete a coded

primitive with a default probability of 0.01. These two new offspring with, as yet, undefined phenotypes and fitnesses, are placed into the empty spaces of the internal population, see fig. 6.8. Parents are randomly picked from the top *n* percentage of the external population, until the offspring have completely filled the internal population.

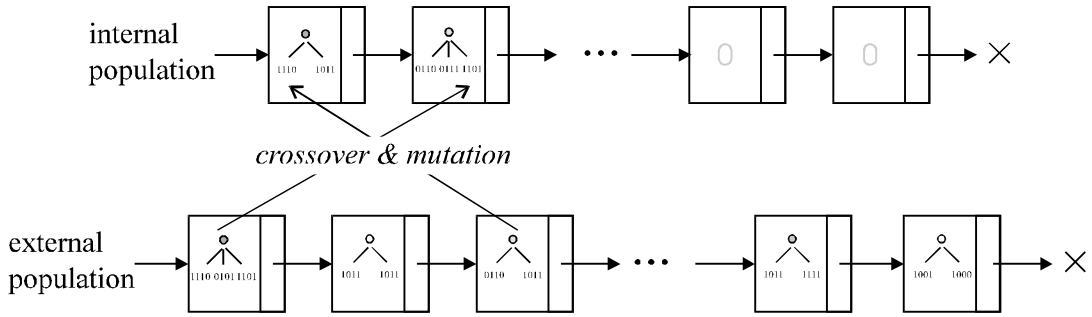

Fig. 6.8 Generating genotypes of new individuals in the internal population from parents in the external population.

6.6.4 Life Spans

The final stage of the GA increments the ages of all the individuals in the external population. (Individuals are aged zero when moved from the internal to the external population, i.e. when being 'born'). If the age of any individual is found to exceed a pre-defined maximum lifespan, then that individual is moved to the back of the linked list (see fig. 6.9), so that it will be treated as being completely unfit. As the weakest individuals are quickly replaced by new offspring in the following generation, this means that an upper limit is placed on the number of generations a single individual can exist in the population.

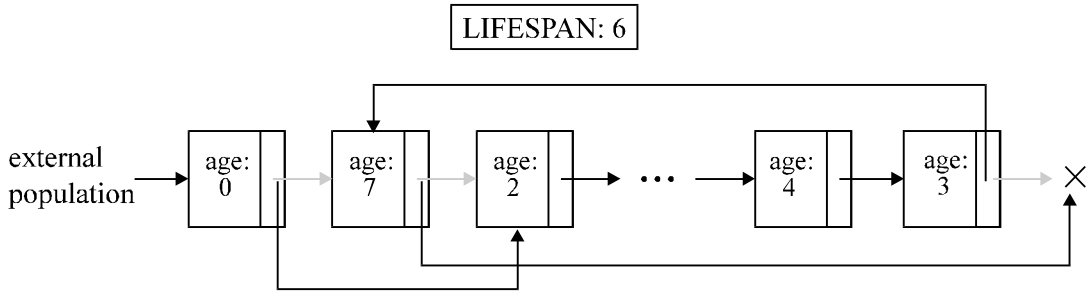

Fig. 6.9 Any individual that reaches its lifespan is moved to the back of the linked list.

This is a new addition to the standard GA, introduced in order to overcome a problem observed during the evolution of designs by the system. As will be described in the next chapter, some modules of evaluation software have random elements to them. For example, the simulation of air-flow fires particles from random initial starting positions and with random initial velocities (between certain ranges), in order to make the movement of these particles over designs more realistic. However, this means that if the same design was evaluated more than once, different fitness values would be returned. In other words, the fitness of designs is variable.

When such modules of evaluation software were used by the design system, it became clear that some designs were being very 'lucky' in their fitness rating. These designs were plainly very poor (to the human eye), but were receiving considerably better fitnesses than those of genuinely good designs, simply because the randomness of the evaluation software happened to be in their favour. For example, designs that were supposed to be 'aerodynamic' as analysed by the air-flow simulator, were managing to get excellent fitness scores simply because the particles fired at them happened to miss the bad parts of the designs.

Moreover, because the steady-state replacement method ensures that the fittest individuals remain in the external population, these lucky individuals were not replaced by offspring. Since many of the lucky individuals were extremely fit, they had many offspring themselves. Very quickly, the entire population would converge to designs similar or identical to these incorrectly scored, bad designs.

