Chapter 5
Genetic Algorithms

Colin R. Reeves

Abstract Genetic algorithms (GAs) have become popular as a means of solving hard combinatorial optimization problems. The first part of this chapter briefly traces their history, explains the basic concepts and discusses some of their theoretical aspects. It also references a number of sources for further research into their applications. The second part concentrates on the detailed implementation of a GA. It discusses the fundamentals of encoding a 'genotype' in different circumstances and describes the mechanics of population selection and management and the choice of genetic 'operators' for generating new populations. In closing, some specific guidelines for using GAs in practice are provided.

5.1 Introduction

The term *genetic algorithm*, almost universally abbreviated nowadays to GA, was first used by John Holland [1], whose book *Adaptation in Natural and Artificial Systems* of 1975 was instrumental in creating what is now a flourishing field of research and application that goes much wider than the original GA. Many people now use the term *evolutionary computing* or *evolutionary algorithms* (EAs), in order to cover the developments of the last 15 years. However, in the context of metaheuristics, it is probably fair to say that GAs in their original form encapsulate most of what one needs to know.

Holland's influence in the development of the topic has been very important, but several other scientists with different backgrounds were also involved in developing similar ideas. In 1960s Germany, Ingo Rechenberg [2] and Hans-Paul Schwefel [3] developed the idea of the *Evolutionsstrategie* (in English, *evolution strategy*), while—also in the 1960s—Bremermann, Fogel and others in the USA implemented

Colin R. Reeves
Department of Mathematics, Statistics and Engineering Science, Coventry University, Priory St, Coventry, UK
e-mail: c.reeves@coventry.ac.uk

M. Gendreau, J.-Y. Potvin (eds.), *Handbook of Metaheuristics*,
International Series in Operations Research & Management Science 146,
DOI 10.1007/978-1-4419-1665-5_5,

their idea for what they called *evolutionary programming*. The common thread in these ideas was the use of mutation and selection—the concepts at the core of the neo-Darwinian theory of evolution.[1] Although some promising results were obtained, evolutionary computing did not really take off until the 1980s. Note the least important reason for this was that the techniques needed a great deal of computational power. Nevertheless, the work of these early pioneers is fascinating to read in the light of our current knowledge; David Fogel (son of one of the early pioneers) has documented some of this work in [4].

1975 was a pivotal year in the development of genetic algorithms. It was in that year that Holland's book was published, but perhaps more relevantly for those interested in metaheuristics, that year also saw the completion of a PhD thesis by one of Holland's graduate students, Ken De Jong [5]. Other students of Holland's had completed theses in this area before, but this was the first to provide a thorough treatment of the GA's capabilities in optimization.

A series of further studies followed, the first conference on the nascent subject was convened in 1985, and another graduate student of Holland's, David Goldberg, produced first an award-winning PhD thesis on his application to gas pipeline optimization, and then, in 1989, an influential book [6]—*Genetic Algorithms in Search, Optimization, and Machine Learning*. This was the final catalyst in setting off a sustained development of GA theory and applications that is still growing rapidly.

Optimization has a fairly small place in Holland's work on adaptive systems, yet the majority of research on GAs tends to assume this is their purpose. De Jong, who initiated this interest in optimization, has cautioned that this emphasis may be misplaced in a paper [7] in which he contends that GAs are not really function optimizers, and that this is in some ways incidental to the main theme of adaptation. Nevertheless, using GAs for optimization is very popular, and frequently successful in real applications, and to those interested in metaheuristics, it will undoubtedly be the viewpoint that is most useful.

Unlike the earlier evolutionary algorithms, which focused on mutation and could be considered as straightforward developments of hill-climbing methods, Holland's GA had an extra ingredient—the idea of recombination. It is interesting in this regard to compare some of the ideas being put forward in the 1960s in the field of *operational research* (OR).

OR workers had by that time begun to develop techniques that seemed able to provide 'good' solutions, even if the quality was not provably optimal (or even near-optimal). Such methods became known as *heuristics*. A popular technique, which remains at the heart of many of the metaheuristics described in this handbook, was that of *neighbourhood search*, which has been used to attack a vast range of combinatorial optimization problems. The basic idea is to explore 'neighbours' of an existing solution—these being defined as solutions obtainable by a specified operation on the base solution.

One of the most influential papers in this context was that published by Lin [8], who found excellent solutions to the travelling salesman problem by investigating

[1] Well-meaning attempts to read off the validity or otherwise of Darwinism from the performance of GAs are illegitimate. GAs are clear examples of 'intelligent design'.

neighbourhoods formed by breaking any three links of a tour and re-connecting them. Empirically, Lin found that these '3-optimal' solutions were of excellent quality—in the case of the (rather small) problems he investigated, often close to the global optimum. However, he also made another interesting observation and suggested a way of exploiting it. While starting with different initial permutations gave different 3-optimal solutions, these 3-optimal solutions were observed to have a lot of features (links) in common. Lin therefore suggested that search should be concentrated on those links about which there was not a consensus, leaving the common characteristics of the solutions alone. This was not a GA as Holland was developing it, but there are clear resonances. Much later, after GAs had become more widely known, Lin's ideas were re-discovered as 'multi-parent recombination' and 'consensus operators'.

Other OR research of the same era took up these ideas. Roberts and Flores [9] (apparently independently) used a similar approach to Lin's for the TSP, while Nugent et al. [10] applied this basic idea for the quadratic assignment problem. However, the general principle was not adopted into OR methodology, and relatively little was done to exploit the idea until GAs came on the OR scene in the 1990s.

In what follows, Section 5.2 provides an overview of the basic GA concepts, Section 5.3 gives a sketch of the theoretical background, while Section 5.4 lists some important sources for further exploration. The remaining sections focus on the various stages required for the implementation of a GA.

5.2 Basic Concepts

Assume we have a discrete search space $\mathscr{X}$ and a function

$$f : \mathscr{X} \mapsto \mathbb{R}.$$

The general problem is to find

$$\arg\min_{x \in \mathscr{X}} f.$$

Here, x is a vector of *decision variables*, and f is the *objective function*. We assume here that the problem is one of minimization, but the modifications necessary for a maximization problem are nearly always obvious. Such a problem is commonly called a discrete or combinatorial optimization problem (COP).

One of the distinctive features of the GA approach is to allow the separation of the *representation* of the problem from the actual variables in which it was originally formulated. In line with biological usage of the terms, it has become customary to distinguish the 'genotype'—the encoded representation of the variables, from the 'phenotype'—the set of variables themselves. That is, the vector x is represented by a string s, of length l, made up of symbols drawn from an alphabet $\mathscr{A}$, using a mapping

$$c : \mathscr{A}^l \mapsto \mathscr{X}.$$

In practice, we may need to use a search space

$$\mathscr{S} \subseteq \mathscr{A}^l,$$

to reflect the fact that some strings in the image of $\mathscr{A}^l$ under c may represent invalid solutions to the original problem. (This is a potential source of difficulty for GAs in combinatorial optimization—a topic that is covered in [11].) The string length l depends on the dimensions of both $\mathscr{X}$ and $\mathscr{A}$, and the elements of the string correspond to 'genes', and the values those genes can take to 'alleles'. This is often designated as the *genotype–phenotype mapping*. Thus the optimization problem becomes one of finding

$$\arg\min_{s \in \mathscr{S}} g,$$

where the function

$$g(s) = f(c(s)).$$

It is usually desirable that c should be a bijection. (The important property of a bijection is that it has an inverse, i.e., there is a unique vector x for every string s, and a unique string s for every vector x.) In some cases the nature of this mapping itself creates difficulties for a GA in solving optimization problems.

In using this device, Holland's ideas are clearly distinct from the similar methodology developed by Rechenberg [2] and Schwefel [3], who preferred to work with the original decision variables directly. Both Holland's and Goldberg's books claim that representing the variables by binary strings (i.e., $\mathscr{A} = \{0, 1\}$) is in some sense 'optimal', and although this idea has been challenged, it is still often convenient from a mathematical standpoint to consider the binary case. Certainly, much of the theoretical work in GAs tends to make this assumption. In applications, many representations are possible—some of the alternatives that can be used in particular COPs are discussed in [11].

The original motivation for the GA approach was a biological analogy. In the selective breeding of plants or animals, for example, offspring are sought that have certain desirable characteristics—characteristics that are determined at the genetic level by the way the parents' chromosomes combine. In the case of GAs, a *population* of strings is used, and these strings are often referred to in the GA literature as *chromosomes*. The recombination of strings is carried out using simple analogies of genetic *crossover* and *mutation*, and the search is guided by the results of evaluating the objective function f for each string in the population. Based on this evaluation, strings that have higher *fitness* (i.e., represent better solutions) can be identified, and these are given more opportunity to breed. It is also relevant to point out here that fitness is not necessarily to be identified simply with the composition $f(c(\mathrm{s}))$; more generally, fitness is $h(f(c(\mathrm{s})))$ where $h : \mathbb{R} \mapsto \mathbb{R}^+$ is a suitable monotonic function used to eliminate the problem of 'negative' fitness.

Perhaps the most fundamental characteristic of genetic algorithms is their use of populations of many strings. Here again, the German 'evolution strategy' (ES) school initially did not use populations and focused almost exclusively on 'mutation' operators which are generally closer in concept to the types of operator used

in neighbourhood search and its extensions. Holland did use mutation, but in his scheme it is generally treated as subordinate to crossover. Thus, in Holland's GA, instead of the search moving from point to point as in methods based on local search, the whole set of strings undergoes 'reproduction' in order to generate a new population.

De Jong's work established that population-based GAs using crossover and mutation operators could successfully deal with optimization problems of several different types, and in the years since his work was published, the application of GAs to COPs has grown almost exponentially.

