INFORMS Journal on Computing
Vol. 9, No. 3, Summer 1997

0899-1499/97/0903-0231 $05.00

FEATURE ARTICLE

Genetic Algorithms for the Operations Researcher

COLIN R. REEVES / *School of Mathematical and Information Sciences, Coventry University, Priory St., Coventry CV1 5FB, United Kingdom, Email: CRReeves@coventry.ac.uk*

(Received: July 1995; revised: April 1997; accepted: May 1997)

Genetic algorithms have become increasingly popular as a means of solving hard combinatorial optimization problems of the type familiar in operations research. This feature article will consider what genetic algorithms have achieved in this area, discuss some of the factors that influence their success or failure, and offer a guide for operations researchers who want to get the best out of them.

In the last 15 years, the growth in interest in heuristic search methods for optimization has been quite dramatic. Having once been something of a Cinderella in the field of optimization, heuristics have now attained considerable respect and are extremely popular. There are several reasons for this (as discussed in [116], for example) which lie outside the scope of this feature article. However, the reader may easily confirm that in nearly every issue of any journal in the field of operations research (OR), there will be several papers describing a new application, or a new development, of a heuristic approach. One of the most interesting developments is in the application of genetic algorithms (GAs). While other methods, such as simulated annealing (SA) or tabu search (TS), have become widely known within the OR community, these methods remain relatively unknown to a wider public. However, GAs have been regularly featured in newspaper articles and TV programs (at least in Europe). It is not possible to give exact figures, but it would appear that the level of funding for GA-based projects by national governments and by supranational government organizations, such as the European Commission, far exceeds any funding given to projects based on methods such as SA or TS. For example, there is a European Union-funded network of excellence (EVONET) to promote awareness and applications of GAs and associated techniques, with an initial budget of more than $200,000. No such program exists in Europe for any other heuristic methodology.

This presents us with a paradox: GAs are more widely known outside the OR community than other techniques such as SA or TS, yet within the OR world the position seems reversed. For this reason, it seems timely to consider the role that GAs can play in OR in some detail, and this feature article will endeavor to explain what GAs are, and how they work; to show how they can be adapted to deal with different OR problems; to give some examples of successful OR applications; and finally, to describe their relationship to more familiar search techniques.

Subject classifications: Genetic algorithms.
Other key words: optimization, heuristics.

1. Background

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

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

The general problem is to find

$$\min_{x \in \mathcal{X}} f.$$

Here, $\mathbf{x}$ is a vector of *decision variables*, and f is the *objective function*. In this feature article, it has been assumed the problem is one of minimization, but the modifications necessary for a maximization problem will be obvious. Otherwise, this formulation is very general, and many techniques have been proposed for solving such a problem, usually relying on some specific properties of the variables and functions under consideration. The most well-known of such cases is undoubtedly linear programming, where the search space $\mathcal{X}$ is a polyhedron formed by the intersection of hyperplanes, and the objective function is a linear function of the decision variables. There are not many classes of problem for which there exist techniques that always find a globally optimal $\mathbf{x}$, and for the general case portrayed above, it is usual to resort to heuristic methods.

Recent developments in heuristic methodology have seen the rise of so-called *metaheuristics*. These are distinguished by their relative freedom from dependence on problem-specific characteristics, and include such approaches as SA, TS, perturbation methods, and GAs. A recent survey of some of these methods may be found in [121]. Genetic algorithms were developed by Holland and his associates at the University of Michigan in the 1960s and 1970s, and the first full, systematic (and mainly theoretical) treatment was contained in Holland's book *Adaptation in Natural and Artificial Systems*[72] published in 1975. The popularity of GAs in OR applications is relatively recent, and it is the purpose of

this feature article to review the principles and practice of GAs in the particular context of OR.

2. Brief Introduction

One of the distinctive features of GAs is the way they work with an encoded representation of the variables **x**, instead of **x** itself. To continue the formalization of the first section, we define a string **s** of symbols drawn from an alphabet $\mathcal{A}$ to be a mapping

$$c : \mathcal{A}^l \mapsto \mathcal{X},$$

where l is the length of the string. The value of l depends on the dimensions of both $\mathcal{X}$ and $\mathcal{A}$. In Holland's original description, the strings were binary (i.e., $\mathcal{A} = \{0, 1\}$), although, more recently, there has been much debate as to whether the extensive use of binary coding is necessary, or even desirable. This debate has not really been resolved, and a full discussion in this feature article of its current status would be of limited value. Goldberg[53] puts the argument in favor of binary coding, but Antonisse[5] has cast doubt on this view. Radcliffe[107] also argues strongly against the necessity of the binary-coding perspective in practical problem-solving. At any rate, the GA is normally still thought of as a search method that works at the level of a string **s** which represents **x** in some way, whether this is a binary coding or some other form of representation, although this should not rule out using **x** directly.

It is usually desirable that c should be a bijective function. (For a formal definition of this term, readers should consult a standard textbook on algebra, such as [89]. For the purposes of this paper, the important feature of a bijective mapping 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**.) Later we shall discuss cases where the nature of this mapping itself creates difficulties for a GA in solving OR problems.

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 $f(c(\mathbf{s}))$; more generally, fitness is $g(f(c(\mathbf{s})))$ where $g : \mathbb{R} \mapsto \mathbb{R}$ is a monotonic function.

Continuing the biological analogy, the individual symbols in a particular string are often called *genes* and the symbols of the alphabet from which the genes are drawn are called *alleles*. A distinction is also drawn between the *genotype*, which in terms of the above definitions is the string **s**, and the *phenotype*, which is the decoded vector **x**.

```
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
```

Figure 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 (typically, randomized rules) under which crossover and mutation are performed, a new population is created (typically, when a fixed number of offspring have been generated), and whereby the whole process is terminated (typically, by a limit on the total number of offspring generated).

Crossover is simply a matter of replacing some of the genes in one parent by the corresponding genes of the other. One-point crossover, for example, would be the following. Given the parents P1 and P2, with crossover point ×, 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 1.

Holland's explanation of why it is advantageous to search the space $\mathcal{A}^l$ rather than $\mathcal{X}$ hinges on three main ideas. Central to this understanding is the concept of a *schema*. A schema is a subset of the space $\mathcal{A}^l$ in which all the strings share a particular set of defined values. This can be represented by using the alphabet $\mathcal{A} \cup *$; 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 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* (or *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 $\mathcal{O}(M^3)$ schemata. The second concept is expressed

by the so-called *Schema Theorem*, in which Holland showed that if there are $N(S, t)$ instances of a given 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

$$E[N(S, \quad + 1)] \geq \frac{F(S, t)}{\bar{F}(t)} N(S, t)\{1 - \epsilon(S, t)\}$$

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

Somewhat extravagant statements have been made on the basis of this theorem along the lines 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 then only for one generation. Thus, any attempt to extrapolate this result for more than one generation is doomed to failure because the terms are then not independent of what is happening in the rest of the population. Also, given the finite population size, it is clear that any exponential increase will rather soon hit a natural ceiling! What can be said is that certain schemata that are resistant to destruction by crossover and mutation, and that are fitter than the average of the current population, are likely to increase their presence in the next population.

By writing the theorem in the form of a lower bound, Holland was able to make a statement about schema S which 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[143] and Whitley.[147]

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[53] calls this the building-block hypothesis, and a substantial part of recent theoretical research into GAs has centered on the question of the circumstances under which this hypothesis fails. As will be discussed later, its failure can create severe difficulties in attempting the solution of OR problems.

Attempts are continuing to construct better mathematical models of a GA in order to characterize more explicitly what is happening to a particular string or schema. Work by Vose[143] and Whitley[147] has allowed the construction of Markov chain models which potentially, at least, should also enable us to gain a better understanding of the GA as a whole. However, the Achilles heel of these approaches (as shown by De Jong, Spears, and Gordan,[31] for example) is the huge computational effort that is needed even to solve small problem instances with simple GAs.

We point out that the original motivation for studying GAs was not optimization as such, but their utility as a general model of adaptive systems. In a sense, their use as optimizers is a by-product that has now become almost the dominant purpose in using them. Recently, De Jong[30]—whose earlier work initiated the application of GAs to optimization—has suggested that the original concept of a GA has changed (mutated?) to a point where it is almost unrecognizable, and should not be given the same name. He prefers the term GA-based function optimizer. However, GA is still the term used by a large majority of researchers and rather than confuse the issue, GA is the term that we use in this feature article.

3. Strategic Considerations

The "plain vanilla" GA outlined in the previous section can be extended and modified in a variety of different ways in order to improve its performance as an optimizer. Before discussing GAs in the particular context of OR, it is appropriate to consider some of these strategic decisions in more detail. Those readers wishing a more comprehensive account of these matters are referred to the books by Goldberg,[53] Davis,[27] and Reeves.[116] Many other applications and examples can be found in the various conference proceedings [3, 10, 25, 36, 41, 43, 60, 62, 90, 100, 111, 112, 125, 131, 142, 148] listed at the end of this article. The two volumes edited by Chambers[18, 19] contain some useful practical case studies. Another excellent source for GA literature is the University of Vaasa bibliography maintained by Alander.[2] This is available by electronic means (`ftp` and `http`) both in a general form and in a version that deals specifically with OR applications. The OR version alone cites more than 300 papers.

3.1 Population Strategy

In any application of a GA, the question of population size N_{pop} arises. Small populations run the risk of failing to cover the solution space adequately, whereas large populations may incur a heavy computational burden without making acceptable progress toward a high-quality solution in a reasonable amount of time. Early attempts to capture this tradeoff[50, 54] suggested "optimal" population sizes which grew exponentially with l. Later work[56] has (fortunately!) concluded that populations of this size are not needed. Many reported implementations seem to produce satisfactory results with populations having as few as 30 strings, although values of 50 or 100 are perhaps more common. Some work reported in [117] helps to explain why small populations are adequate, at least for binary codings.

3.2 Reproductive Strategy

Holland's canonical GA assumes replacement of the whole population en bloc at each generation. 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[28] introduced the concepts of *elitism* and *population overlaps*. His ideas are simple—an elitist strategy ensures the survival of the best individual so far by preserving it and replacing only the remaining ($N_{pop} - 1$) 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[27] gives a good general introduction to this type of GA.

Selection schemes also have to be considered. The original roulette-wheel method simply generates a probability distribution for selection in which the selection probability of a given string is proportional to its fitness. Pseudo-random numbers are then used to choose strings for parenthood. This approach has a high 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*[6] is a particularly effective way of realizing this outcome. It can be thought of as a circle divided into sectors, one for each chromosome, such that the sector area is proportional to the fitness of the chromosome. Attached to this circle is an equally spaced multi-armed spinner. Spinning the wheel produces simultaneously the values N_C for all the chromosomes in the population. Recent experimental work by Hancock[64] clearly demonstrates the superiority of this approach. It is somewhat disappointing, therefore, to find that much published work still appears to rely on the roulette-wheel method.

An associated problem is that of finding a suitable measure of fitness for the members of the population. Simply using the objective function values $f(x)$ is rarely sufficient, because the scale on which $f(x)$ is measured matters. (For example, values of 10 and 20 are much more clearly distinguished than 1010 and 1020.) Further, if the objective is minimization rather than maximization, a transformation is clearly required.