There are three possible solutions to the problem of lucky individuals dominating evolution. First, remove the random elements from all modules of evaluation software. Unfortunately, this is not an acceptable option, for it is not feasible to realistically model real-world phenomena such as air-flow, without incorporating some randomness (i.e. noise). A second alternative is to repeat the evaluation a number of times. For example, either the number of particles fired at designs in the air-flow simulator could be increased, or designs could be re-evaluated and the lowest fitness value used. (This latter method was used by Cliff to overcome the problem of

randomness caused by the real-word evaluation of the vision and movement of robots; Harvey et. al., 1994.) Such methods would decrease the probability of a lucky individual appearing. However, increasing the evaluations per design is not a desirable solution, as it dramatically slows down the time needed by the system to evolve designs. Moreover, there would still be a remote possibility that lucky individuals could dominate evolution.

The third alternative solution to this problem, is to limit the amount of time any very fit individual can spend in the population. As mentioned previously, most individuals are typically pushed out of the external population by new offspring that usually have fitnesses equal to or better than the best in the population. In other words, the normal process of evolution seems to ensure that no individual remains in the population for more than about four or five generations. However, very lucky individuals do not behave in this manner. A lucky individual will typically sit at the front of the population forever - becoming essentially immortal - and generating huge numbers of (usually less lucky and hence very unfit) offspring. Because it is so lucky, its fitness far exceeds the score that any legitimate design could obtain, and so it is never replaced. Hence, if these lucky individuals are prevented from becoming immortal, this problem is overcome.

By making the value of the lifespan (or maximum age) of all individuals high enough so that no individuals 'die' of old age when their fitnesses are calculated by non-random evaluation modules, this new mechanism can be made to affect only the unwanted 'lucky' individuals. Since the evolutionary design system automatically reports whenever it has located an individual that has reached its lifespan, the lifespan value can be set very simply. Experiments have shown that no individuals (evaluated by non-random modules) reach their lifespan when this value is six or more, i.e. individuals usually seem to be pushed out of the population after a maximum of five generations. Hence, the default value of the lifespan parameter is six.

This solution has been found to work very effectively. A dramatic reduction of lucky individuals was evident when the 'lifespan' mechanism was introduced to the evolutionary

design system. Moreover, since this technique only affects the unwanted lucky individuals, the genuinely good designs evolved by the system remained unaffected.

6.7 Summary

This chapter has described the core of the generic evolutionary design system: the genetic algorithm. This algorithm was felt to be the most suitable choice because of its robustness, the fact that it is comparable to the human design process, that it has been successfully demonstrated on all related types of design system, and because it is the best emulation in Computer Science of natural evolution (which has produced the most complex designs ever known).

The genetic algorithm uses two populations of individuals: the internal and external population, which are stored as linked lists. Designs are created 'from scratch' by seeding the initial population with entirely random genotypes, which correspond to random, free-form 'blobs'. The GA uses an explicit mapping stage to generate the phenotypes of individuals from the genotypes. This allows every genotype to correspond to a legal phenotype (with no primitive shapes overlapping) and also allows a partial design to be reflected in order to create symmetrical designs of greater complexity.

These phenotypes then each receive multiple fitness scores from the selected evaluation software. The GA uses a novel multiobjective fitness ranking method (SWGR) to ensure that every criteria in the modules of evaluation software is treated equally. This removes the need for laborious fine-tuning of weighting parameters to make the system evolve acceptable designs. The ranking method also allows separate criteria to have increased relative importance - an essential feature for some design problems.

Once an overall fitness value has been established for each individual in the internal population, the GA then introduces these new individuals into the main, external population. A 'steady-

state' replacement method is used, where the weakest individuals are removed to make room for the new individuals from the internal population. This is an efficient way to ensure that good solutions in the population are never lost, which can increase the performance of evolution and reduce the number of evaluations required per generation.

Next, the GA picks parents from the fittest 80% of the external population, increasing the selection pressure in order to reduce the number of generations required by the GA to converge to good solutions. The hierarchical crossover and mutation operators described in the previous chapter are then used to generate new individuals in the internal population, from the parent individuals. Finally, the age of each individual is incremented, with any individuals that are found to have reached their lifespan, being made very unfit. This prevents 'lucky' individuals from becoming immortal and dominating evolution.

The GA then maps the genotypes of the new individuals in the internal population to corresponding phenotypes, which are then evaluated, and so on, until a pre-defined number of generations have passed, or the user halts evolution.