These operators and some developments of them are described more fully in Sections 5.9 and 5.10. At this point, however, it might be helpful to provide a very basic introduction. Crossover is a matter of replacing some of the genes in one parent by corresponding genes of the other. An example of one-point crossover would be the following. Given the parents `P1` and `P2`, with crossover point 3 (indicated by `X`), the offspring will be the pair `O1` and `O2`:

```
P1    1 0 1 0 0 1 0        O1    1 0 1 1 0 0 1
            X
P2    0 1 1 1 0 0 1        O2    0 1 1 0 0 1 0
```

The other common operator is mutation in which a gene (or subset of genes) is chosen randomly and the allele value of the chosen genes is changed. In the case of binary strings, this simply means complementing the chosen bit(s). For example, the string `O1` above, with genes 3 and 5 mutated, would become `1 0 0 1 1 0 1`. A simple template for the operation of a genetic algorithm is shown in Figure 5.1. The individual parts of this very general formulation will be discussed in detail later.

```
Choose an initial population of chromosomes;
while termination condition not satisfied do
   repeat
        if crossover condition satisfied then
        {select parent chromosomes;
        choose crossover parameters;
        perform crossover};
        if mutation condition satisfied then
        {choose mutation points;
        perform mutation};
        evaluate fitness of offspring
   until sufficient offspring created;
select new population;
endwhile
```

Fig. 5.1 A genetic algorithm template. This is a fairly general formulation, accommodating many different forms of selection, crossover and mutation. It assumes user-specified conditions under which crossover and mutation are performed, a new population is created, and whereby the whole process is terminated.

5.3 Why Does It Work?

Exactly how and why GAs work is still hotly debated. There are various schools of thought, and none can be said to provide a definitive answer. A comprehensive survey is available in [12]. Meanwhile, the following is a brief guide to the main concepts that have been used.

5.3.1 The 'Traditional' View

Holland's explanation of why it is advantageous to search the space $\mathscr{A}^l$ rather than $\mathscr{X}$ hinges on three main ideas. Central to this understanding is the concept of a *schema* (plural *schemata*). A schema is a subset of the space $\mathscr{A}^l$ in which all the strings share a particular set of defined values. This can be represented by using the alphabet $\mathscr{A} \cup *$; in the binary case, `1 * * 1`, for example, represents the subset of the 4-dimensional hypercube $\{0,1\}^4$ in which both the first and the last genes take the value 1, i.e., the strings `{1 0 0 1, 1 0 1 1, 1 1 0 1, 1 1 1 1}`.

The first of Holland's ideas is that of *intrinsic* (also known as *implicit*) parallelism—the notion that information on many schemata can be processed in parallel. Under certain conditions that depend on population size and schema characteristics, Holland estimated that a population of size M contains information on $\mathscr{O}(M^3)$ schemata. However, these schemata cannot *actually* be processed in parallel, because independent estimates of their fitness cannot be obtained in general [14].

The second concept is expressed by the so-called Schema Theorem, in which Holland showed that if there are $N(S,t)$ instances of schema S in the population at time t, then at the next time step (following reproduction), the expected number of instances in the new population can be bounded by

$$E[N(S,t+1)] \geq \frac{F(S,t)}{\overline{F(t)}} N(S,t)\{1-\varepsilon(S,t)\},$$

where $F(S,t)$ is the fitness of schema S, $\overline{F(t)}$ is the average fitness of the population, and $\varepsilon(S,t)$ is a term that reflects the potential for genetic operators to destroy instances of schema S.

By failing to appreciate the stochastic and dynamic nature of this relationship, somewhat extravagant conclusions have been drawn from this theorem, expressed in the frequently made statement that good schemata will receive exponentially increasing numbers of trials in subsequent generations. However, it is clear that the Schema Theorem is a result in expectation only, and even then for just one generation. Any attempt to extrapolate this result for more than one generation is doomed to failure because the terms are then no longer independent of what is happening in the rest of the population. Moreover, given the finite population size, it is clear that any exponential increase cannot last very long.

Holland also attempted to model schema processing (or hyperplane competitions) by means of an analogy to stochastic two-armed bandit problems. This is

a well-known statistical problem: we are given two 'levers' which if pulled give 'payoff' values according to different probability distributions. The problem is to use the results of previous pulls in order to maximize the overall future expected payoff. In [1] it is argued that a GA approximates an 'optimal' strategy which allocates an (exponentially) increasing number of trials to the observed better lever; this is then used to contend for the supposed efficiency of a GA in distinguishing between competing schemata and hyperplanes.

Early accounts of GAs suggested quite strongly that in a GA we had thus discovered an algorithm that used the best available search strategy to solve not merely one, but many hyperplane competitions at once: the 'only case where combinatorial explosion works in our favour'. Unfortunately, Wolpert and Macready's 'No-Free-Lunch' Theorem (NFLT) [13] has rather destroyed such dreams.[2]

In fact, intrinsic parallelism turns out to be of strictly limited application; it merely describes the number of schemata that are likely to be present in some numbers given certain assumptions about string length, population size and (most importantly) the way in which the population has been generated—and the last assumption is unlikely to be true except at a very early stage of the search. Even then, only in very unusual circumstances—that of orthogonal populations [14]—could the hyperplane competitions actually be processed in parallel; normally, the competitions are not independent. The two-armed bandit analogy also fails in at least two ways: Macready and Wolpert [15] have firstly argued that there is no reason to believe that the strategy described by Holland as approximated by a GA is an optimal one, while they also believe there is also a flaw in Holland's mathematics.

This is not to say that the Schema Theorem in particular, or the idea of a schema in general, is useless, but that what it says is of limited and mainly short-term value—principally, that certain schemata are likely to increase their presence in the next population, and that those schemata will be on the average fitter, and less resistant to destruction by crossover and mutation, than those that do not. Nevertheless, several researchers are working on new ways of formulating and understanding schema theory, while connecting it to other approaches; a recent summary can be found in [16].

This brings us to the third assumption implicit in the implementation of a GA—that the recombination of small pieces of the genotype (good schemata) into bigger pieces is indeed a sensible method of finding optimal solutions. Goldberg [6] calls this the building-block hypothesis (BBH). There is certainly some negative evidence, in that problems constructed to contain misleading building blocks may indeed be hard for a GA to solve. The failure of the BBH is often invoked as an explanation when a GA fails to solve particular COPs.

However, the properties of these problems are not usually such that they are uniquely difficult for GAs. Holland himself, with two other co-workers, looked for positive evidence in favour of the building-block hypothesis [17] and found

[2] The NFLT, put simply, says that on the average, nothing—ant colonies, GAs, simulated annealing, tabu search, etc.—is better than random search. Success comes from adapting the technique to the problem at hand, which of course implies some input of information from the researcher.

the results rather problematical: functions constructed precisely to provide a 'royal road' made up of building blocks of increasing size and fitness turned out to be much more efficiently solved by 'non-genetic' methods.

5.3.2 Other Approaches

By writing his theorem in the form of a lower bound, Holland was able to make a statement about schema *S* that is independent of what happens to other schemata. However, in practice what happens to schema *S* will influence the survival (or otherwise) of other schemata, and what happens to other schemata will affect what happens to *S*, as is made plain by the exact models of Vose [18] and Whitley [19].

Markov chain theory [18, 19] has been applied to GAs [20–22] to gain a better understanding of the GA as a whole. However, while the results are fascinating in illuminating some nuances of GA behaviour, the computational requirements are formidable for all but the smallest of problems, as shown by De Jong et al. [22], for example.

Shapiro et al. [23] first examined GAs from a statistical mechanics perspective, and there is a growing literature on this topic. Peck and Dhawan [24] have linked GAs to global randomized search methods. But one of the difficulties in analyzing GAs is that there is not a single generic GA, the behaviour of which will characterize the class of algorithms that it represents. In practice, there is a vast number of ways of implementing a GA, as will be seen in the discussion later, and what works in one case may not work in another. Some workers have therefore tried to look for ways of predicting algorithm performance for particular problem classes.

Reeves and Wright [14] summarize a perspective based on relating GAs to statistical methods of experimental design, which draws upon the biological concept of *epistasis*. This expresses the idea that the expression of a chromosome is not merely a sum of the effects of its individual alleles, but that the alleles located in some genes influence the expression of the alleles in others. From a mathematical viewpoint, epistasis is equivalent to the existence of interactions in the fitness function. If we knew the extent of these non-linearities, we might be able to choose an appropriate algorithm. Unfortunately, as is explained in [25], it is unlikely that this approach will be successful, although the literature surrounding the question of epistasis has produced some useful insights into GAs.

Several authors [26–28] have pointed out connections between GAs and neighbourhood search methods, and this has led to a considerable literature on the analysis of problem landscapes. The concept of a landscape has been used informally for many years, but recent work [29, 30] has put the idea on a rigorous mathematical foundation which is still being explored. Some of its uses in the context of GAs is described in [31]. It appears that this way of thinking about algorithms has great potential for unifying different metaheuristics and increasing our understanding of them.

5.4 Applications and Sources

There are numerous examples of the successful application of GAs to combinatorial optimization problems. Books such as those by Davis [32] and Chambers [33, 34] are useful in displaying the range of problems to which GAs have been applied. In a chapter such as this, it is impossible to give an exhaustive survey of relevant applications of GAs, but [11] lists some of the more useful and accessible references that should be of interest to people who are experimenting with metaheuristics. However, because of the enormous growth in reported applications of GAs, this list is inevitably incomplete, as well as somewhat dated already. For a time, Alander attempted to maintain a comprehensive bibliography: an early version of this is included in [34]. However, this is one area where the phenomenon of exponential growth is indubitable, and the sheer number of papers published in the last 15 years have rather overwhelmed this enterprise. Nonetheless, updates are made available periodically of selected papers in specific areas—the one of most interest to readers of this book being the OR bibliography [35], which is claimed to be comprehensive up to 1998, although it also includes some papers published later.