Some sort of scaling is thus usually applied, and Goldberg[53] gives a simple algorithm to deal with both minimization and maximization. The method is cumbersome, however, and the need for continual rescaling as the search progresses is irksome. Two alternatives provide more elegant solutions. *Ranking* the chromosomes in fitness order loses some information, but there is a gain in ease of selection when the mechanism uses some function of the ranks to choose parents (a linear function is often used, as in [120]). The other approach is *tournament selection*,[55] in which a set of T chromosomes is chosen and compared, the best one being selected for parenthood. When $T = 2$, this approach has similar properties to linear ranking, and, in general, tournament selection can be shown to be a form of ranking in expectation.

In the case of incremental reproduction it is also necessary to select members of the population for deletion. Many implementations, such as Whitley's GENITOR,[144] use the tactic of deleting the worst member(s) of the population, although (as Goldberg and Deb[57] 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 α% of the population (for example, Reeves[120] used $\alpha = 50$, 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.

3.3 Recombination Strategy

The original one-point crossover operator is not necessarily the best approach to take for every problem. Of course, it can be generalized to a multipoint operator, where r crossover points are chosen randomly. In fact, it can be further generalized by making r a random variable as well and simply copying a given gene from the first parent with probability p and from the second parent with probability $(1 - p)$. The case $p = 0.5$ is commonly called *uniform crossover*. Several of these operators have received a helpful theoretical treatment by De Jong and Spears,[29] who have characterized precisely the amount of disruption introduced by a given crossover operator.

The question of how often crossover should be applied has also been investigated experimentally. Many applications assume crossover is carried out stochastically, where the probability of applying crossover to any particular pair of strings (previously selected for parenthood) is a value $P_c < 1$. De Jong's experiments[28] suggested that a value $P_c = 0.6$ was appropriate. Grefenstette[61] proposed 0.95, while Schaffer et al.[126] suggested it should vary with N_{pop} and l. From a more traditional optimization perspective, it is perhaps difficult to see why P_c should ever be anything less than 1, at least in the absence of a mutation, because it would seem more important to explore a new area of the search space rather than to risk remaining at the same place. Opinion differs as to the importance of this. Some practitioners, such as Davis,[27] advocate a policy of avoiding duplicates (i.e., revisiting strings already seen), but others would justify multiple visits using schema-processing arguments. Certainly, there are cases where allowing $P_c < 1$ enables a global optimum to be found, but $P_c = 1$ does not, as is shown by Whitley,[147] for example.

Some effort has also gone into trying to decide how often mutation should be applied. As in the case of crossover, it is common to specify a value P_m which defines the probability with which each gene is to be mutated. This value is often called the *mutation rate*, and the procedure is carried out in practice by generating pseudo-random numbers. Different optimal mutation rates have been reported, but again there is no real consensus. Perhaps the most common approaches are either to use a small mutation probability (e.g., $P_m = 0.01$), or to use a value $P_m = 1/l$, because there is some theoretical and experimental evidence (see [65]) that this is a suitable value for some problems. Of course, there is no reason why the mutation rate should remain constant for all generations, or indeed at all gene positions (or *loci*) in the string. Fogarty[40] addresses these issues experimentally for some fairly small problems. His work suggests that reducing the value of P_m with time improves performance. However,

Davis[27] advocates increasing the rate with which mutation is applied as the search progresses (while correspondingly reducing the frequency of application of crossover). He also describes several heuristics for implementing his idea. Hesser and Männer[68] derive a complex formula for P_m at each locus that depends not only on N_{pop} and l, but also on the relative diversity of alleles at that locus.

4. Problem Categories

Before proceeding to examining GAs for OR in detail, it is important to distinguish between two categories of optimization problems, depending on whether the decision variables are continuous or discrete.

As formulated in Section 1, GAs work on a discrete encoding (genotype) of the underlying parameters of a problem (phenotype), so there are some particular questions to be addressed if GAs are to be used for solving problems in the continuous domain. A large body of experimental evidence now exists (see [82], for example) that shows that GAs can be used very successfully for problems of this type. On the other hand, in problems of *combinatorial* optimization where the decision variables are naturally discrete, the traditional GA faces difficulties of a different kind.

Most of the remainder of this feature article will concentrate on combinatorial optimization problems (COPs). There are two main reasons for this. First, many OR problems (although by no means all) fall naturally into this class. Second, optimization of problems in continuous variables is often more naturally handled using the related but slightly different approach of the *evolution strategy*. However, for the sake of completeness, a brief discussion of the continuous domain is given here.

4.1 Coding for the Continuous Domain

In order to apply the GA approach to optimization problems in the continuous domain, the problem must first be discretized by encoding each component of the vector $\mathbf{x}$ as a binary string. These components are then concatenated in order to obtain a chromosome. This procedure assumes that we know some other parameters—the limits of the range within which each variable will lie, and the degree of precision that will ensure that a global optimum can be found. For example, if the global optimum lies in the hyperplane where the first decision variable (i.e., component of $\mathbf{x}$) is $x_1 = 0$, but we have imposed limits of $1 \leq x_1 \leq 2$, it is self-evident that the global optimum will never be found. Similarly, if the number of bits used to encode each string is too small, the consequent resolution of the search pattern may be inadequate for locating the global optimum.

In practice, it is debatable whether we always *do* have knowledge of these parameters. In fact, a poor choice of values can have an important influence on the effectiveness and efficiency of a genetic search. The question of choosing appropriate values for these parameters has been addressed by several GA researchers who have advocated a strategy of adaptively sizing the range and/or the degree of precision as the search progresses [129, 132, 145]. However, there is not much evidence that these techniques are routinely used to solve optimization problems by GAs.

On the assumption that a binary representation is to be used, some authors (e.g., Caruana and Schaffer[17]) have suggested that a Gray code would be more appropriate than the standard binary code. The standard binary code has the unfortunate property that adjacent values in the continuous domain may be maximally separated in the discrete binary domain. For example, consider the integers 31 and 32, encoded by a 6-bit binary string. The binary equivalents of these numbers are (011111) and (100000). Gray codes have the property that adjacent values in the continuous domain remain minimally separated in the discrete domain. (There are many Gray codes; in the most commonly used version, 31 and 32 would be represented by (010000) and (110000), respectively.) However, although there are undoubtedly cases where this improves performance, it is not hard to show problems in which Gray coding actually makes a problem more difficult for a GA (Reeves and Wright[122] provide a simple example).

4.2 The Evolution Strategy Approach

Before leaving the topic of optimization in the continuous domain, we should mention an alternative approach to the traditional GA, known as the evolution strategy (ES), that is well-suited to continuous optimization. Whereas this approach often works well in the continuous domain, it is not necessarily better in general than a GA, although it does avoid the necessity of choosing an appropriate encoding.

The ES approach was developed independently from Holland's GA by Rechenberg[113] and Schwefel[130] in Germany. Instead of using a discrete encoding, the ES works directly with the decision variables themselves. The basic ES is described by two parameters μ and λ. A population consists of μ solutions to an optimization problem, from which two parents are chosen. These parents are used to construct a new solution by taking combinations of their respective decision variables. This then undergoes mutation, such as a Gaussian-distributed move away from the existing solution. The procedure is repeated λ ($\geq \mu$) times, and a new population is selected using either the offspring alone (the (μ, λ) strategy), or from the complete set of solutions (the $(\mu + \lambda)$ strategy). The size of the Gaussian step is parameterized by a variable σ. The question of an appropriate value for σ can be neatly handled by making it an auxiliary decision variable. This is then treated in the same way as the real decision variables. Many other variants are possible, but space limitations preclude a full discussion here. For further details, Hoffmeister and Bäck[69] give a good introduction to the ES approach.

5. GAs for Combinatorial Problems

In this section, we provide examples that illustrate the application of GAs to typical OR problems. Each problem has different characteristics in terms of its natural encoding.

5.1 Binary Coding: Graph Bisection Problem

Many combinatorial problems can be formulated in terms of a 0–1 integer programming problem. For such problems, the

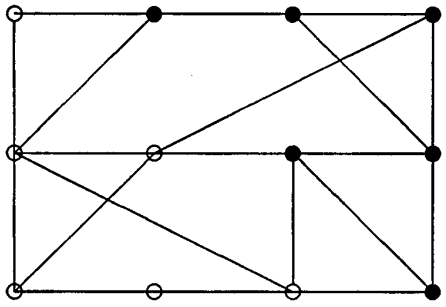

Figure 2. Graph bisection. In this example, each node belongs to one of two subsets as indicated.

application of a GA would appear to be straightforward, as the encoding of a solution as a binary string seems obvious.

Graph bisection is a case in point. The problem is as follows: given a graph $G = (V, E)$ with a cost function $c : E \subseteq V \times V \mapsto \mathbb{R}$ and a size $S(u)$ attached to each vertex, partition the nodes of G into two subsets of equal size (we assume $|V|$ is even) such that the total cost of the edges cut is minimized. Figure 2 provides an illustration.

The coding of the graph bisection problem for a GA is particularly simple. A solution can be represented by a binary string of length $|V|$, each bit (gene) corresponding to one vertex, the allele denoting the subset (0 or 1) to which the vertex belongs. For example, in a 6-vertex problem, the string (010101) allocates vertices {1, 3, 5} to one subset, and vertices {2, 4, 6} to the other. However, difficulties will arise once the GA operators are applied. It is clear that one-point crossover (and indeed other more sophisticated versions) may generate offspring that do not represent balanced solutions. Even though both parents represent balanced subsets, there is no guarantee that their children will do so. Further, the same solution may be represented by different strings. For example, the string (101010) represents the same solution to the 6-vertex problem as (010101).

Even this relatively easy problem demonstrates two of the recurring difficulties in applying GAs to COPs. In the first place, many problems have constraints that may not need to be made explicit in traditional approaches—for example, using neighborhood search methods, it is easy to maintain a balanced partition by a suitable choice of neighborhood. With a GA however, this balance constraint needs to be considered more carefully. One solution is to allow unbalanced partitions, but to penalize them in assigning fitness (i.e., the function g referred to in Section 2 includes a term for violation of the balance constraint). This raises the question of how to choose an appropriate penalty. Happily, this choice is straightforward in the case of graph bisection. Lee, Park, and Kim[86] show how it is possible to transform the above graph bisection problem on G into a related one on another graph G^* so that optimal (balanced) solutions for G will correspond to the optimal solutions for G^*. The GA can then be applied to the problem on G^*, and the balance constraint will take care of itself.

The second difficulty is the presence of what is variously

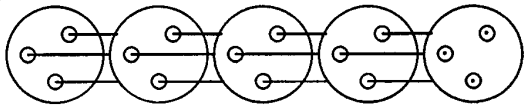

Figure 3. Rotor stacking problem.

described in the literature as redundancy or competing conventions.[128] (Recently, Radcliffe and Surry[110] have argued for the use of the term degeneracy to describe this phenomenon.) The labeling of the subsets in their genetic representation is arbitrary—there is no intrinsic reason why the first subset should be represented by the label 0 and the second by the label 1, rather than the converse. Thus, as we have already seen, every solution to the graph bisection problem (phenotype) can be encoded by two genotypes; in terms of the formal discussion in Section 2, the function c is not bijective. (In particular, for this example, c is not *injective*.) We shall return to this problem later in Section 6.