For more information on applications, and on GAs in general, the reader has several useful books to choose from: the early ones by Holland, Goldberg and Michalewicz [1, 6, 36] tend to be over-committed to the schema-processing point of view, but they are all still useful sources of information. Reeves [37] also reflects the state of the theory at the time the book was written, although it covers other heuristic methods too. More recently, Mitchell [38] and Falkenauer [39] demonstrate a more careful approach to schemata, and Bäck [40] covers the wider field of evolutionary algorithms. Eiben and Smith [41] also provide an elementary overview of the whole field, while—in contrast—Spears [42] offers an in-depth study on the trade-off between mutation and crossover.

All are worth consulting, but the best book now available is the recent work by De Jong [43]. For a very rigorous theoretical study, there is the book by Vose [44], which deals mainly with the Markov chain and dynamical systems approach, while Reeves and Rowe [12] have surveyed in some detail several other theoretical perspectives on GAs. Another rigorous theoretical study is that by Schmitt [45, 46].

There are now also many conferences on GAs and related topics—too many to list in detail. The original biennial *International Conference on Genetic Algorithms* series [47–53] is still of considerable historical interest[3], while the IEEE established an alternative series under the title of the *Congress on Evolutionary Computation* [54–56]. These have now merged, under the auspices of the ACM since 2005, to create the annual GECCO series of conferences [57–59]. In Europe, there are two biennial series of somewhat wider scope: the *Parallel Problem Solving from Nature* series [67–72] and the *International Conference on Artificial Neural Networks and Genetic Algorithms* [77–82], recently renamed the *International Conference on Adaptive and Natural Computing Algorithms* [83, 84]. For the theoretically minded,

[3] Apart from the intrinsic interest of these papers, it is well worth checking to see if someone has tried your bright new idea already!

there is a biennial workshop to consider—the *Foundations of Genetic Algorithms* series [85–93].

There are also many journals now publishing GA-related research. The major GA journals are *Evolutionary Computation* (MIT Press) and *IEEE Transactions on Evolutionary Computation* (IEEE); other theoretical articles appear in journals related to AI or to complex systems. Most OR journals—*INFORMS Journal on Computing*, *Computers and OR*, *Journal of the OR Society*, *European Journal of OR*, etc.—have frequent papers on GAs, mainly applications. There are discussion groups on the Internet (`comp.ai.genetic`), and the moderated news digest at `GA-List-Request@aic.nrl.navy.mil`.

5.5 Initial Population

The previous sections have provided an overview of the underlying concepts, but it should be clear already that implementation of a GA requires many practical decisions. The major initial questions to consider relate to the population: first its size and second the method by which its individuals are chosen. The size of the population has been approached from several theoretical points of view, although the underlying idea is always of a trade-off between efficiency and effectiveness. Intuitively, it would seem that there should be some 'optimal' value for a given string length, on the grounds that too small a population would not allow sufficient room for exploring the search space effectively, while too large a population would so impair the efficiency of the method that no solution could be expected in a reasonable amount of time. Goldberg [94, 95] was probably the first to attempt to answer this question, using the idea of schemata. Unfortunately, from this viewpoint, it appeared that the population size M should increase as an exponential function of the string length. Experimental evidence [96, 97] suggests that populations of the size proposed by Goldberg's theory are not necessary.

A slightly different question that we could ask is regarding a *minimum* population size for a meaningful search to take place. In Reeves [98], the initial principle was adopted that, at the very least, every point in the search space should be reachable from the initial population by crossover only. This requirement can only be satisfied if there is at least one instance of every allele at each locus in the whole population of strings. On the assumption that the initial population is generated by a random sample with replacement (which is a conservative assumption in this context), the probability that at least one allele is present at each locus can be found. For binary strings this is easily seen to be

$$P_2^* = (1 - (1/2)^{M-1})^l,$$

from which we can calculate that, for example, a population of size 17 is enough to ensure that the required probability exceeds 99.9% for strings of length 50. For q-ary alphabets, the calculation is somewhat less straightforward, but expressions are

given in [98] that can be converted numerically into graphs for specified confidence levels. The results of this work suggested that a population growth of $\mathcal{O}(\log l)$ would be sufficient to cover the search space.

Finally, as to how the population is chosen, it is nearly always assumed that initialization should be random. Rees and Koehler [99], using a model-based approach that draws on the theoretical work of Vose [18], have demonstrated that sampling without replacement is preferable in the context of very small populations. More generally, it is obvious that randomly chosen points do not necessarily cover the search space uniformly, and there may be advantages in terms of coverage if we use more sophisticated statistical methods, especially for non-binary alphabets. One such simple idea is a generalization of the Latin hypercube which can be illustrated as follows:

Suppose each gene has 5 alleles, labelled $0,\ldots,4$. We choose the population size M to be a multiple of 5, and the alleles in each 'column' are generated as an independent random permutation of $0,\ldots,(M-1)$, which is then taken modulo 5. Figure 5.2 shows an example for a population of size 10. To obtain search space coverage at this level with simple random initialization would need a much larger population.

Individual	Gene
1	0 1 3 0 2 4
2	1 4 4 2 3 0
3	0 0 1 2 4 3
4	2 4 0 3 1 4
5	3 3 0 4 4 2
6	4 1 2 4 3 0
7	2 0 1 3 0 1
8	1 3 3 1 2 2
9	4 2 2 1 1 3
10	3 2 4 0 0 1

Fig. 5.2 An example of Latin hypercube sampling for $l = 6$ and $|\mathcal{A}| = 5$. Notice that each allele occurs exactly twice for each gene.

Another point to mention here is the possibility of 'seeding' the initial population with known good solutions. Some reports (e.g., in [100, 101]) have found that including a high-quality solution, obtained from another heuristic technique, can help a GA find better solutions rather more quickly than it can from a random start. However, there is also the possibility of inducing premature convergence [102, 103].

5.6 Termination

Unlike simple neighbourhood search methods that terminate when a local optimum is reached, GAs are stochastic search methods that could in principle run for ever. In practice, a termination criterion is needed; common approaches are to set a limit

on the number of fitness evaluations or the computer clock time or to track the population's diversity and stop when this falls below a preset threshold. The meaning of diversity in the latter case is not always obvious, and it could relate either to the genotype or to the phenotype, or even, conceivably, to the fitnesses, but the most common way to measure it is by genotype statistics. For example, we could decide to terminate a run if at every locus the proportion of one particular allele rose above 90%. Some attempts have been made to attack this problem from a theoretical point of view [104, 105], but as they are based on the idea of finding a probabilistic guarantee that all possible strings have been seen, their practical application is limited.

5.7 Crossover Condition

Given the stress on recombination in Holland's original work, it might be thought that crossover should always be used, but in fact there is no reason to suppose that it has to be so. Thus, while we could follow a strategy of crossover-AND-mutation to generate new offspring, it is also possible to use crossover-OR-mutation. There are many examples of both in the literature. The first strategy initially tries to carry out crossover, then attempts mutation on the offspring (either or both). It is conceivable that in some cases nothing actually happens at all with this strategy—the offspring are simply clones of the parents. Others always do something, either crossover or mutation, but not both. (Even then, cloning is still possible with crossover if the parents are too alike.)

The mechanism for implementing such choices is customarily a randomized rule, whereby the operation is carried out if a pseudo-random uniform deviate exceeds a threshold value. In the case of crossover, this is often called the crossover rate, often denoted by the symbol χ. For mutation, we have a choice between describing the number of mutations per string or per bit; bit-wise mutation, at a rate denoted by μ, is more common.

In the -OR- case, there is a further possibility of modifying the relative proportions of crossover and mutation as the search progresses. Davis [32] has argued that different rates are appropriate at different times: high crossover at the start, high mutation as the population converges. In fact, he has suggested that the operator proportions could be adapted online, in accordance with their track record in finding new high-quality chromosomes.

5.8 Selection

The basic idea of selection is that it should be related to fitness, and the original scheme for its implementation is commonly known as the *roulette-wheel* method. It uses a probability distribution for selection in which the selection probability of

a given string is proportional to its fitness. Figure 5.3 provides a simple example of roulette-wheel selection (RWS). Pseudo-random numbers are used one at a time to choose strings for parenthood. For example, in Figure 5.3, the number 0.13 would select string 1, the number 0.68 would select string 4.

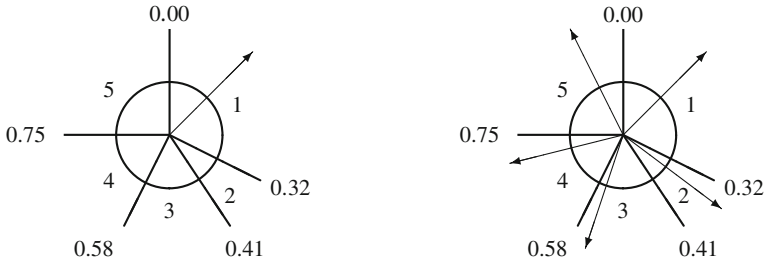

Fig. 5.3 Suppose there are five strings in a population with fitnesses $\{32, 9, 17, 17, 25\}$, respectively. The probability of selection of each individual is proportional to the area of a sector of a roulette-wheel (or equivalently, to the angle subtended at the centre). The numbers on the spokes of the wheel are the cumulative probabilities for use by a pseudo-random number generator. On the left we have standard roulette-wheel selection, with a single pointer that has to be spun five times. On the right we have SUS, using five connected equally spaced pointers; one spin provides five selections.

Finding the appropriate number for a given pseudo-random number r requires searching an array for values that bracket r—this can be done in $\mathcal{O}(\log M)$ time for a population of size M. However, this approach has a high stochastic variability, and the actual number of times N_C that chromosome C is selected in any generation may be very different from its expected value $E[N_C]$. For this reason, sampling *without* replacement may be used to ensure that at least the integral part of $E[N_C]$ is achieved, with fractions being allocated using random sampling.

In practice, Baker's *stochastic universal selection* (SUS) [106] is a particularly effective way of realizing this outcome. Instead of a single choice at each stage, we imagine that the roulette wheel has an equally spaced multi-armed spinner. Spinning the wheel produces simultaneously the values N_C for all the chromosomes in the population. From the viewpoint of statistical sampling theory, this corresponds to systematic sampling [107]. Experimental work by Hancock [108] clearly demonstrates the superiority of this approach, although much published work on applications of GAs still appears to rely on the basic roulette-wheel method[4].

An associated problem is that of finding a suitable measure of fitness for the members of the population. Simply using the objective function values $f(\mathrm{x})$ is rarely sufficient, because the scale on which $f(\mathrm{x})$ is measured is important. (For example, values of 10 and 20 are much more clearly distinguished than 1010 and 1020.) Also, in some cases, observed values of f may be negative, which complicates

[4] Note that the purpose of SUS is not to reduce the total of random numbers needed. Having generated a multiset of size M as our 'mating pool', we still have to decide which pairs mate together, whereas in RWS we can simply pair them in the order generated.

fitness-proportional schemes. Further, if the objective is minimization rather than maximization, a transformation is clearly required.

Some sort of scaling is thus usually applied, and Goldberg [6] gives a simple algorithm to deal with both minimization and maximization. The method is cumbersome, however, and it needs continual re-scaling as the search progresses. Two alternatives provide more elegant solutions.

5.8.1 Ranking

Ranking the chromosomes in fitness order loses some information, but there is no need for re-scaling, and selection algorithm is simpler and more efficient. Suppose the probability of selecting the string that is ranked kth in the population is denoted by $P[k]$. In the case of linear ranking, we assume that

$$P[k] = \alpha + \beta k,$$

where α and β are constants. The requirement that $P[k]$ be a probability distribution gives us one condition:

$$\sum_{k=1}^{M} (\alpha + \beta k) = 1,$$

which leaves us free to choose the other parameter in a way that tunes the *selection pressure*. This term is loosely used in many papers and articles on GAs. Here, we mean the following:

Definition 5.1 *Selection pressure*

$$\phi = \frac{Prob.[selecting\ fittest\ string]}{Prob.[selecting\ average\ string]}.$$

In the case of linear ranking, we interpret the average as meaning the *median* string, so that

$$\phi = \frac{\alpha + \beta M}{\alpha + \beta (M+1)/2}$$

(This assumes the population size is odd—however, the analysis holds *mutatis mutandis* for the case of an even number.) Some simple algebra soon establishes that

$$\beta = \frac{2(\phi - 1)}{M(M-1)} \quad \text{and} \quad \alpha = \frac{2M - \phi(M+1)}{M(M-1)}$$

which implies that $1 \leq \phi \leq 2$. With this framework, it is easy to see that the cumulative probability distribution can be stated in terms of the sum of an arithmetic progression, so that finding the appropriate k for a given pseudo-random number r is simply a matter of solving the quadratic equation

$$\alpha k + \beta \frac{k(k+1)}{2} = r,$$

for k, which can be done simply in $O(1)$ time. The formula is

$$k = \frac{-(2\alpha+\beta) \pm \sqrt{(2\alpha+\beta)^2 + 4\beta r}}{2\beta}.$$

In contrast, searching for k (given a value for r) using ordinary fitness-proportional selection needs at least $\mathcal{O}(\log M)$ time.

Other functions can be used besides linear ranking [108, 109] but the above scheme is sufficiently flexible for most applications.

5.8.2 Tournament Selection

The other alternative to strict fitness-proportional selection is *tournament selection*, in which a set of τ chromosomes are chosen and compared, the best one being selected for parenthood. This approach has similar properties to linear ranking for $\tau = 2$. It is easy to see that the best string will be selected every time it is compared, while the median string will be chosen with probability $2^{-(\tau-1)}$. Thus the selection pressure is given by $\phi = 2^{\tau-1}$, which for $\tau = 2$ is similar to linear ranking when $\alpha \to 0$.

One potential advantage of tournament selection over all other forms is that it only needs a preference ordering between pairs or groups of strings, and it can thus cope with situations where there is no formal objective function at all—in other words, it can deal with a purely *subjective* objective function!

However, tournament selection is also subject to arbitrary stochastic effects in the same way as roulette-wheel selection—there is no guarantee that every string will appear in a given cycle. Indeed, using sampling with replacement there is a probability of approximately $e^{-1} (\approx 0.368)$ that a given string will not appear at all. One way of coping with this, at the expense of a little extra computation, is to use a variance reduction technique from simulation theory. Saliby [110] distinguishes between the *set* effect and the *sequence* effect in drawing items from a finite population. In applying his ideas here, we know that we need τ items to be drawn M times, so we simply construct τ random permutations[5] of the numbers $1, \ldots, M$—the indices of the individuals in the population. These are concatenated into one long sequence which is then chopped up into M pieces, each containing the τ indices of the individuals to be used in the consecutive tournaments. If M is not an exact multiple of τ, there is the small chance of some distortion where the permutations join, but this is a relatively minor problem.

[5] There is a simple algorithm for doing this efficiently—see Nijenhuis and Wilf [111], for example, or look at the Stony Brook Algorithm Repository [112].

5.9 Crossover

Crossover is simply a matter of replacing some of the genes in one parent by the corresponding genes of the other. Suppose we have two strings a and b, each consisting of six variables, i.e.,

$$(a_1, a_2, a_3, a_4, a_5, a_6) \quad \text{and} \quad (b_1, b_2, b_3, b_4, b_5, b_6),$$

which represent two possible solutions to a problem. One-point crossover (1X) has been described earlier in the context of a binary alphabet. (Note that we have chosen here to leave the alphabet unspecified, to emphasize that binary representation is not a critical aspect of GAs.) Two-point crossover (denoted by 2X) is very similar: two crosspoints are chosen at random from the numbers $1, \ldots, 5$, and a new solution produced by combining the pieces of the original 'parents'. For instance, if the crosspoints were 2 and 4, the 'offspring' solutions would be

$$(a_1, a_2, b_3, b_4, a_5, a_6) \quad \text{and} \quad (b_1, b_2, a_3, a_4, b_5, b_6)$$

A similar prescription can be given for m-point crossover where $m > 1$.

An early and thorough investigation of multipoint crossovers is that by Eshelman et al. [113], who examined the biasing effect of traditional one-point crossover and considered a range of alternatives. Their central argument is that two sources of bias exist to be exploited in a genetic algorithm: *positional* bias and *distributional* bias. One-point crossover has considerable positional bias, in that it relies on the building-block hypothesis, and if this is invalid, the bias may prevent the production of good solutions.

On the other hand, 1X has no distributional bias, in that the crossover point is chosen randomly using the uniform distribution. But this lack of bias is not necessarily a good thing, as it limits the exchange of information between the parents. In [113], the possibilities of changing these biases, in particular by using multipoint crossover, were investigated and empirical evidence strongly supported the suspicion that one-point crossover is not the best option. In fact, despite some ambiguities, the evidence seemed to point to an 8-point crossover operator as the best overall, in terms of the number of function evaluations needed to reach the global optimum, averaged over a range of problem types.

Another obvious alternative, which removes any bias, is to make the crossover process completely random—the so-called uniform crossover. This can be seen most easily by observing that a crossover operator itself can be written as a binary string or *mask*—in fact, when implementing crossover in a computer algorithm, this is the obvious way to do it. For example, the mask

```
1 1 0 0 1 1
```

represents the 2-point crossover used above, where a 1 means that the alleles are taken from the first parent, while a 0 means they come from the second.

By generating the pattern of 0s and 1s stochastically (using a Bernoulli distribution) we thus get uniform crossover (UX), which might generate a mask such as

```
1 0 1 0 0 1
```

which implies that the 1st, 3rd and 6th alleles are taken from the first parent, the others from the second. This idea was first used by Syswerda [114], who implicitly assumed the Bernoulli parameter $p = 0.5$. Of course, this is not necessary: we can bias UX towards one or the other parent by choosing p appropriately.

De Jong and Spears [115] produced a theoretical analysis that was able to characterize the amount of disruption introduced by a given crossover operator exactly. In particular, the amount of disruption in UX can be tuned by choosing different values of p.

Of course, there are also many practical considerations that influence the implementation of crossover. How often do we apply it? Some always do, others use a stochastic approach, applying crossover with a probability $\chi < 1$. Do we generate one offspring or two? In many cases there are natural 'twin' offspring resulting, but in more sophisticated problems it may be that only one offspring arises. When we choose only one from two, how do we do it? In accordance with the stochastic nature of the GA, we may well decide to choose either of the offspring at random. Alternatively, we could bias the decision by making use of some other property such as the fitness of the new individuals or the loss (or gain) in diversity that results in choosing one rather than the other.

Booker [116] reported significant gains from using an adaptive crossover rate: the rate was varied according to a characteristic called *percent involvement*. This is simply the percentage of the current population that is producing offspring—too small a value is associated with loss of diversity and premature convergence.

5.9.1 Non-linear Crossover

In cases of non-linear encodings, crossover has to be reinterpreted. One of the most frequently occurring problems is where the solution space is the space of permutations (Π_l) of the numbers $1,\ldots,l$—well-known examples of this include many scheduling problems, and the famous travelling salesman problem (TSP).