5.2 *k*-ary Coding: Rotor Stacking

The rotor stacking problem, as originally described by McKee and Reed,[94] is as follows. A set of n rotors is 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 3. 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 produces the least deviation.

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 (132) represents a solution to a 3-rotor problem where hole 1 of the first rotor is aligned with hole 3 of the second and hole 2 of the third. 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 is then the problem of what to do about binary strings that do not correspond to any actual orientation.

This seems very straightforward, but there is a subtle point here that could be overlooked. The assignment of labels to the holes is arbitrary, and this once again creates the problem of competing conventions. This can be alleviated in this case by fixing the labeling 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 crossover can be used here, but mutation needs some careful consideration in the case of k-ary coding. Since there are several possible allele values for each gene, mutation is no longer a simple matter; 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 this example) there is some ordinal relation between allele val-

ues, it may be more sensible to restrict the choice to close neighbors of the current value.

5.3 Sequence-Coding: Traveling Salesman Problem

Given the status of the traveling salesman problem (TSP) as the archetypal NP-hard combinatorial optimization problem, it is not surprising to find that the TSP was one of the earliest OR problems to be attacked by a GA. In this case, the natural coding (although not the only one) is a permutation of the cities. So the solution (1432567) to a 7-city problem represents the order in which the cities are visited—city 1 followed by city 4 followed by city 3, etc.

Unfortunately, the standard crossover operators patently fail to preserve the permutation except in very fortunate circumstances. For instance, the offspring of the parents in the following case are clearly illegal:

```
P1 2 1 3 4 5 6 7   O1 2 1 3 2 7 1 5
         X
P2 4 3 6 2 7 1 5   O2 4 3 6 4 5 6 7.
```

Thus, the question arises: What is it that traditional crossover is doing that a crossover for sequence-encoded strings should try to imitate? Broadening the terms of Holland's schema analysis and the building-block hypothesis, it can be seen that the crossover operator's task is to ensure that good features of the parents can be inherited by the offspring, and to allow small features of parents to join together to become large ones. (In fact, it is possible to extend the concept of a schema to problems of this type in a formal way, as in Goldberg,[53] but the meaningful features for permutation problems depend to a much greater extent on the specific problem than they do in a traditional binary GA.)

A number of different operators have been proposed for dealing with this situation, all of which try to maintain some feature of the parents. The PMX (partially mapped crossover) of Goldberg and Lingle,[51] perhaps the most well-known, works as follows. Two crossover points are chosen, defining an interchange mapping. Thus, in the 7-city example, PMX could give:

```
P1 2 1 3 4 5 6 7   O1 4 1 6 2 7 3 5
       X     Y
P2 4 3 6 2 7 1 5   O2 2 6 3 4 5 1 7.
```

Here, the crossover points X and Y define an interchange mapping {3 ↔ 6, 4 ↔ 2, 5 ↔ 7}. Thus, considering P1, first the positions of cities 3 and 6 are interchanged, then 4 and 2, and finally 5 and 7, in order to create O1. Carrying out the same interchanges on P2 creates O2.

Another operator is what Reeves[116] called C1 crossover. (This has also been used by several other authors, including Smith[133] and Prosser,[104] without apparently acquiring a definitive name. A version of this operator was first introduced by Davis,[26] who called it order crossover. It can be generalized in several different ways, as suggested by Oliver, Smith, and Holland[97] and Davis.[27]) For the basic procedure, a crossover point X is chosen randomly, and the left-hand section of the first parent is copied to the offspring, with the chromosome being completed by taking in order each unassigned element from the second parent. For the 7-city example, the C1 operator could generate the following offspring:

```
P1 2 1 3 4 5 6 7   O1 2 1 4 3 6 7 5
       X
P2 4 3 6 2 7 1 5   O2 4 3 2 1 5 6 7.
```

The rationale for such operators as C1 and PMX is that they preserve the absolute positions in the sequence of elements of one parent, and the relative positions of those from the other. The conjecture is that this allows the offspring to inherit useful features of the chromosome(s) while preserving feasibility.

However, further problems may still occur because of the existence of competing conventions. As discussed earlier, the building-block hypothesis may fail because of redundancy in the ways the problem can be encoded. For example, the permutations {1, 2, 3, 4, 5, 6, 7} and {7, 1, 2, 3, 4, 5, 6} actually represent the same solution, but recombination of these strings with C1 or PMX will create a different solution. Again, it is possible to alleviate some of the effects of this by fixing the first city to be visited. However, for symmetric TSPs, there is still a two-fold redundancy in that tours are the same regardless of the direction of travel (for example, the tour {1, 2, 3, 4, 5, 6, 7} is the same as {1, 7, 6, 5, 4, 3, 2}).

In fact, operators that focus on the *cities* produce fairly unimpressive results. It has often been observed that TSP heuristics perform better when the emphasis is on the *edges*, that is, the city-to-city links. Whitley, Starkweather, and Shaner[146] have found markedly better performance using an edge-recombination operator, which emphasizes the preservation of the adjacencies found in the parents, rather than their actual sequence positions, and thereby avoids such problems as that discussed above. However, for problems such as machine sequencing (see [120]), where actual sequence positions are the important features, PMX and C1 perform much better.

Perhaps because there is so much that seems problem-specific in permutation problems, the invention of recombination operators for these problems has proliferated. Fox and McMahon[44] discuss several such operators, while others may be found in [11, 12, 73, 135]. Poon and Carter[102] have also reported a comparison of permutation operators across a range of COPs. They found consistently good results, relative to the other operators, from an improved implementation of the Union operator of Fox and McMahon.[44] Sequence-based coding is perhaps the most well-researched of any in the area of combinatorial optimization, and operators such as PMX, C1, and Union appear to be fairly robust, and to generate high-quality solutions, for a variety of sequence-coded COPs.

6. The Application of GAs

Because of the relative ease of programming at least a simple GA, it is tempting to apply GAs to almost any optimization problem. A moment's thought should be enough to suggest that GAs are not always appropriate and, in fact, the literature contains many examples of situations where the use of a GA has not proved to be the most sensible approach.

6.1 Inappropriate Use of GAs

An extensive family of optimization techniques has developed from linear programming, and it should be obvious that GAs will not be competitive with these methods when applied to problems for which linear programming is suitable. As another example, fairly simple unimodal functions in the continuous domain are better attacked by classical calculus-based methods, and relatively small discrete optimization problems may also be better approached by hill-climbing techniques.

Perhaps more surprisingly, there are some large NP-hard combinatorial problems for which GAs seem very well suited, but in practice other methods easily outperform them. Zero–one knapsack problems are a good example. The coding is obvious, and infeasible solutions can be penalized in a straightforward way. However, in unpublished work by the author, computational experiments on a set of multiconstrained 0–1 knapsack problem instances involving n objects, a simple GA that was allowed 2^n trials frequently failed to find the optimum even when n was as small as 10. In other words, exhaustive search was nearly always superior. (Such poor results were, in the referees' view, a strong disincentive to publication!) Gordon, Bohm, and Whitley[58] report similarly discouraging results on much larger problem instances, although there was only a single constraint in their cases. In every instance, the GA failed to find a better solution than that obtained within 10 seconds of CPU time by a depth-first branch-and-bound strategy.

Results on the TSP also suggest that other approaches such as iterated Lin–Kernighan,[45, 77] insertion methods,[47] and perturbation methods[92, 152] are usually superior to GAs, although a better understanding of how GAs work can lead to significant improvements. Often, the apparent failure of a GA is due to the use of an inappropriate means of encoding the problem, as is demonstrated in [110].

The size of a given problem instance also seems to influence the ease of solution. Keane[82] describes some experiments on complex engineering problems in the continuous domain where classical methods seem more powerful for small instances, but a GA was superior for larger ones.

The general question of why a given search method performs well for some problems and not for others is one that still awaits an answer, although the recent interest in the analysis of search methods initiated by the no-free-lunch (NFL) theorem of Wolpert and Macready[149] suggests that it may be a vital area of research in the future. For those not familiar with it, a brief discussion on the NFL theorem is included as an appendix to this article. The salient argument of [149] is that, on average, all search algorithms that satisfy certain conditions on how they work perform exactly the same. If there are some problems for which algorithm A is better than B, then there must be problems for which B is better than A. However, in this feature article, we are more interested in how to design a GA that performs well on a particular problem. It is not always easy to predict how a GA will perform in a particular situation. Nevertheless, as we will consider in the next section, there are certain characteristic details of implementation that can be identified as likely to promote good GA performance.

6.2 GA Pathology

Genetic algorithms provide a very general optimization scheme, and it is not surprising that specially tailored algorithms can usually do better on a specific problem. Perhaps the surprise lies in how well GAs can perform, provided that some care is taken to understand the problem and how the selected coding and operators interact. Moreover, if a GA is hybridized with a problem-specific heuristic, or with local search, the resulting performance can be extremely good. Conversely, if the coding or the operators are poorly chosen, a stand-alone GA can perform poorly. The specific performance varies from problem to problem, but there are some general principles that should be adopted in order to avoid failure and promote success. In this section, we address this issue by looking at some general reasons for GA failure.

6.2.1 Coding Problems

Many cases of GA failure can be attributed to a problem in finding a suitable coding. The case of the bin-packing problem[123] provides a good illustration of this difficulty.

In the case of (one-dimensional) bin packing, we have the problem of assigning n objects to bins of identical size in such a way as to minimize the number of bins used. Thus, we could define a binary chromosome of length nq, where q is an upper bound on the optimal number of bins, and

$$x_{ij} = \begin{cases} 1 & \text{if object } i \text{ is in bin } j \\ 0 & \text{otherwise.} \end{cases}$$

This would enable traditional crossover to be used, but at the expense of a rather lengthy chromosome. However, many of the solutions would almost inevitably be infeasible, and penalty functions would be needed.

Another approach would be to define a chromosome of length n^2, where

$$x_{ij} = \begin{cases} 1 & \text{if object } i \text{ is in the same bin as object } j \\ 0 & \text{otherwise.} \end{cases}$$

In other words, the problem is reinterpreted as finding the minimal number of equivalence classes defined by the bin-packing size constraint. There are also clear difficulties with this approach. First, the number of bins implied by each solution is not obvious, and we would need to calculate this number. Second, crossover may destroy the transitivity property of a solution (i.e., $x_{ij} = x_{jk} = 1 \Rightarrow x_{ik} = 1$); this would have to be restored after each recombination. Third, infeasibilities are likely to arise, and fourth, the chromosome will be even longer than in the first approach to coding.

Using a k-ary coding also leads to difficulties. A natural coding would label the bins from 1 to k, and assign each object a bin number. However, even in fairly trivial cases, this may lead to quite severe difficulties. For example, consider the following situation, where we have seven objects:

```
P1 1 2 1 3 2 2 1   O1 1 2 1 3 1 1 4
             X
P2 4 1 4 3 1 1 4   O2 4 1 4 3 2 2 1.
```

Here, P1 and P2 actually represent the same solution (labeling of the bins is arbitrary, given identical bins), each using three bins. Yet, traditional crossover is oblivious to

this fact, and actually creates two different solutions with four bins. Thus, recombination of good solutions can easily lead to inferior ones, while it is equally obvious that recombination of feasible solutions can often lead to infeasible ones.