For instance, the simple-minded application of 1X with crosspoint $X = 2$ in the following case produces an infeasible solution:

```
P1   1 6 3 4 5 2        O1   1 6 1 2 6 5
        X
P2   4 3 1 2 6 5        O2   4 3 3 4 5 2
```

If this represents a TSP, the first offspring visits cities 1 and 6 twice, and never gets to cities 3 or 4. A moment's thought is enough to realize that this type of behaviour will be the rule, not an exception. Clearly we need to think of something rather smarter if we are to be able to solve such problems.

One of the first ideas for such problems was the PMX (partially mapped crossover) operator [94], which operates as follows: Two crossover points are chosen uniformly at random between 1 and l. The section between these points defines

an interchange mapping. Thus, in the example above, PMX (with crosspoints X=2 and Y=5) might proceed as follows:

```
P1   1 6 3 4 5 2        O1   3 5 1 2 6 4
        X     Y
P2   4 3 1 2 6 5        O2   2 1 3 4 5 6
```

Here the crossover points X and Y define an interchange mapping

$$3 \leftrightarrow 1 \quad 4 \leftrightarrow 2; \quad 5 \leftrightarrow 6$$

on their respective strings, which means that the cut blocks have been swapped and now appear in different contexts from before. Another possibility is to apply a binary mask, as in linear crossover, but with a different meaning. Such a mask, generated as with UX, say, might be the following

```
1 0 1 0 0 1
```

which is applied to the parents in turn. First the components corresponding to 1s are copied from one parent, and then those that correspond to 0s are taken in the order they appear from the second parent in order to fill the gaps. Thus the above example generates the following pairs of strings:

```
P1   1 6 3 4 5 2  ->  1 _ 3 _ _ 2     O1   1 4 3 6 5 2

P2   4 3 1 2 6 5  ->  4 _ 1 _ _ 5     O2   4 6 1 3 2 5
```

5.10 Mutation

First we note that in the case when crossover-OR-mutation is used, we must first decide whether any mutation is carried out at all. Assuming that it is the concept of mutation is even simpler than crossover, and again, this can easily be represented as a bit-string, so we generate a mask such as

```
0 1 0 0 0 1
```

using a Bernoulli distribution at each locus—with a small value of p in this case. (The above example would then imply that the 2nd and 6th genes are assigned new allele values.) However, it appears that there are variant ways of implementing this simple idea that can make a substantial difference to the performance of a GA. The naive idea would be to draw a random number for *every* gene in the string and compare it to μ, but this is potentially expensive in terms of computation if the strings are long and the population is large. An efficient alternative is to draw a random variate from a Poisson distribution with parameter λ, where λ is the average number of mutations per chromosome. A common value for λ is 1—in other words, if l is the string length, the (bit-wise) mutation rate is $\mu = 1/l$, which as early as 1966 [118] was shown to be in some sense an 'optimal' mutation rate. If our Poisson

random draw proposes that there are (say) m mutations, we draw m random numbers (without replacement) uniformly distributed between 1 and l in order to specify the loci where mutation is to take place.

In the case of binary strings, mutation simply means complementing the chosen bit(s). More generally, when there are several possible allele values for each gene, if we decide to change a particular allele, we must provide some means of deciding what its new value should be. This could be a random choice, but if (as in some cases) there is some ordinal relation between allele values, it may be more sensible to restrict the choice to alleles that are close to the current value or at least to bias the probability distribution in their favour.

It is often suggested that mutation has a somewhat secondary function, that of helping to preserve a reasonable level of population diversity—an insurance policy which enables the process to escape from sub-optimal regions of the solution space, but not all authors agree. Proponents of evolutionary programming ([119], for example), consider crossover to be an irrelevance, and mutation plays the major role. The balance between crossover and mutation is often a problem-specific one, and definite guidelines are hard to give.

However, several authors have suggested some type of adaptive mutation: for example, Fogarty [120] experimented with different mutation rates at different loci. Reeves [100] varied the mutation probability according to the diversity in the population (measured in terms of the coefficient of variation of fitnesses). More sophisticated procedures are possible, and anecdotal evidence suggests that many authors use some sort of diversity maintenance policy. In this connection, it should also be mentioned that there is interest currently in 'parameter-less' GAs. It is impossible to eliminate all parameter values, of course, but there has always been interest in some sort of adaptation as the search proceeds, not only for mutation rates but also for other parameters, such as population size. Eiben et al. [121] summarize some of the recent work in this area.

Finally, it should be no surprise that the values of different parameters interact with each other, in terms of the overall performance of the GA. For example, choosing a high selection pressure may mean that we also need a high mutation rate in order to avoid premature convergence. De Jong [43] has an extensive discussion on such matters.

5.11 New Population

Holland's original GA assumed a *generational* approach: selection, recombination and mutation were applied to a population of M chromosomes until a new set of M individuals had been generated. This set then became the new population. From an optimization viewpoint this seems an odd thing to do—we may have spent considerable effort obtaining a good solution, only to run the risk of throwing it away and thus preventing it from taking part in further reproduction. For this reason, De Jong [5] introduced the concepts of *élitism* and *population overlaps*. His ideas

are simple—an élitist strategy ensures the survival of the best individual so far by preserving it and replacing only the remaining $(M-1)$ members of the population with new strings. Overlapping populations take this a stage further by replacing only a fraction G (the *generation gap*) of the population at each generation. Finally, taking this to its logical conclusion produces the so-called steady-state or incremental strategies, in which only one new chromosome (or sometimes a pair) is generated at each stage. Davis [32] gives a good general introduction to this type of GA.

Slightly different strategies are commonly used in the ES community, which traditionally designates them either λ, μ or $\lambda + \mu$. In the first case, $\mu (> \lambda)$ offspring are generated from λ parents, and the best λ of these offspring are chosen to start the next generation. For the $+$ strategy, μ (not necessarily $> \lambda$) offspring are generated and the best λ individuals are chosen from the combined set of parents and offspring.

In the case of incremental reproduction it is also necessary to select members of the population for deletion. Some GAs have assumed that parents are replaced by their children. Many implementations, such as Whitley's GENITOR [109], use the tactic of deleting the worst member(s) of the population, although (as Goldberg and Deb [122] have pointed out) this exerts a very strong selective pressure on the search, which may need fairly large populations and high mutation rates to prevent a rapid loss of diversity. A milder prescription is to select from the worst $p\%$ of the population (for example, Reeves [100] used $p = 50$, i.e., selection from those worse than the median). This is easily implemented when rank-based selection is used. Yet another approach is to base deletion on the *age* of the strings.

5.11.1 Diversity Maintenance

As hinted above, one of the keys to good performance (in nature as well as in GAs) is to maintain the diversity of the population as long as possible. The effect of selection is to reduce diversity, and some methods can reduce diversity very quickly. This can be mitigated by having larger populations or by having greater mutation rates, but other techniques are also often employed.

A popular approach, commonly linked with steady-state or incremental GAs, is to use a 'no-duplicates' policy [32]. This means that the offspring are not allowed into the population if they are merely clones of existing individuals. The downside, of course, is the need to compare each current individual with the new candidate, which adds to the computational effort needed—an important consideration with large populations. (In principle, some sort of 'hashing' approach could be used to speed this process up, but whether this has ever been tried is not clear.)

We can of course take steps to reduce the chance of cloning before offspring are generated. For instance, with 1X, the two strings

```
1 1 0 1 0 0 1
1 1 0 0 0 1 0
```

will generate only clones if the crossover point is any of the first three positions. Booker [116] suggested that before applying crossover, we should examine the selected parents to find suitable crossover points. This entails computing an 'exclusive-OR' (XOR) between the parents, so that only positions between the outermost 1s of the XOR string (the 'reduced surrogate') should be considered as crossover points. Thus in the example above, the XOR string is

```
0 0 0 1 0 1 1
```

so that, as previously stated, only the last three crossover points will give rise to a different string.

5.12 Representation

As remarked in Section 5.1, the focus in this handbook is on using GAs as optimizers in a search space, given a suitable encoding and fitness function. We now consider how the search space $\mathscr{S}$ might be constructed in some generic cases.

5.12.1 Binary Problems

In some problems a binary encoding might arise naturally. Consider the operational research problem known as the *knapsack* problem, stated as follows.

Example 1 (The 0-1 knapsack problem) A set of n items is available to be packed into a knapsack with capacity C units. Item i has value v_i and uses up c_i units of capacity. Determine the subset I of items which should be packed in order to maximize

$$\sum_{i \in I} v_i$$

such that

$$\sum_{i \in I} c_i \leq C.$$

If we define

$$x_i = \begin{cases} 1 \text{ if item } i \text{ is packed} \\ 0 \text{ otherwise} \end{cases}$$

the knapsack problem can be re-formulated as an *integer program*:

$$\text{maximize } \sum_{i=1}^{n} x_i v_i,$$

$$\text{such that } \sum_{i=1}^{n} x_i c_i \leq C,$$

from which it is clear that we can define a solution as a binary string of length n. In this case there is thus no distinction between genotype and phenotype.

However, such problems are not necessarily easy to solve with a GA. In this case, the presence of constraints is likely to cause difficulties—two feasible parents may not produce feasible offspring, unless special crossover operators are constructed. In fact, such problems as these are really subset selection problems, which are best tackled by other means [123], despite the seductiveness of the binary encoding. It is now widely recognized that 'natural' binary encodings nearly always bring substantial problems for simple GAs.

5.12.2 Discrete (but Not Binary) Problems

There are cases in which a discrete alphabet of higher cardinality than 2 might be appropriate. The rotor-stacking problem, as originally described by McKee and Reed [124], is a good example.

Example 2 A set of n rotors are available, each of which has k holes drilled in it. The rotors have to be assembled into a unit by stacking them and bolting them together, as in Figure 5.4. Because the rotors are not perfectly flat, stacking them in different orientations will lead to assemblies with different characteristics in terms of deviations from true symmetry, with the consequent effect (in operation) that the assembled unit will wobble as it spins. The objective is to find which of all the possible combinations of orientations produce the least deviation.