The same difficulty was evident in all three of our examples in Section 5, that is, a degree of arbitrariness in the labeling of the discrete objects that form the basis of a solution. In more traditional neighborhood search methods that use unary operators, this is not an issue because the particular convention that is used can be fixed at the start and then remain unchanged. However, when we use a binary operator such as crossover, the convention needs to be considered, because it is by no means clear that two genotypes will necessarily follow the same convention, at least in the initial stages. What usually happens is that eventually one convention wins out, but the algorithm may take some time to settle down even when this occurs.

The problems encountered in such cases have received a thorough analysis from Radcliffe,[105, 106] using the concept of *formae*. Whereas a schema defines subsets of chromosomes that are similar in the specific sense that they have the same alleles at some specified loci, a forma defines subsets of chromosomes that are similar in some possibly more general way—usually, in terms of the phenotype rather than in terms only of the genotype. For example, in the case of bin packing, the defining characteristic of a forma might be that a particular group of objects is packed together. Traditional schema analysis is able to explain a posteriori why the simple crossover works. With Radcliffe's approach, it is possible to invert the process and, a priori, construct operators to have desirable properties, including those he calls *respect* and *proper assortment*. Informally, respect implies that when parents share a particular characteristic, their offspring should always inherit this characteristic. Proper assortment implies that when parents have different (but compatible) characteristics, it should at least be possible that their offspring contain both characteristics. (Proper assortment can be interpreted as a more systematic statement of the building-block hypothesis.) These properties are not always compatible. Radcliffe gives a simple example of incompatibility that concerns the TSP: if one parent includes the links (1–2, 2–3) and the other parent includes (1–2, 2–4), then it is clear that although proper assortment would insist on the possibility of the offspring including (3–2, 2–4), this is not possible if we also wish to have respect, because respect implies that the link (1–2) *must* be present.

It may also be that the operators that can be built from the requirements of these operators are computationally more expensive than simple crossover and mutation. In devising operators, it may be convenient to relax proper assortment to a weaker condition that simply requires the eventual possibility of such an offspring arising, even if several recombinations intervene.

6.2.2 Constraints

That many OR problems involve constraints is well known. Genetic algorithms need to be effective in dealing with constraints if they are to perform well in solving these problems. However, there is no single mechanism that has performed consistently well in handling constrained problems with GAs.

The essence of the difficulty is clear: there is rarely a guarantee (at least with simple operators and codings) that two feasible parent solutions will provide feasible offspring. Several solutions to this difficulty have been proposed.

Penalties

The most obvious solution to the problem of constraints is simply to ignore them. If an infeasible solution is encountered, it is given a very low fitness value so that its chance of entering the population (or of remaining, if it does enter) is minimal. Unfortunately, this may still lead to an excessive number of infeasible solutions in the population, and it also fails to recognize that the degree of infeasibility does supply some information. It is common to find the global optimum on or near a constraint boundary, so that solutions that are slightly infeasible may actually be fairly close to the optimum. (This is the argument made, for example, by proponents of the concept of *strategic oscillation* in TS.[49]) Thus, the obvious next step is to modify the objective function by incorporating a penalty term. There are cases in which this works very well, as in the graph bisection problem, where theoretical considerations permit an appropriate choice of penalty. The nature of the constraints is clearly another factor that may influence the performance of a GA. For example, the set-covering problem (which has inequalities) proves to be somewhat easier for a GA (as in Beasley and Chu[9]) than the set-partitioning problem, for which the constraints are all strict equalities.

In fact, a naive attempt to use penalties often fails. If the penalty is too small, many infeasible solutions are allowed to propagate; if it is too large, the search is confined to the interior of the search space, far from the boundaries of the feasible region.

To address this failing, Fairley[37] tried using an adaptive penalty function in a multi-constrained 0–1 knapsack problem. The function was based on an analogy with Lagrangean relaxation, in which penalty factors are chosen with the objective of obtaining a high-quality lower bound to the optimal solution (of a minimization problem). Some very sophisticated procedures have been developed in Lagrangean relaxation methods for adapting the penalty factors (see Beasley[8]). However, a simple heuristic that works quite well increases the penalty for a constraint that is infeasible and decreases it for a constraint that is feasible. This heuristic is easy to implement in a GA framework. Although Fairley's results were better than those obtained using fixed penalties, they were not outstandingly so, and were not as good as those produced by a repair approach. (The concept of repair is described in the next section.)

Another adaptive penalty approach is used by Smith and Tate.[134] This method penalizes solution S by an amount

$$\pi(S) = f(\theta, \bar{\theta})(V_{\text{feas}} - V^*)$$

where V^* is the best (infeasible) solution found, V_{feas} is the current best feasible solution, and θ, $\bar{\theta}$ are measures of the

total infeasibility of the current solution and a solution of moderate infeasibility, respectively. The rationale here is that moderately infeasible solutions will lie near the feasibility boundary and close to optimal solutions. Infeasible solutions of this type should not be prohibited from the population because they have the potential of recombining or mutating to produce an optimal solution. The form of the function f is chosen to allow some infeasibility, although in [134] the exact choice of function was not found to be critical. By setting a single penalty for the entire solution, the problem of adapting penalties to different constraints is avoided. By making the size of the penalty a function of the distance of the current best solution V_{feas} from the unconstrained best solution, the search concentrates on the most promising areas of the parameter space. The choice of the θ measure of infeasibility is very likely to vary from one problem to another, but the results of this approach for the facility layout problems investigated were good, and suggest it is a procedure worthy of more detailed study.

Repair

Another popular solution to GAs for constrained problems is to accept an infeasible solution, but to repair it before inserting it into the population. Some biological justification is often claimed for this approach, along the lines that the infeasible solution should be treated as an immature individual that, although initially demonstrably unfit for its purpose, can nevertheless develop over time into a perfectly reasonable solution. This analogy has also been applied to methods that use the GA offspring as the springboard for a local optimization. Such hybrid techniques have been championed by Davis,[27] and are becoming increasingly popular (as in [140], for example, for the TSP, and in [70] for graph partitioning). Such hybrids have recently received a more formal specification (under the name of memetic algorithms) by Radcliffe and Surry.[109]

Mühlenbein[96] describes such an approach in the context of the TSP. His MPX (maximal preservative crossover), like edge-recombination, tries to ensure that as many of the common edges in two parents as possible are inherited, by first copying a substring of one parent directly to the offspring. Then an attempt is made to construct the tour by adding edges from the other parent, starting from the last city in the copied substring. Unfortunately, by itself, this may lead to infeasible solutions. On detection of incipient infeasibility, MPX next tries edges from the first parent and, if all else fails, it creates a new edge by taking the next available feasible city from the second parent. As Mühlenbein points out, this is essentially a form of a repair procedure. A related procedure is *2-repair* (see [59]), where new edges introduced by MPX are considered for a local search using the well-known two-opt heuristic.

Falkenauer and Delchambre[38] also use a repair mechanism in their treatment of bin-packing problems. Their solution is to focus not on the objects, but on the bins. The chromosome is in two parts. The first part uses a k-ary encoding of the bin identifiers, and the second part encodes the actual bin identifiers that are used. Crossover takes place only on the second part of the chromosome, and normally leads to both underspecification and overspecification with respect to the bin contents. This is dealt with by using a repair mechanism based on the *first-fit descending* heuristic.

In such cases, there is also the practical problem of deciding whether it should be the original offspring or the repaired version that should be used in the next generation. Orvosh and Davis[98] report some interesting experimental evidence which suggests that this decision should itself be a stochastic one.

Multiple Objectives

Many authors have suggested using infeasibility as a secondary objective, thus avoiding the introduction of penalties. Recent work by Chu and Beasley[20] provides a good example of this idea in the context of set partitioning. In addition to the usual fitness measure (based on the true objective function), they define a secondary measure that they call *unfitness* to represent the degree of infeasibility present in a given solution. The selection scheme for reproduction favored strings that were maximally compatible, while selection for deletion was carried out by trying to find an existing population member that was dominated by the new offspring. The operators that they used were standard crossover, mutation, and a repair-type operator based on one the authors developed for the set-covering problem.

Modified Operators

As mentioned in the case of the TSP, one approach to alleviating the difficulties encountered by the standard operators is to design operators that are appropriate for the problem. The edge-recombination operator[146] is a typical example. Rather than aim at preserving city positions in the TSP, the operator lists the *edges* contained in parent strings, and selects edges using an algorithm that tries to ensure that an offspring string contains as few foreign edges (i.e., those contained in neither parent) as possible. Modifications have been described in [93, 135] that improve on the performance of the original operator.

Radcliffe's approach to the construction of operators also takes account of the structure of a specific problem, but tries to do so in a more general way. In [106], he proposes six design principles that are illustrated by his random respectful recombination operator. This works by copying all the shared alleles of both parents into the offspring, which is completed by a random choice over the alleles at each unspecified locus. In a later paper,[108] a random assorting recombination (RAR) operator is described. The RAR was devised to deal with a class of subset-selection problems that give rise to the same sort of difficulties described in Section 6.2.1 in the context of bin packing (another type of subset-selection problem). The RAR has a user-specified parameter w that defines the amount of assortment possible. Parents are compared to identify common included or excluded elements and w copies of these elements are then stored, together with a single copy of each of the other elements and their complements. The offspring is then constructed by drawing items from storage in a random order, accepting or rejecting elements according to the requirements of feasibility, until storage is exhausted or the offspring is complete. If

storage is exhausted first, the offspring has to be completed by a random choice from all elements.

Similar ideas have been used by Höhn and Reeves[71] in the context of graph partitioning. This problem is a natural generalization of graph bisection—in this case the graph has to be partitioned into k subsets, where $k > 2$. The problems of redundancy and feasibility are much more acute, and it becomes a much greater challenge to find suitable operators. A k-ary alphabet was used to denote the subsets, and operators were devised that attempted to fulfill Radcliffe's criteria of strict transmission and respect.

Finally, we should mention the work of Michalewicz and Janikow.[95] This work introduced modified crossover and mutation operators in the context of problems with linear constraints, i.e., generalized transportation problems. First, they recommended using equality constraints for the elimination of some variables, thus reducing the dimensionality of the problem. Second, the remaining constraints are employed to limit the range of mutation for each variable, while recombination is accomplished by taking various convex combinations of the parent strings. However, in [95], the context is not combinatorial problems as defined in this feature article, although it is stated that the approach can be extended to such cases.

Modified Problem Formulation

Another approach that has proved fruitful in some cases is to modify the problem formulation. Reeves[123] found good solutions to bin-packing problems by reformulating the problem. Rather than using the subset-selection interpretation, bin packing was treated as a sequencing problem, with the sequence being decoded by using an on-line heuristic. In computational experiments, Reeves found that using *best-fit* to decode the sequence was superior to *first-fit*. In [123], further improvements were made by exploiting the *population* aspect of a GA to reduce the size of the search space. Information obtained from current solutions was used to hypothesize that certain variables must take particular values in an optimal solution. Such variables can then be fixed in subsequent investigations, so that the size of the remaining problem is reduced. In the context of bin packing, Martello and Toth[91] describe a general procedure that exploits an IP formulation. Reeves used a simpler approach that is straightforward to implement in a GA context. Overall, the results of this approach proved superior in some cases to the mathematical programming-based algorithms described by Martello and Toth.[91]

Modification of the problem formulation has also been used in other applications. Starkweather et al.[135] solved a complex warehouse scheduling problem where the strings encoded the sequence in which tasks were to be presented to a constructive heuristic that produced the actual schedule. Schaffer and Eshelman[127] described a similar approach to an extremely complicated line-balancing problem in the production of printed circuit boards. The GA was used to find a good sequence to feed to a set of heuristics that produced the final solution.