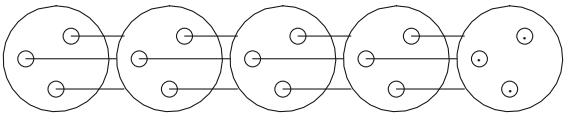

Fig. 5.4 Rotor-stacking problem with $n = 5$ rotors and $k = 3$ holes.

In this case a k-ary coding is natural. A solution is represented by a string of length n, each gene corresponding to a rotor and the alleles, drawn from $\{1,\ldots,k\}$, representing the orientation (relative to a fixed datum) of the holes. Thus, the string (1322) represents a solution to a 4-rotor problem where hole 1 of the first rotor is aligned with hole 3 of the second and hole 2 of the third and fourth. Of course, it would be possible to encode the alleles as binary strings, but there seems little point in so doing—particularly if k is not a power of 2, as there will then be some binary strings that do not correspond to any actual orientation.

This seems very straightforward, although there is a subtle point that could be overlooked. The assignment of labels to the holes is arbitrary, and this creates the

problem of 'competing conventions' as it has been called[6]. For example, given a natural order for labelling each rotor, the string (3211) represents the same solution as (1322). This can be alleviated in this case by fixing the labelling for one rotor, so that a solution can be encoded by a string of length $(n-1)$.

As far as the operators are concerned, standard crossovers can be used here, but mutation needs some careful consideration in the case of k-ary coding, as outlined in Section 5.10.

5.12.3 Permutation Problems

There are also some problems where the 'obvious' choice of representation is defined, not over a set, but over a permutation. The TSP is one of many problems for which this is true. As another example, consider the permutation flowshop sequencing problem (PFSP).

Example 3 Suppose we have n jobs to be processed on m machines, where the processing time for job i on machine j is given by $p(i,j)$. For a job permutation $\{\pi_1, \pi_2, \ldots, \pi_n\}$, we calculate the completion times $C(\pi_i, j)$ as follows:

$$\begin{aligned} C(\pi_1,1) &= p(\pi_1,1) \\ C(\pi_i,1) &= C(\pi_{i-1},1) + p(\pi_i,1) \quad \text{for } i = 2,\ldots,n \\ C(\pi_1,j) &= C(\pi_1,j-1) + p(\pi_1,j) \quad \text{for } j = 2,\ldots,m \\ C(\pi_i,j) &= \max\{C(\pi_{i-1},j), C(\pi_i,j-1)\} + p(\pi_i,j) \\ \text{for } i = 2,\ldots,n;\ & j = 2,\ldots,m \end{aligned}$$

The PFSP is then to find a permutation π^* in the set of all permutations Π such that

$$f(\pi^*) \leq f(\pi) \quad \forall \pi \in \Pi.$$

(Several performance measures $f(\cdot)$ are possible; common ones are the maximum or mean completion time.)

Here the natural encoding (although not the only one) is simply the permutation of the jobs as used to calculate the completion times. So the solution (1462537), for example, simply means that job 1 is first on each machine, then job 4, job 6, etc.

Unfortunately, the standard crossover operators patently fail to preserve the permutation except in very fortunate circumstances, as discussed in Section 5.9.1. Some solutions to this problem were outlined there; more comprehensive discussion of possible methods of attack is contained in [126, 127], while [100, 128] describe some approaches of particular relevance to the PFSP.

[6] This phenomenon is a common one whenever the coding function $c(\cdot)$ is not injective. It has been observed in problems ranging from optimizing neural nets to the TSP. Radcliffe, who calls it 'degeneracy' [125], has presented the most thorough analysis of this problem and how to treat it.

5.12.4 Non-binary Problems

In many cases the natural variables for the problem are not binary, but integer or real-valued. In such cases a transformation to a binary string is required first. (Note that this is a different situation from the rotor-stacking example, where the integers were merely labels: here the values are assumed to be meaningful as numbers.) While the main thrust of metaheuristics research and application is directed to discrete optimization, it is perhaps appropriate to mention these other problems here.

Example 4 It is required to maximize

$$f(x) = x^3 - 60x^2 + 900x + 100$$

over the search space $\mathscr{X} = \{x : x \in \mathbb{Z}; x \in \{0, 31\}\}$, i.e., the solution x^* is required to be an integer in the range $[0, 31]$.

To use the conventional form of genetic algorithm here, we would use a string of 5 binary digits with the standard binary to integer mapping, i.e., $(0,0,0,0,0) = 0, \ldots, (1,1,1,1,1) = 31$. Of course, in practice we could solve such a problem easily without recourse to encoding the decision variable in this way, but it illustrates neatly the sort of optimization problem to which GAs are often applied. Such problems assume first that we know the domain of each of our decision variables, and second that we have some idea of the precision with which we need to specify our eventual solution. Given these two ingredients, we can determine the number of bits needed for each decision variable and concatenate them to form the chromosome. More information on this topic can be found in [12].

5.13 Random Numbers

As GAs are stochastic in nature, it is clear that a reliable random number source is very important. Most computer systems have built-in `rand()` functions, and that is the usual method of generating random numbers. Not all random number generators are reliable, however, as Ross [129] has pointed out, and it is a good idea to use one that has been thoroughly tested, such as those described in the *Numerical Recipes* series [130].

5.14 Conclusions

While this exposition has covered the basic principles of GAs, the number of variations that have been suggested is enormous. Probably everybody's GA is unique! Many variations in population size, in initialization methods, in fitness definition, in selection and replacement strategies, in crossover and mutation are

obviously possible. Some have added information such as age, or artificial tags, to chromosomes; others have allowed varying population sizes or induced the formation of multiple populations in 'niches'. It is in the nature of GAs that parallel processing can often be used to advantage, and here again, there are many possibilities, ranging from simple parallelization of function evaluations to very sophisticated implementations that add a spatial aspect to the algorithm.

The GA community has yet to reach a consensus on any of these things, and in the light of the NFLT, this is perhaps not surprising. However, some ideas do emerge as a reasonable set of recommendations. From a practitioner's viewpoint, Levine made the following observations:

1. A steady-state (or incremental) approach is generally more effective and efficient than a generational method.
2. Don't use simple roulette-wheel selection. Tournament selection or SUS is better.
3. Don't use one-point crossover. UX or 2X should be preferred.
4. Make use of an adaptive mutation rate—one that is fixed throughout the search (even at $1/l$) is too inflexible.
5. Hybridize wherever possible; don't use a GA as a black box, but make use of any problem-specific information that you have.

   Not everyone will agree with this particular list, and there is a conflict inherent in the first two points, since SUS functions best in a generational setting. Broadly speaking, however, it is one with which many researchers would be comfortable. Two other points could be added:

6. Make diversity maintenance a priority.
7. Don't be afraid to run the GA several times.

Why this last point? Statements are frequently made that GAs can find global optima. Well, they can—but usually they tend to converge to some other 'attractor'. In fact, there is some evidence [131] that even with very large populations the attractors are a subset of the local optima relating to a neighbourhood search. With practical population sizes, the attractors may not even be restricted to such a set and may be some distance from global optimality. It thus makes sense to explore several alternatives.

References

1. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, Michigan; re-issued by MIT Press (1992) (1975)
2. Rechenberg, I.: Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipen der biologischen Evolution. 2nd edn. 1993 Frommmann-Holzboog Verlag, Stuttgart (1973)
3. Schwefel, H-P.: Numerische Optimierung von Computer-modellen mittels der Evolutionsstrategie. Birkhäuser, Basel. (English edn. Numerical Optimization of Computer Models, John, Chichester, (1981) (1977)

4. Fogel, D.B.: Evolutionary Computation: The Fossil Record. IEEE Press, Piscataway, NJ (1998)
5. De Jong, K.A.: An Analysis of the Behavior of a Class of Genetic Adaptive Systems. Doctoral dissertation, University of Michigan, Ann Arbor, Michigan (1975)
6. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, Massachusetts (1989)
7. De Jong, K.A.: Genetic algorithms are NOT function optimizers. In: [Whitley, L.D. (ed.): Foundations of Genetic Algorithms 2. Morgan Kaufmann, San Mateo, CA (1993)], pp. 5–18 (1993)
8. Lin, S.: Computer solutions of the traveling salesman problem. Bell Systems Tech. J. **44**, pp. 2245–2269 (1965)
9. Roberts, S.M., Flores, B.: An engineering approach to the travelling salesman problem. Man. Sci. **13**, 269–288 (1966)
10. Nugent, C.E., Vollman, T.E., Ruml, J.E.: An experimental comparison of techniques for the assignment of facilities to locations. Oper. Res. **16**, 150–173 (1968)
11. Reeves, C.R.: Genetic algorithms for the Operations Researcher. INFORMS J Comput. **9**, 231–250 (1997)
12. Reeves, C.R., Rowe, J.E.: Genetic Algorithms—Principles and Perspectives. Kluwer, Norwell, MA (2002)
13. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**, 67–82 (1997)
14. Reeves, C.R., Wright, C.C.: Genetic algorithms and the design of experiments. In: Davis, L.D. De Jong, K.A. Vose, M.D., Whitley, L.D. (eds.) Evolutionary Algorithms: IMA Volumes in Mathematics and its Applications, vol. 111, 207–226 Springer, New York (1999)
15. Macready, W.G., Wolpert, D.H.: Bandit problems and the exploration/exploitation tradeoff. IEEE Trans. Evol. Comput. **2**, 2–13 (1998)
16. Stephens, C.R., Zamora, A., Wright, A.H.: Perturbation theory and the renormalization group in genetic dynamics. In: [Wright, A.H., et al. (eds.): Foundations of Genetic Algorithms 8, LNCS 3469. Springer, Berlin (2005)], pp. 192–214 (2005)
17. Mitchell, M., Holland, J.H., Forrest, S.: When will a genetic algorithm outperform hill climbing? In: Cowan, J.D., Tesauro, G., Alspector, J. (eds.) (1994) Advances in Neural Information Processing Systems 6, Morgan Kaufmann, San Mateo, CA (1994)
18. Vose, M.D.: Modeling simple genetic algorithms. In [Whitley, L.D. (ed.): Foundations of Genetic Algorithms 2. Morgan Kaufmann, San Mateo, CA (1993)], CA, 63–73 (1993)
19. Whitley, D.: An executable model of a simple genetic algorithm. In: [Whitley, L.D. (ed.): Foundations of Genetic Algorithms 2. Morgan Kaufmann, San Mateo, CA (1993)], pp. 45–62 (1993)
20. Vose, M.D.: A closer look at mutation in genetic algorithms. Ann. Math. AI **10**, 423–434 (1994)
21. Vose, M.D., Wright, A.H.: Stability of vertex fixed points and applications. In: [Whitley, D., Vose, M. (eds.): Foundations of Genetic Algorithms 3. Morgan Kaufmann, San Mateo, CA (1995)], pp. 103–113 (1995)
22. De Jong, K.A., Spears, W.M., Gordon, D.F.: Using Markov chains to analyze GAFOs. In: [Whitley, D., Vose, M. (eds.): Foundations of Genetic Algorithms 3. Morgan Kaufmann, San Mateo, CA (1995)], pp. 115–137 (1995)
23. Shapiro, J.L., Prügel-Bennett, A., Rattray, M.: A statistical mechanics formulation of the dynamics of genetic algorithms. Lecture Notes in Computer Science **65**, Springer Berlin, 17–27 (1994)
24. Peck, C.C., Dhawan, A.P.: Genetic algorithms as global random search methods: An alternative perspective. Evolutionary Computation **3**, 39–80 (1995)
25. Reeves, C.R.: Predictive measures for problem difficulty. In: Proceedings of 1999 Congress on Evolutionary Computation, IEEE Press, 736–743 (1999)

26. Reeves, C.R.: Genetic algorithms and neighbourhood search. In: Fogarty, T.C., (ed.) Evolutionary Computing: AISB Workshop, Leeds, UK, April 1994; Selected Papers, Springer, Berlin, 115–130 (1994)
27. Jones, T.C.: Evolutionary Algorithms, Fitness Landscapes and Search. Doctoral dissertation, University of New Mexico, Albuquerque, NM (1995)
28. Culberson, J.C.: Mutation-crossover isomorphisms and the construction of discriminating functions. Evol. Comput. **2**, 279–311 (1995)
29. Stadler, P.F., Wagner, G.P.: Algebraic theory of recombination spaces. Evol. Comput. **5**, 241–275 (1998)
30. Reidys, C.M., Stadler, P.F.: Combinatorial landscapes. SIAM Review **44**, 3–54 (2002)
31. Reeves, C.R.: Fitness landscapes and evolutionary algorithms. In: Fonlupt, C., Hao, J-K., Lutton, E., Ronald, E., Schoenauer, M., (eds.) Artificial Evolution: 4th European Conference; Selected Papers. Springer, Berlin, 3–20 (2000)
32. Davis, L. (ed.): Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York (1991)
33. Chambers, L. (ed.): Practical Handbook of Genetic Algorithms: Applications, Volume I. CRC Press, Boca Raton, Florida (1995)
34. Chambers, L. (ed.): Practical Handbook of Genetic Algorithms: New Frontiers, Volume II. CRC Press, Boca Raton, Florida (1995)
35. Alander, J.T.: An Indexed Bibliography of Genetic Algorithms in Operations Research. ftp://garbo.uwasa.fi/cs/report94-1/gaORbib.pdf. last accessed 16 May 2009 (2008)
36. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs, 3rd edn. Springer, Berlin (1996)
37. Reeves, C.R. (ed.): Modern Heuristic Techniques for Combinatorial Problems. Blackwell Scientific Publications, Oxford, UK; re-issued by McGraw-Hill, London, UK (1995) (1993)
38. Mitchell, M.: An Introduction to Genetic Algorithms. MIT Press, Cambridge, MA (1996)
39. Falkenauer, E.: Genetic Algorithms and Grouping Problems. John Wiley & Sons, Chichester (1998)
40. Bäck, Th.: Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford University Press, Oxford (1996)
41. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing, 2nd edn. Springer, Berlin (2007)
42. Spears, W.M.: Evolutionary Algorithms: the Role of Mutation and Recombination. Springer, New York (2000)
43. De Jong, K.A.: Evolutionary Computation. MIT Press, Cambridge, MA (2006)
44. Vose, M.D.: The Simple Genetic Algorithm: Foundations and Theory. MIT Press, Cambridge, MA (1999)
45. Schmitt, L., Nehaniv, C.L., Fujii, R.H.: Linear analysis of genetic algorithms. Theor. Comput. Sci. **200**, 101–134 (1998)
46. Schmitt, L.: Theory of genetic algorithms. Theor. Comput. Sci. **259**, 1–61 (2001)
47. Grefenstette, J.J. (ed.): Proceedings of an International Conference on Genetic Algorithms and their applications. Lawrence Erlbaum Associates, Hillsdale, NJ (1985)
48. Grefenstette, J.J. (ed.): Proceedings of the 2nd International Conference on Genetic Algorithms. Lawrence Erlbaum Associates, Hillsdale, NJ (1987)
49. Schaffer, J.D. (ed.): Proceedings of 3rd International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1989)
50. Belew, R.K., Booker, L.B. (eds.): Proceedings of 4th International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1991)
51. Forrest, S. (ed.): Proceedings of 5th International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1993)
52. Eshelman, L.J. (ed.): Proceedings of 6th International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1995)
53. Bäck, Th. (ed.): Proceedings of 7th International Conference on Genetic Algorithms. Morgan Kaufmann, San Francisco, CA (1997)

54. Angeline, P.J. (ed.): Proceedings of the 1999 Congress on Evolutionary Computation. IEEE Press, Piscataway, NJ (1999)
55. Zalzala, A. (ed.): Proceedings of the 2000 Congress on Evolutionary Computation. IEEE Press, Piscataway, NJ (2000)
56. Kim, J-H. (ed.): Proceedings of the 2001 Congress on Evolutionary Computation. IEEE Press, Piscataway, NJ (2001)
57. Banzhaf, W., et al. (eds.): Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999). Morgan Kaufmann, San Francisco (1999)
58. Whitley, D., et al. (eds.): Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2000). Morgan Kaufmann, San Francisco (2000)
59. Spector, L., et al. (eds.): Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001). Morgan Kaufmann, San Francisco, CA (2001)
60. Langdon, W.B., et al.: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002). Morgan Kaufmann, San Francisco, CA (2002)
61. Cantú-Paz, E., et al.: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003), LNCS 2723/2724. Springer, Berlin (2003)
62. Deb, K., et al. (eds.): Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004), LNCS 3102/3103. Springer, Berlin (2004)
63. Beyer, H-G., et al.: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2005). ACM Press, New York (2005)
64. Cattolico, M. (ed.): Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2006). ACM Press, New York (2006)
65. Lipson, H. (ed.): Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007). ACM Press, New York (2007)
66. Ryan, C., et al.: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2008). ACM Press (2008)
67. Schwefel, H-P., Männer, R. (eds.): Parallel Problem-Solving from Nature. Springer, Berlin (1991)
68. Männer, R., Manderick, B. (eds.): Parallel Problem-Solving from Nature, 2. Elsevier Science Publishers, Amsterdam (1992)
69. Davidor, Y., Schwefel, H-P., Männer, R. (eds.): Parallel Problem-Solving from Nature, 3. Springer, Berlin (1994)
70. Voigt, H-M., et al. (eds.): Parallel Problem-Solving from Nature, 4, LNCS 1141. Springer, Berlin (1996)
71. Eiben, A.E., et al. (eds.): Parallel Problem-Solving from Nature, 5, LNCS 1498. Springer, Berlin (1998)
72. Schoenauer, M., et al. (eds.): Parallel Problem-Solving from Nature, 6, LNCS 1917. Springer, Berlin (2000)
73. Merelo Guervs, J.J., et al. (eds.): Parallel Problem-Solving from Nature, 7, LNCS 2439. Springer, Berlin (2002)
74. Yao, X., et al. (eds.): Parallel Problem-Solving from Nature, 8, LNCS 3242. Springer, Berlin (2004)
75. Runarsson, T.P., et al. (eds.): Parallel Problem-Solving from Nature, 9, LNCS 4193. Springer, Berlin (2006)
76. Rudolph, G., et al.: Parallel Problem-Solving from Nature, 10, LNCS 5199. Springer, Berlin (2008)
77. Albrecht, R.F., Reeves, C.R., Steele, N.C. (eds.): Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms. Springer, Vienna (1993)
78. Pearson, D.W., Steele, N.C., Albrecht, R.F. (eds.): Proceedings of the 2nd International Conference on Artificial Neural Networks and Genetic Algorithms. Springer, Vienna (1995)
79. Smith, G.D., Steele, N.C., Albrecht, R.F. (eds.): Proceedings of the 3rd International Conference on Artificial Neural Networks and Genetic Algorithms. Springer, Vienna (1997)