7. Some Examples of Successful GAs

Notwithstanding the examples of how the application of GAs may create real difficulties, there are also numerous examples of the successful application of GAs to COPs. In an article such as this, it is not possible to give an exhaustive survey of relevant applications of GAs, but Table I lists some of the more useful and accessible references that should be of interest to the OR/CS community. Because of the enormous growth in reported applications of GAs, this list inevitably reflects the author's own interests. For a much more complete review, readers are referred to Alander's bibliographies.[2]

7.1 Comparisons

A weakness of many early reports of GA applications (and a frequent criticism of GA research by OR workers) is that few comparisons have been made with other techniques. In fairness to the GA community, we realize that this may often have been for sensible pragmatic reasons. The development of these GA applications was often driven by practitioners whose primary interest was in getting a solution that was better than the current one. Furthermore, the problems may not have been in a suitable form, so that the solution from a GA could not easily be compared to a solution obtained by an exact method. Awareness of other solution methods may also have been absent.

In a brief but thoughtful recent discussion, Dowsland[34] has suggested several other reasons for this lacuna, and proposed that this absence of comparisons with other methods may be one factor in the relatively slow acceptance of GAs among OR researchers and practitioners. However, such comparisons have now started to appear rather more frequently, and what follows is a brief discussion of some areas where comparisons have been made.

Kapsalis, Smith, and Rayward-Smith[81] showed that a GA generated high-quality results for graphical Steiner tree problems. More recently, Esbensen[35] gave details of an improved GA implementation that outperformed two other heuristics that had previously been found to perform well. However, when Julstrom[80] considered the rectilinear version of the Steiner tree problem, his GA was inferior to other methods. Palmer and Kershenbaum[99] compared a GA with another good heuristic for the optimal communication spanning tree problem, and found the GA to be superior.

Houck, Joines, and Kay[74] investigated the application of a GA to large-scale location-allocation problems. In comparison with iterated restart and local search, the GA was clearly superior. Reeves[120] found a fairly simple GA that assumed no problem-specific knowledge beyond the use of a sequence-based crossover performed slightly better than SA, and considerably better than iterated local search, for flowshop sequencing problems. However, results from a TS heuristic (see [115]) were superior to the results from the GA.

Aggarwal, Orlin, and Tai[1] applied a GA to the problem of finding a maximum independent set in an undirected graph. The crossover operator used in their approach was not restricted to generating a single offspring. Instead, their

Table I. Successful Applications of Genetic Algorithms

Bin-packing
- Falkenauer and Delchambre[38]
- Reeves[123]
- Smith[133]

Graph-related
- Höhn and Reeves[71]
- Jones and Beltramo[78]
- von Laszewski[84]
- von Laszewski and Mühlenbein[85]
- Pirkul and Rolland[101]

Location
- Brown, Huntley, and Spillane[14]
- Conway and Venkataramanan[23]
- George[48]
- Houck, Joines, and Kay[74]
- Huntley and Brown[75]
- Palmer and Kershenbaum[99]
- Tam[137]
- Tate and Smith[138]

Scheduling
- Cartwright and Mott[16]
- Cleveland and Smith[22]
- Davis[26]
- Della Croce, Tadei, and Volta[32]
- Dorndorf and Pesch[33]
- Fang, Ross, and Corne[39]
- Kobayashi, Ono, and Yamamura[83]
- Reeves[120]
- Starkweather et al.[135]
- Syswerda and Palmucci[136]
- Yamada and Nakano[151]

Set-related
- Aggarwal, Orlin, and Tai[1]
- Al-Sultan, Hussain, and Nizami[4]
- Beasley and Chu[9]
- Chu and Beasley[20,21]
- Levine[87]
- Liepins and Potter[88]
- Richardson et al.[124]

Steiner tree
- Esbensen[35]
- Hesser, Männer, and Stucky[67]
- Julstrom[80]
- Kapsalis, Smith, and Rayward-Smith[81]

Transport scheduling
- Blanton and Wainwright[12]
- Thangiah, Vinayagamoorthy, and Gubbi[139]
- Wren and Wren[150]

Traveling salesman
- Freisleben and Merz[46]
- Homaifar, Guan, and Liepins[73]
- Jog, Suh, and Van Gucht[76]
- Mathias and Whitley[93]
- Mühlenbein[96]
- Oliver, Smith, and Holland[97]
- Prinetto, Rebaudengo, and Sonza Reorda[103]
- Ulder et al.[140]
- Whitley, Starkweather, and Shaner[146]

operator was constructed to search for the *best* possible offspring. This idea has been suggested before by Reeves,[118] in the sense of finding a local optimum in the reduced search space defined by two parents. Aggarwal, Orlin, and Tai appear to be the first to propose applying an exact algorithm to generate the globally best solution in this search space. The other novel aspect of their approach is to generate a second offspring that is forced to be as different as possible from the first, while remaining in the reduced search space. The results of their GA show it to be one of the best heuristics available for this problem.

The GA of Al-Sultan, Hussain, and Nizami[4] for the set-covering problem found solutions that were equal to or better than those obtained by a Lagrangean heuristic in slightly less computation time. In contrast, Levine[87] obtained much better results for airline crew scheduling (a common application of set covering) from a branch-and-cut algorithm than from a GA. However, the rather different GA approach of Chu and Beasley[20] performed significantly better than Levine's GA. The method of Chu and Beasley was successful in finding good solutions to large crew-scheduling problems obtained from the OR-Library,[7] although its computational efficiency was nevertheless still inferior to traditional mathematical programming techniques.

Chu and Beasley[21] extended their implementation to investigate the generalized assignment problem. In this case, their GA was more successful in finding the optimum than any of nine other heuristics when tested on problem instances ranging from 75 to 4000 decision variables.

Yamada and Nakano[151] carried out comparisons of several GAs and other heuristics for the job-shop scheduling problem. Their GA used an extension of the idea in [118], where a local search is used to find a good solution in the reduced search space defined by two parents. Their results showed that this GA outperformed other GA approaches, and was competitive with other methods based on TS and the shifting bottleneck technique.

What distinguishes most of these approaches is their effective hybridization of some GA concepts with ideas drawn from other areas of heuristic methodology. The algorithms that result are techniques that have the potential to find globally optimal solutions in some cases, although the computational requirements may sometimes be quite large. However, these authors all report that solutions of excellent quality are obtained relatively quickly. Table II records the achievements of some of these algorithms in terms of the largest problems solved to optimality and quoted CPU times.

Of course, mere size is not necessarily the main determinant of how difficult a problem instance is to solve, but it is often the most important single factor. Perhaps not too much should be read into the computer times quoted in Table II either. For one thing, the computer systems are not always completely specified, while in one case the times are estimates based on converting from another machine. Furthermore, in some cases, these times are averages over repeated runs with different random numbers; not all runs found a global optimum, so it may take considerably longer to find

Table II. Size of Some Problem Instances for Which GAs Have Found a Global Optimum

Problem [Reference]	Size	Time	Computer system
Generalized assignment[21]	20 agents, 200 jobs		SGI
Independent set[1]	1035 nodes, 533115 edges		SGI (estimate)
Job-shop scheduling[151]	20 jobs, 10 machines		DEC α
Set covering[9]	1000 rows, 10000 columns		SGI
Traveling salesman[46]	1400 cities		DEC α

a global optimum than stated. On the other hand, it is also true that in some cases a global optimum was actually found relatively early in the run, well before the stopping criterion was satisfied, so it may also take considerably less time than stated in Table II. Bearing these qualifications in mind, it is nonetheless evident from Table II that well-designed GAs can now succeed in being both effective in finding global optima for large COPs, and efficient in their use of computational resources.

7.2 Further Developments

It is probably true that in some of the application areas just described, GAs have not been tested against the most recent version of other metaheuristics such as TS or SA. It is also likely that in many cases the performance of a simple GA for a given problem instance will not be competitive with that of a carefully tailored heuristic. Nonetheless, by combining some GA concepts, such as selection from a population and the combination of elements from several solutions, with other heuristic ideas, it is possible to produce solutions that are superior to those of other heuristics. In many of the papers cited in Table I, success in using GAs comes from two sources. First, it is essential to understand the problem, and thus find an appropriate coding. Second, hybrid techniques usually succeed better than a standard GA—for example, problem-specific operators often give superior results to standard crossovers. There is also another factor that is becoming increasingly relevant as computing hardware continues to develop, and that is the ease with which GAs can be incorporated into a parallel computing environment. Gorges-Schleuter[59] reports on one of the earliest efforts to implement parallel GAs for OR problems, whereas Hauser and Männer[66] provide a brief but useful survey of some of the possibilities for a variety of different multiprocessor systems. Although reports of parallel GA implementations for OR problems are not yet widespread in the literature, it is clear that such developments have great potential.

GAs are ideal candidates for searching for solutions to what are otherwise very difficult problems for more traditional OR techniques. These include stochastic and dynamic COPs. In such cases, it can be argued that the use of a population-based approach is likely to present advantages that point-based methods lack. Some examples of published work in this area include the early paper by Goldberg and Smith[52] on a knapsack problem with an oscillating constraint, and a paper by Reeves[114] on machine sequencing when job processing times are stochastic. Haas et al.[63] used a GA approach to optimize the planning of radiotherapy treatment—a multiobjective problem that gives rise to some very complex modeling, and involves trying to optimize simultaneously the radiation dosage in different locations in the patient's body. A good review of the possibilities for multiobjective GAs, although not in the OR context, is given by Fonseca and Fleming.[42] Some work has been reported on applications of GAs to instances of stochastic, dynamic, and multiobjective OR problems, and there appears to be considerable interest in the GA community in extending this work into other application areas in OR.

8. Relation to Neighborhood Search

Early accounts of GAs (and similar approaches such as the evolution strategy) stress the biological analogy that provided the original motivation. Thus, the impression has too readily been conveyed that GAs are rather different types of search processes from those with which the OR community are more familiar. Vaessens, Aarts, and Lenstra[141] were perhaps the first to put GAs into the context of local search methods, and recent work has started to explore this common ground in more detail. In the process, this has uncovered some of the presuppositions of existing techniques, and suggested alternative ways of proceeding.

The distinctive features of a GA (in contrast to more conventional search methods) would appear to be the use of a population-based rather than point-based search, and the use of the binary crossover operator rather than the unary operators common in the local search literature. On closer examination it can be argued that the first of these differences has its parallels in more traditional multistart heuristic methods. It is the way the population is used to *mix* solutions from different regions of the search space that appears to be the most significant difference from normal local search. The effect of the latter is intuitively clear. Relative to the small steps taken by local search methods, a genetic search takes big jumps around the search space. Just how big these jumps may be is explored by Reeves[118] in the context of a GA in the binary hypercube.