80. Dobnikar, A., Steele, N.C., Pearson, D.W., Albrecht, R.F. (eds.): Proceedings of the 4th International Conference on Artificial Neural Networks and Genetic Algorithms. Springer, Vienna (1999)
81. Kůrková, V., Steele, N.C., Neruda, R., Kárný, M. (eds.): Proceedings of the 5th International Conference on Artificial Neural Networks and Genetic Algorithms. Springer, Vienna (2001)
82. Pearson, D.W., Steele, N.C., Albrecht, R.F. (eds.): Proceedings of the 6th International Conference on Artificial Neural Networks and Genetic Algorithms. Springer, Vienna (2003)
83. Ribeiro, B., Albrecht, R.F., Steele, N.C., Dobnikar, A., Pearson, D.W., Steele, N.C. (eds.): Proceedings of the International Conference on Adaptive and Natural Computing Algorithms. Springer, Vienna (2005)
84. Beliczynski, B., Dzielinski, A., Iwanowski, M., Ribeiro, B. (eds.): Adaptive and Natural Computing Algorithms; Proceedings of ICANNGA 2007, LNCS 4431/4432. Springer, Berlin (2007)
85. Rawlins, G.J.E. (ed.): Foundations of Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1991)
86. Whitley, L.D. (ed.): Foundations of Genetic Algorithms 2. Morgan Kaufmann, San Mateo, CA (1993)
87. Whitley, D., Vose, M. (eds.): Foundations of Genetic Algorithms 3. Morgan Kaufmann, San Mateo, CA (1995)
88. Belew, R.K., Vose, M.D. (eds.): Foundations of Genetic Algorithms 4. Morgan Kaufmann, San Francisco, CA (1997)
89. Banzhaf, W., Reeves, C.R. (eds.): Foundations of Genetic Algorithms 5. Morgan Kaufmann, San Francisco, CA (1999)
90. Martin, W.N., Spears, W.M. (eds.): Foundations of Genetic Algorithms 6. Morgan Kaufmann, San Francisco, CA (2001)
91. De Jong, K.A., Poli, R., Rowe, J.E. (eds.): Foundations of Genetic Algorithms 7. Morgan Kaufmann, San Francisco, CA (2003)
92. Wright, A.H., et al. (eds.): Foundations of Genetic Algorithms 8, LNCS 3469. Springer, Berlin (2005)
93. Stephens, C.R., et al. (eds.): Foundations of Genetic Algorithms 9, LNCS 4436. Springer, Berlin (2007)
94. Goldberg, D.E.: Optimal Initial Population Size for Binary-Coded Genetic Algorithms. TCGA Report 85001. University of Alabama, Tuscaloosa (1985)
95. Goldberg, D.E.: Sizing populations for serial and parallel genetic algorithms. In: Schaffer, J.D. (ed.): Proceedings of 3rd International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA, pp. 70–79 (1989)
96. Grefenstette, J.J.: Optimization of control parameters for genetic algorithms. IEEE-SMC **16**, 122–128 (1986)
97. Schaffer, J.D., Caruana, R.A., Eshelman, L.J., Das, R.: A study of control parameters affecting online performance of genetic algorithms for function optimization. In: [Schaffer, J.D. (ed.): Proceedings of 3rd International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1989)], pp. 51–60 (1989)
98. Reeves, C.R.: Using genetic algorithms with small populations. In: [Forrest, S. (ed.): Proceedings of 5th International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1993)], pp. 92–99 (1993)
99. Rees, J., Koehler, G.J.: An investigation of GA performance results for different cardinality alphabets. In: Davis, L.D., De Jong, K.A., Vose, M.D., Whitley, L.D. (eds.): Evolutionary Algorithms: IMA Volumes in Mathematics and its Applications, Vol. 111, Springer, New York, pp. 191–206 (1998) (1999)
100. Reeves, C.R.: A genetic algorithm for flowshop sequencing. Comput. Oper. Res. **22**, 5–13 (1995)
101. Ahuja, R.K., Orlin, J.B.: Developing fitter GAs. INFORMS J. Comput. **9**, 251–253 (1997)

102. Kapsalis, A., Smith, G.D., Rayward-Smith, V.J.: Solving the graphical steiner tree problem using genetic algorithms. J. Oper. Res. Soc. **44**, 397–406 (1993)
103. Levine, D.: GAs: A practitioner's view. INFORMS J. Comput. **9**, 256–257 (1997)
104. Aytug, H., Koehler, G.J.: New stopping criterion for genetic algorithms. Eur. J. Oper. Res. **126**, 662–674 (2000)
105. Greenhalgh, D., Marshall, S.: Convergence criteria for genetic algorithms. SIAM J. Comput. **30**, 269–282 (2000)
106. Baker, J.E.: Reducing bias and inefficiency in the selection algorithm. In: [Grefenstette, J.J. (ed.): Proceedings of the 2nd International Conference on Genetic Algorithms. Lawrence Erlbaum Associates, Hillsdale, NJ (1987)], 14–21 (1987)
107. Lohr, S.L.: Sampling: Design and Analysis. Duxbury Press, Pacific Grove, CA (1999)
108. Hancock, P.J.B.: An empirical comparison of selection methods in evolutionary algorithms. In: Fogarty, T.C. (ed.) Evolutionary Computing: AISB Workshop, Leeds, UK, April 1994; Selected Papers, Springer, Berlin, pp. 80–94 (1994)
109. Whitley, D.: The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In: [Schaffer, J.D. (ed.): Proceedings of 3rd International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1989)], pp. 116–121 (1989)
110. Saliby, E.: Descriptive sampling: A better approach to Monte Carlo simulation. J. Oper. Res. Soc. **41**, 1133–1142 (1990)
111. Nijenhuis, A., Wilf, H.S.: Combinatorial Algorithms for Computers and Calculators. Academic, New York (1978)
112. Skiena, S.S.: The Stony Brook Algorithm Repository. http://www.cs.sunysb.edu/∼algorith/major_section/1.3.shtml. last accessed 18 August 2010 (2000)
113. Eshelman, L.J., Caruana, R.A., Schaffer, J.D.: Biases in the crossover landscape. In: [Schaffer, J.D. (ed.): Proceedings of 3rd International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1989)], pp. 10–19 (1989)
114. Syswerda, G. Uniform crossover in genetic algorithms. In: [Schaffer, J.D. (ed.): Proceedings of 3rd International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1989)], 2–9 (1989)
115. De Jong, K.A., Spears, W.M.: A formal analysis of the role of multi-point crossover in genetic algorithms. Ann. Math. *AI* **5**, 1–26 (1992)
116. Booker, L.B.: Improving search in genetic algorithms. In: Davis, L. (ed.) (1987) Genetic Algorithms and Simulated Annealing. Morgan Kauffmann, Los Altos, CA, pp. 61–73 (1987)
117. Goldberg, D.E., Lingle, R.: Alleles, loci and the traveling salesman problem. In: [Grefenstette, J.J. (ed.): Proceedings of an International Conference on Genetic Algorithms and their applications. Lawrence Erlbaum Associates, Hillsdale, NJ (1985)], pp. 154–159 (1985)
118. Bremermann, H.J., Rogson, J., Salaff, S.: Global properties of evolution processes. In: Pattee, H.H. (ed.) Natural Automata and Useful Simulations, Spartan Books, Washington DC, pp. 3–42 (1966)
119. Fogel, D.B.: An overview of evolutionary programming. In: Davis, L.D., De Jong, K.A., Vose, M.D., Whitley, L.D. (eds.) Evolutionary Algorithms: IMA Volumes in Mathematics and its Applications, Vol. 111, Springer, New York, pp. 89–109 (1999)
120. Fogarty, T.C.: Varying the probability of mutation in the genetic algorithm. In: [Schaffer, J.D. (ed.): Proceedings of 3rd International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1989)], pp. 104–109 (1989)
121. Eiben, A.E., Schut, M.C., de Wilde, A.R.: Is self-adaptation of selection pressure and population size possible? A case study. In: [Runarsson, T.P., et al. (eds.): Parallel Problem-Solving from Nature, 9, LNCS 4193. Springer, Berlin (2006)], pp. 900–909 (2006)
122. Goldberg, D.E., Deb, K.: A comparative analysis of selection schemes used in genetic algorithms. In: [Rawlins, G.J.E. (ed.): Foundations of Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1991)], pp. 69–93 (1991)

123. Radcliffe, N.J., George, F.A.W.: A study in set recombination. In: [Forrest, S. (ed.): Proceedings of 5th International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1993)], pp. 23–30 (1993)
124. McKee, S., Reed, M.B.: An algorithm for the alignment of gas turbine components in aircraft. IMA J. Math. Manag. **1**, 133–144 (1987)
125. Radcliffe, N.J., Surry, P.D.: Formae and the variance of fitness. In: [Whitley, D., Vose, M. (eds.): Foundations of Genetic Algorithms 3. Morgan Kaufmann, San Mateo, CA (1995)], pp. 51–72 (1995)
126. Fox, B.R., McMahon, M.B.: Genetic operators for sequencing problems. In: [Rawlins, G.J.E. (ed.): Foundations of Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1991)], pp. 284–300 (1991)
127. Poon, P.W., Carter, J.N.: Genetic algorithm crossover operators for ordering applications. Comput. Oper. Res. **22**, 135–147 (1995)
128. Reeves, C.R., Yamada, T.: Genetic algorithms, path relinking and the flowshop sequencing problem. Evol. Comput. **6**, 45–60 (1998)
129. Ross, P. `srandom()` anomaly. Genetic Algorithms Digest, http://www.aridolan.com/maillists/mlframes_ns.html. last accessed 18 August 2010, **11:23** (1997)
130. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes in C : The Art of Scientific Computing. Cambridge University Press, Cambridge, UK (1992)
131. Reeves, C.R.: The 'crossover landscape' and the Hamming landscape for binary search spaces. In: [De Jong, K.A., Poli, R., Rowe, J.E. (eds.): Foundations of Genetic Algorithms 7. Morgan Kaufmann, San Francisco, CA (2003)], pp. 81–97 (2002)