However, recent work has suggested that a fundamental reappraisal of our understanding of search algorithms is needed. Culberson[24] and Jones[79] pointed out that distances in a search space should be measured relative to the operator being used. Each operator induces a different neighborhood graph, and the neighbors of a point under one operator may be radically different from those under another operator. Even more interestingly, Culberson showed

that operators and encodings are intimately related to each other. One operator-encoding pair may be isomorphic to another, in the sense that the induced neighborhood graphs to which they correspond are the same.

Some of these ideas have been exploited by Höhn and Reeves[70] in developing heuristics for the graph-partitioning problem. The conventional crossover operator actually has a dual influence. The first is simply to effect a reduction of the search space, in that (assuming a respectful operator) characteristics which belong to both parents are simply inherited. Having performed this function, crossover then simply becomes just another operator, with the effect of moving to one of the induced neighbors in the reduced search space. Recognizing this dual function opens up two further possibilities: one can investigate alternative ways of reducing the search space (for example, by the use of multiparent strategies), while one can also investigate the effect of using different operators. (This dual influence is implicitly recognized in many other implementations of GAs—the reproduction scheme developed by Tate and Smith[138] for the quadratic assignment problem is one example. However, the full implications of crossover's dual effect have not usually been explored in detail.)

In [70], the second possibility (that of using different operators) is explored. An abstract GA is defined that consists of a population of point-based local searches connected by means of selection and recombination. An individual in the population not only represents a point in the search space, but also encodes a (reduced) subset of the search space in which a given operator is to be applied. In this way, the standard binary crossover operator can be reduced to a unary operator (complementary crossover), and its effects relative to several other operators can be studied. It is also worth pointing out that, in some cases, this approach may enable one to search the neighborhood much more efficiently when the cost of a neighbor can be found by a calculation of lower time complexity than the normal requirement for evaluating the new point from scratch.

Conversely, by fixing the operator, it is possible to compare the effect of recombination to other local-optima-avoiding schemes such as iterated hill climbing. In the context of some instances of graph bisection problems, it was found that application of the Lee–Park–Kim (LPK) operator[86] clearly outperformed the complementary crossover and a simple bit-flipping operator. The way in which recombination affects different operators was also observed—aside from its obvious primary effect of reducing the search space, recombination also provides (for some operators) a secondary effect whereby it is possible to escape from what would otherwise be local optima with respect to the relevant operator. In [70], a further experiment on a larger graph was carried out in which this secondary effect could be seen, in that recombination using the LPK operator outperformed iterated hill climbing using the same LPK operator.

9. Summary

This feature article has attempted to review some of the ideas of genetic algorithms in the context of OR applications. Rather than try to present a taxonomic survey of all GA/OR research, it has been the purpose of this article to point out some of the guiding principles and generic difficulties that need to be taken into account by anyone who wishes to implement a genetic solution to a complex combinatorial problem.

To summarize, GAs have several advantages to offer for such problems; some of their key attractions are as follows.

- *Generality.* Because a GA works on a *coding* of a problem, it is easy to write one general computer program for solving many different optimization problems. However, a specific coding may have a significant impact on the GA's ability to find good solutions, unless the operator used to search the space is carefully selected with respect to the coding.
- *Nonlinearity.* Many conventional optimization techniques rely on unrealistic assumptions of linearity, convexity, and differentiability. None of these is needed by a GA. The only requirement is the ability to calculate some measure of performance, which may be highly complicated and nonlinear. Indeed, the performance measure may not be expressible in a traditional mathematical form at all, as in the interesting case reported by Caldwell and Johnston[15] where the objective function is entirely subjective!
- *Robustness.* Many researchers have observed empirically that, given the right encoding and operators, GAs appear to be inherently robust. Although it is possible to fine-tune a GA to work better on a given problem, it is often true that a wide range of parameter settings (e.g., population sizes, crossover, and mutation rates) will give very acceptable results.
- *Ease of modification.* Even relatively minor modifications to a particular problem may cause severe difficulties to some heuristics. By contrast, it is easy to change a GA to model variations of the original problem.
- *Parallel nature.* Quite apart from the property of intrinsic parallelism (about which there is still some debate), there is great potential for *implementing* GAs in parallel. As more sophisticated computing hardware becomes available, developments in this area will surely become increasingly common.

Despite these important features, it is nearly always sensible to incorporate problem-specific information into a GA, perhaps by developing problem-specific codings or operators, or by hybridizing with other search methods.

The amount of research interest in GAs does not yet appear to have peaked, and there are clearly many inviting avenues still to be explored. Several of these have been mentioned in this feature article, but perhaps the most interesting developments will arise from two areas. The first is a more intensive study of the connections of GAs with other methods of neighborhood search. Second, an exploration of the implications of the NFL theorem would seem to be an area where existing GA theory can be exploited. One area of GA research concerns the investigation of problems that can be expected to be either hard or easy for GAs to solve, and

the NFL results suggest that this is the type of question we should increasingly try to answer, not only for GAs, but for heuristic search methods in general.

Acknowledgment

The useful and constructive comments of Dr. Ed Wasil and of the anonymous referees are gratefully acknowledged.

Appendix
The No-Free-Lunch Theorem

Considerable interest has been aroused by the recent publication by Wolpert and Macready (W&M) of the "No-Free-Lunch" Theorem[149] concerning the performance of algorithms that search for the optimum of a cost function.

Summarizing briefly, what they show is that, averaged over all possible functions, the performance of all search algorithms that satisfy certain conditions is the same. However, the idealized search algorithm they describe is different in some important respects from "real" search algorithms such as SA, TS, or GAs. Nonetheless, W&M frequently claim that these results are relevant to such algorithms, as for example,

> ..., if simulated annealing outperforms genetic algorithms on some [sub]set ϕ [of the set of all functions $\mathcal{F}$], genetic algorithms must outperform simulated annealing on $\mathcal{F}\backslash\phi$.

This is true in terms of what W&M mean by "outperform," but as we shall argue below, their interpretation is somewhat idiosyncratic, and makes certain assumptions that may not be easily verified in practice. In Section A.1, we present the theorem, largely in the notation of [149], which is slightly different from that of the main body of this article.

A.1 The Theorem

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

$$f : \mathcal{X} \mapsto \mathcal{Y} \subset \mathbb{R}.$$

For convenience, the value of $f(\mathbf{x})$ is called the cost of $\mathbf{x}$. The general problem is to find an optimal solution, i.e., a point $\mathbf{x}^* \in \mathcal{X}$ that minimizes or maximizes f.

W&M assume a search algorithm A that has generated a set of m distinct points denoted by $\{d_m^x(i)\}$ and associated cost values $\{d_m^y(i)\}$ where $y = f(\mathbf{x})$ and the index $i = 1, \ldots, m$ implies some ordering (the most obvious one being an ordering with respect to the search chronology). For convenience, the whole ensemble of points and cost values can be denoted by d_m.

Thus initially, starting from a point $\mathbf{x}_1$ with cost y_1, we have $d_1^x(1) = \mathbf{x}_1$ and $d_1^y(1) = y_1$. Subsequent points are generated by A based on d_m; that is, A is a function

$$A : d \mapsto \mathbf{x}, \quad \text{where} \quad \mathbf{x} \in \mathcal{X}\backslash\{d_m^x(i)\}.$$

The information generated by this sequence of points can be encapsulated in a histogram $\vec{c}$ of the cost values $\{d_m^y(i)\}$; the quality of the algorithm's performance can be measured in terms of some characteristic of $\vec{c}$ such as its minimum, mean, or median. For a given f, the quantity of interest is thus the conditional probability $P[\vec{c}|f, m, A]$.

The NFL Theorem. *For any pair of algorithms A_1 and A_2,*

$$\sum_f P[\vec{c}|f, m, A_1] = \sum_f P[\vec{c}|f, m, A_2].$$

As originally stated, the NFL theorem also assumes that A is deterministic—which search methods such as GAs and SA are not. However, W&M show that the definition of A can be extended so that the NFL theorem also encompasses stochastic algorithms.

More generally, attention can be shifted to some *performance measure* $\Phi(\vec{c})$. As a corollary of the NFL theorem, it is clear that the average of $P[\Phi(\vec{c})|f, m, A]$ over all functions f is independent of A, which provides the justification for such interpretations of the theorem as those quoted above.

A.2 Nonrevisiting Search

In the previous section, it was emphasized that the sequence of points generated was distinct, i.e., the search does not revisit previous points. W&M extend their result to allow for a search algorithm A that does revisit (as real algorithms do) by defining a new algorithm A' that "skips over" such points. The new algorithm can be described as a "compacted" version of the original. The NFL theorem then clearly applies to the compacted version A', and can also be applied to A as long as the value m is reinterpreted as the number of distinct points seen. The histogram $\vec{c}$ must also be interpreted as relating only to the distinct points and not to possible repeats.

W&M then argue that the

> real-world cost in using an algorithm is usually set by the number of *distinct* evaluations of f. [emphasis added]

If this is so, then their statements concerning real-world algorithms, such as that quoted above with respect to GAs and SA, would be justified.

However, in the real world, search algorithms do tend to revisit previously seen points, and such revisiting is not costless. Either we have to accept the fact, and incur the expense of evaluating f for a point $\mathbf{x}$ that we have already seen (and evaluated) before, or we have to implement some mechanism for detecting that a revisit has occurred in order to save ourselves an unnecessary evaluation. The latter approach would clearly become more and more computationally demanding as the number of points visited increases.

The problem of revisiting is particularly acute in some implementations of a GA. As the GA population converges, parents become more similar and tend to produce offspring that are simply "clones." For this reason Booker[13] proposed a "reduced surrogate" crossover mechanism that would identify regions of the parent strings where crossover would generate a clone. Similarly, Davis[27] suggested imposing a "no-duplicate" policy, whereby each offspring generated in a steady-state GA is compared to every member of the current population and only admitted if it has no duplicate. Neither of these approaches guarantees that the next point is distinct from *all* those seen before, but they are relatively

inexpensive ways of reducing the chance of revisiting, and both are claimed to enhance the performance of a GA.

The question of revisiting is, of course, a prominent factor in the motivation for TS. As usually implemented, TS does not guarantee that points will never be revisited, but it does offer a straightforward and effective way of discouraging revisits.

That this is important seems intuitively sensible, but it also follows as a consequence of the NFL theorem. It is a straightforward argument from the NFL theorem, that if we have a decision between two algorithms, we should choose that algorithm which consistently generates a lower fraction of duplicates. Of course, whether such an algorithm (one that *consistently* generated fewer duplicates for all f) exists is an open question. There are also other considerations to be taken into account—most importantly, what mechanism is used to discourage revisiting, and how much computational effort is required by it.

A.3 Implications

This is not intended to be a comprehensive survey of the issues raised in [149]. Nevertheless, there are several fairly obvious implications of the NFL theorem that have some relevance to operations research.

- In heuristic search, it is probable that there are "horses for courses." Some problem classes can be expected to be more easily solved by one technique than by another. As an example of this, Reeves[119] showed that a particular class of "deceptive" functions that are hard for a GA can be optimized easily by a TS implementation.
- A popular way of comparing different search methods is to devise some test problem instances (often generated by some random process), and then to observe and analyze their performance on the test problems. In the light of the NFL theorem, this process is clearly in need of careful thought—if the test instances are biased in some way, we may end up tuning a heuristic to a particular set of benchmarks, only to find subsequently that it performs much less well in the more general case.
- It would seem reasonable that revisiting should be discouraged in general, so methods that allow this to be implemented simply and effectively would seem to be favored. However, there are other considerations that the NFL theorem ignores—for example, the question of the computational complexity of an algorithm.

References

1. C.C. AGGARWAL, J.B. ORLIN, and R.P. TAI, 1997. Optimized Crossover for the Independent Set Problem, *Operations Research 45*, 226–234.
2. J. ALANDER, 1996. *An Indexed Bibliography of Genetic Algorithms.* Available by `ftp` in directory `cs` from `ftp.uwasa.fi`.
3. R.F. ALBRECHT, C.R. REEVES, and N.C. STEELE (eds.), 1993. *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, Springer-Verlag, Vienna.
4. K.S. AL-SULTAN, M.F. HUSSAIN, and J.S. NIZAMI, 1996. A Genetic Algorithm for the Set Covering Problem, *Journal of the Operational Research Society 47*, 702–709.
5. J. ANTONISSE, 1989. A New Interpretation of Schema Notation that Overturns the Binary Encoding Constraint, in *Proceedings of 3rd International Conference on Genetic Algorithms*, J.D. Schaffer (ed.), Morgan Kaufmann, Los Altos, CA, 86–91.
6. J.E. BAKER, 1987. Reducing Bias and Inefficiency in the Selection Algorithm, in *Proceedings of the 2nd International Conference on Genetic Algorithms*, J.J. Grefenstette (ed.), Lawrence Erlbaum Associates, Hillsdale, NJ, 14–21.
7. J.E. BEASLEY, 1990. OR-Library: Distributing Test Problems by Electronic Mail, *Journal of the Operational Research Society 41*, 1069–1072.
8. J. BEASLEY, 1993. Lagrangean Relaxation, in *Modern Heuristic Techniques for Combinatorial Problems*, C.R. Reeves (ed.), Blackwell Scientific Publications, Oxford, UK, 243–303.
9. J.E. BEASLEY and P.C. CHU, 1996. A Genetic Algorithm for the Set Covering Problem, *European Journal of Operational Research 94*, 392–404.
10. R.K. BELEW and L.B. BOOKER, (eds.), 1991. *Proceedings of 4th International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA.
11. J.N. BHUYAN, V.V. RAGHAVAN, and V.K. ELAYAVALLI, 1991. Genetic Algorithm for Clustering with an Ordered Representation, in *Proceedings of 4th International Conference on Genetic Algorithms*, R.K. Belew and L.B. Booker (eds.), Morgan Kaufmann, San Mateo, CA, 408–415.
12. J.L. BLANTON, JR. and R.L. WAINWRIGHT, 1993. Multiple Vehicle Routing with Time and Capacity Constraints using Genetic Algorithms, in *Proceedings of 5th International Conference on Genetic Algorithms*, S. Forrest (ed.), Morgan Kaufmann, San Mateo, CA, 452–459.
13. L.B. BOOKER, 1987. Improving Search in Genetic Algorithms, in *Genetic Algorithms and Simulated Annealing*, L. Davis (ed.), Morgan Kaufmann, Los Altos, CA, 61–73.
14. D.E. BROWN, C.L. HUNTLEY, and A.R. SPILLANE, 1989. A Parallel Genetic Heuristic for the Quadratic Assignment Problem, in *Proceedings of 3rd International Conference on Genetic Algorithms*, J.D. Schaffer (ed.), Morgan Kaufmann, Los Altos, CA, 406–415.
15. C. CALDWELL and V.S. JOHNSTON, 1991. Tracking a Criminal Suspect through "Face-Space" with a Genetic Algorithm, in *Proceedings of 4th International Conference on Genetic Algorithms*, R.K. Belew and L.B. Booker (eds.), Morgan Kaufmann, San Mateo, CA, 416–421.
16. H.M. CARTWRIGHT and G.F. MOTT, 1991. Looking Around: Using Clues from the Data Space to Guide Genetic Search, in *Proceedings of 4th International Conference on Genetic Algorithms*, R.K. Belew and L.B. Booker (eds.), Morgan Kaufmann, San Mateo, CA, 108–114.
17. R.A. CARUANA and J.D. SCHAFFER, 1988. Representation and Hidden Bias: Gray vs. Binary Coding for Genetic Algorithms, in *Proceedings of the 5th International Conference on Machine Learning*, Morgan Kaufmann, Los Altos, CA.
18. L. CHAMBERS (ed.), 1995. *Practical Handbook of Genetic Algorithms: Applications, Volume I*, CRC Press, Boca Raton, FL.
19. L. CHAMBERS (ed.), 1995. *Practical Handbook of Genetic Algorithms: New Frontiers, Volume II*, CRC Press, Boca Raton, FL.
20. P.C. CHU and J.E. BEASLEY, 1994. *A Genetic Algorithm for the Set Partitioning Problem*, Technical Report, The Management School, Imperial College, London.
21. P.C. CHU and J.E. BEASLEY, 1995. *A Genetic Algorithm for the Generalised Assignment Problem*, Technical Report, The Management School, Imperial College, London.
22. G.A. CLEVELAND and S.F. SMITH, 1989. Using Genetic Algorithms to Schedule Flow Shop Releases, in *Proceedings of 3rd*

International Conference on Genetic Algorithms, J.D. Schaffer (ed.), Morgan Kaufmann, Los Altos, CA, 160–169.
23. D.G. CONWAY and M.A. VENKATARAMANAN, 1994. Genetic Search and the Dynamic Facility Layout Problem, *Computers & Operations Research 21*, 955–960.
24. J.C. CULBERSON, 1995. Mutation-Crossover Isomorphisms and the Construction of Discriminating Functions, *Evolutionary Computation 2*, 279–311.
25. Y. DAVIDOR, H-P. SCHWEFEL, and R. MÄNNER (eds.), 1994. *Parallel Problem-Solving from Nature 3*, Springer-Verlag, Berlin.
26. L. DAVIS, 1985. Job Shop Scheduling with Genetic Algorithms, in *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, J.J. Grefenstette (ed.), Lawrence Erlbaum Associates, Hillsdale, NJ, 136–140.
27. L. DAVIS (ed.), 1991. *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York.
28. K.A. DE JONG, 1975. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Doctoral dissertation, University of Michigan, Ann Arbor, MI.
29. K.A. DE JONG and W.M. SPEARS, 1992. A Formal Analysis of the Role of Multi-point Crossover in Genetic Algorithms, *Annals of Mathematics and Artificial Intelligence 5*, 1–26.
30. K.A. DE JONG, 1993. Genetic Algorithms are NOT Function Optimizers, in *Foundations of Genetic Algorithms 2*, D. Whitley (ed.), Morgan Kaufmann, San Mateo, CA, 5–18.
31. K.A. DE JONG, W.M. SPEARS, and D.F. GORDON, 1995. Using Markov Chains to Analyze GAFOs, in *Foundations of Genetic Algorithms 3*, D. Whitley and M. Vose (eds.), Morgan Kaufmann, San Mateo, CA, 115–137.
32. F. DELLA CROCE, R. TADEI, and G. VOLTA, 1995. A Genetic Algorithm for the Job Shop Problem, *Computers & Operations Research 22*, 15–24.
33. U. DORNDORF and E. PESCH, 1995. Evolution Based Learning in a Job Shop Scheduling Environment, *Computers & Operations Research 22*, 25–40.
34. K.A. DOWSLAND, 1996. Genetic Algorithms—A Tool for OR? *Journal of the Operational Research Society 47*, 550–561.
35. H. ESBENSEN, 1995. Computing Near-Optimal Solutions to the Steiner Tree Problem in a Graph Using a Genetic Algorithm, *Networks 26*, 173–185.
36. L.J. ESHELMAN (ed.), 1995. *Proceedings of 6th International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA.
37. A. FAIRLEY, 1991. *Comparison of Methods of Choosing the Crossover Point in the Genetic Crossover Operation*, Technical Report, Department of Computer Science, University of Liverpool, Liverpool, UK.
38. E. FALKENAUER and A. DELCHAMBRE, 1992. A Genetic Algorithm for Bin Packing and Line Balancing, in *Proceedings of the IEEE International Conference on Robotics and Automation*.
39. H.L. FANG, P. ROSS, and D. CORNE, 1993. A Promising Genetic Algorithm Approach to Job-Shop Scheduling, Re-scheduling, and Open-Shop Scheduling Problems, in *Proceedings of 5th International Conference on Genetic Algorithms*, S. Forrest (ed.), Morgan Kaufmann, San Mateo, CA, 375–382.
40. T.C. FOGARTY, 1989. Varying the Probability of Mutation in the Genetic Algorithm, in *Proceedings of 3rd International Conference on Genetic Algorithms*, J.D. Schaffer (ed.), Morgan Kaufmann, Los Altos, CA, 104–109.
41. T.C. FOGARTY (ed.), 1994. *Evolutionary Computing: AISB Workshop, Leeds, UK, April 1994; Selected Papers*, Springer-Verlag, Berlin.
42. C.M. FONSECA and P.J. FLEMING, 1993. Genetic Algorithms for Multi-Objective Optimization: Formulation, Discussion and Generalization, in *Proceedings of 5th International Conference on Genetic Algorithms*, S. Forrest (ed.), Morgan Kaufmann, San Mateo, CA, 416–423.
43. S. FORREST (ed.), 1993. *Proceedings of 5th International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA.
44. B.R. FOX and M.B. MCMAHON, 1991. Genetic Operators for Sequencing Problems, in *Foundations of Genetic Algorithms*, G.J.E. Rawlins (ed.), Morgan Kaufmann, San Mateo, CA, 284–300.
45. M.L. FREDMAN, D.S. JOHNSON, L.A. MCGEOCH, and G. OSTHEIMER, 1995. Data Structures for Traveling Salesmen, *Journal of Algorithms 18*, 432–479.
46. B. FREISLEBEN and P. MERZ, 1996. New Genetic Local Search Operators for the Traveling Salesman Problem, in *Parallel Problem-Solving from Nature 4*, H-M. Voigt, W. Ebeling, I. Rechenberg, and H-P. Schwefel (eds.), Springer-Verlag, Berlin, 890–899.
47. M. GENDREAU, A. HERTZ, and G. LAPORTE, 1992. New Insertion and Post-Optimization Procedures for the Traveling Salesman Problem, *Operations Research 40*, 1086–1094.
48. F.A.W. GEORGE, 1996. Hybrid Genetic Algorithms with Immunisation to Optimise Networks of Retail Outlets, *Studies in Location Analysis 8*, 59–74.
49. F. GLOVER and M. LAGUNA, 1993. Tabu Search, in *Modern Heuristic Techniques for Combinatorial Problems*, C.R. Reeves (ed.), Blackwell Scientific Publications, Oxford, UK, 70–150.
50. D.E. GOLDBERG, 1985. *Optimal Initial Population Size for Binary-coded Genetic Algorithms*, TCGA Report 85001, University of Alabama, Tuscaloosa, AL.
51. D.E. GOLDBERG and R. LINGLE, 1985. Alleles, Loci and the Traveling Salesman Problem, in *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, J.J. Grefenstette (ed.), Lawrence Erlbaum Associates, Hillsdale, NJ, 154–159.
52. D.E. GOLDBERG and R.E. SMITH, 1987. Nonstationary Function Optimization using Genetic Algorithms with Dominance and Diploidy, in *Proceedings of the 2nd International Conference on Genetic Algorithms*, J.J. Grefenstette (ed.), Lawrence Erlbaum Associates, Hillsdale, NJ, 59–68.
53. D.E. GOLDBERG, 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA.
54. D.E. GOLDBERG, 1989. Sizing Populations for Serial and Parallel Genetic Algorithms, in *Proceedings of 3rd International Conference on Genetic Algorithms*, J.D. Schaffer (ed.), Morgan Kaufmann, Los Altos, CA, 70–79.
55. D.E. GOLDBERG, 1990. A Note on Boltzmann Tournament Selection for Genetic Algorithms and Population-Oriented Simulated Annealing, *Complex Systems 4*, 445–460.
56. D.E. GOLDBERG and M. RUDNICK, 1991. Genetic Algorithms and the Variance of Fitness, *Complex Systems 5*, 265–278.
57. D.E. GOLDBERG and K. DEB, 1991. A Comparative Analysis of Selection Schemes Used in Genetic Algorithms, in *Foundations of Genetic Algorithms*, G.J.E. Rawlins (ed.), Morgan Kaufmann, San Mateo, CA, 69–93.
58. V.S. GORDON, A.P.W. BOHM, and D. WHITLEY, 1993. *A Note on the Performance of Genetic Algorithms on Zero–One Knapsack Problems*, Technical Report CS-93-108, Colorado State University, Fort Collins, CO.
59. M. GORGES-SCHLEUTER, 1989. ASPARAGOS: An Asynchronous Parallel Genetic Optimization Strategy, in *Proceedings of 3rd International Conference on Genetic Algorithms*, J.D. Schaffer (ed.), Morgan Kaufmann, Los Altos, CA, 422–427.
60. J.J. GREFENSTETTE (ed.), 1985. *Proceedings of an International*

Conference on Genetic Algorithms and Their Applications, Lawrence Erlbaum Associates, Hillsdale, NJ.

61. J.J. GREFENSTETTE, 1986. Optimization of Control Parameters for Genetic Algorithms, *IEEE Transactions on Systems, Man and Cybernetics SMC-16*, 122–128.
62. J.J. GREFENSTETTE (ed.), 1987. *Proceedings of the 2nd International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, NJ.
63. O. HAAS, K.J. BURNHAM, J. MILLS, C.R. REEVES, and M.H. FISHER, 1996. Hybrid Genetic Algorithms Applied to Beam Orientation in Radiotherapy, in *Proceedings of the 4th European Congress on Intelligent Techniques and Soft Computing*.
64. P.J.B. HANCOCK, 1994. An Empirical Comparison of Selection Methods in Evolutionary Algorithms, in *Evolutionary Computing: AISB Workshop, Leeds, UK, April 1994; Selected Papers*, T.C. Fogarty (ed.), Springer-Verlag, Berlin, 80–94.
65. I. HARVEY, 1992. *Evolutionary Robotics and SAGA: The Case for Hill Crawling and Tournament Selection*, Technical Report, School of Cognitive and Computing Sciences, University of Sussex, Brighton, UK.
66. R. HAUSER and R. MÄNNER, 1994. Implementation of Standard Genetic Algorithm on MIMD Machines, in *Parallel Problem-Solving from Nature 3*, Y. Davidor, H-P. Schwefel and R. Männer (eds.), Springer-Verlag, Berlin, 504–513.
67. J. HESSER, R. MÄNNER, and O. STUCKY, 1989. Optimization of Steiner Trees Using Genetic Algorithms, in *Proceedings of 3rd International Conference on Genetic Algorithms*, J.D. Schaffer (ed.), Morgan Kaufmann, Los Altos, CA, 231–236.
68. J. HESSER and R. MÄNNER, 1991. Towards an Optimal Mutation Probability for Genetic Algorithms, in *Parallel Problem-Solving from Nature*, H-P. Schwefel and R. Männer (eds.), Springer-Verlag, Berlin, 23–32.
69. F. HOFFMEISTER and T. BÄCK, 1991. Genetic Algorithms and Evolution Strategies: Similarities and Differences, in *Parallel Problem-Solving from Nature*, H-P. Schwefel and R. Männer (eds.), Springer-Verlag, Berlin, 455–471.
70. C. HÖHN and C.R. REEVES, 1995. *Embedding Local Search Operators in a Genetic Algorithm*, Technical Report, School of Mathematical and Information Sciences, Coventry University, Coventry, UK.
71. C. HÖHN and C.R. REEVES, 1996. Graph Partitioning using Genetic Algorithms, in *Proceedings of the 2nd Conference on Massively Parallel Computing Systems*, G.R. Sechi (ed.), IEEE Computer Society, Los Alamitos, CA, 31–38.
72. J.H. HOLLAND, 1975. *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI; re-issued by MIT Press, 1992.
73. A. HOMAIFAR, S. GUAN, and G.E. LIEPINS, 1993. A New Approach on the Traveling Salesman Problem by Genetic Algorithms, in *Proceedings of 5th International Conference on Genetic Algorithms*, S. Forrest (ed.), Morgan Kaufmann, San Mateo, CA, 460–466.
74. C.R. HOUCK, J.A. JOINES, and M.G. KAY, 1996. Comparison of Genetic Algorithms, Random Restart and Two-Opt Switching for Solving Large Location-Allocation Problems, *Computers & Operations Research 23*, 587–596.
75. C.L. HUNTLEY and D.E. BROWN, 1991. Parallel Heuristics for the Quadratic Assignment Problem, *Computers & Operations Research 18*, 275–289.
76. P. JOG, J.Y. SUH, and D. VAN GUCHT, 1991. The Effects of Population Size, Heuristic Crossover and Local Improvement on a Genetic Algorithm for the Traveling Salesman Problem, in *Proceedings of 3rd International Conference on Genetic Algorithms*, J.D. Schaffer (ed.), Morgan Kaufmann, Los Altos, CA, 110–115.
77. D.S. JOHNSON, 1990. Local Optimization and the Traveling Salesman Problem, in *Automata, Languages and Programming, Lecture Notes in Computer Science 443*, G. Goos and J. Hartmanis (eds.), Springer-Verlag, Berlin, 446–461.
78. D.R. JONES and M.A. BELTRAMO, 1991. Solving Partitioning Problems with Genetic Algorithms, in *Proceedings of 4th International Conference on Genetic Algorithms*, R.K. Belew and L.B. Booker (eds.), Morgan Kaufmann, San Mateo, CA, 442–449.
79. T.C. JONES, 1995. *Evolutionary Algorithms, Fitness Landscapes and Search*, Doctoral dissertation, University of New Mexico, Albuquerque, NM.
80. B.A. JULSTROM, 1993. A Genetic Algorithm for the Rectilinear Steiner Tree Problem, in *Proceedings of 5th International Conference on Genetic Algorithms*, S. Forrest (ed.), Morgan Kaufmann, San Mateo, CA, 474–480.
81. A. KAPSALIS, G.D. SMITH, and V.J. RAYWARD-SMITH, 1993. Solving the Graphical Steiner Tree Problem using Genetic Algorithms, *Journal of the Operational Research Society 44*, 397–406.
82. A.J. KEANE, 1993. Structural Design for Enhanced Noise Performance Using Genetic Algorithm and Other Optimisation Techniques, in *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, R.F. Albrecht, C.R. Reeves, and N.C. Steele (eds.), Springer-Verlag, Vienna, 536–543.
83. S. KOBAYASHI, I. ONO, and M. YAMAMURA, 1995. An Efficient Genetic Algorithm for Job Shop Scheduling Problems, in *Proceedings of 6th International Conference on Genetic Algorithms*, L.J. Eshelman (ed.), Morgan Kaufmann, San Mateo, CA, 506–511.
84. G. VON LASZEWSKI, 1991. Intelligent Structural Operators for the *k*-way Graph Partitioning Problem, in *Proceedings of 4th International Conference on Genetic Algorithms*, R.K. Belew and L.B. Booker (eds.), Morgan Kaufmann, San Mateo, CA, 45–52.
85. G. VON LASZEWSKI and H. MÜHLENBEIN, 1991. Partitioning a Graph with a Parallel Genetic Algorithm, in *Parallel Problem-Solving from Nature*, H-P. Schwefel and R. Männer (eds.), Springer-Verlag, Berlin, 165–169.
86. C-H. LEE, C-I. PARK, and M. KIM, 1989. Efficient Algorithm for Graph Partitioning Problem using a Problem Transformation Method, *Computer Aided Design 21*, 611–618.
87. D. LEVINE, 1996. Application of a Hybrid Genetic Algorithm to Airline Crew Scheduling, *Computers & Operations Research 23*, 547–558.
88. G.E. LIEPINS and W.D. POTTER, 1991. A Genetic Algorithm Approach to Multiple-Fault Diagnosis, in *Handbook of Genetic Algorithms*, L. Davis (ed.), Van Nostrand Reinhold, New York, 237–250.
89. S. MACLANE and G. BIRKHOFF, 1979. *Algebra*, 2nd ed., Macmillan, New York.
90. R. MÄNNER and B. MANDERICK (eds.), 1992. *Parallel Problem-Solving from Nature 2*, Elsevier Science Publishers, Amsterdam.
91. S. MARTELLO and P. TOTH, 1991. *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley & Sons, New York.
92. O. MARTIN, S.W. OTTO, and E.W. FELTEN, 1992. Large Step Markov Chains for the TSP Incorporating Local Search Heuristics, *Operations Research Letters 11*, 219–224.
93. K. MATHIAS and D. WHITLEY, 1992. Genetic Operators, the Fitness Landscape and the Traveling Salesman Problem, in *Parallel Problem-Solving from Nature 2*, R. Männer and B. Manderick (eds.), Elsevier Science Publishers, Amsterdam, 219–228.
94. S. MCKEE and M.B. REED, 1987. An Algorithm for the Alignment of Gas Turbine Components in Aircraft, *IMA Journal of Mathematics in Management 1*, 133–144.
95. Z. MICHALEWICZ and C.Z. JANIKOW, 1991. Handling Con-

straints in Genetic Algorithms, in *Proceedings of 4th International Conference on Genetic Algorithms*, R.K. Belew and L.B. Booker (eds.), Morgan Kaufmann, San Mateo, CA, 151–157.

96. H. MÜHLENBEIN, 1991. Evolution in Time and Space—the Parallel Genetic Algorithm, in *Foundations of Genetic Algorithms*, G.J.E. Rawlins (ed.), Morgan Kaufmann, San Mateo, CA, 316–337.
97. I.M. OLIVER, D.J. SMITH, and J.R.C. HOLLAND, 1987. A Study of Permutation Crossover Operators on the Traveling Salesman Problem, in *Proceedings of the 2nd International Conference on Genetic Algorithms*, J.J. Grefenstette (ed.), Lawrence Erlbaum Associates, Hillsdale, NJ, 224–230.
98. D. ORVOSH and L. DAVIS, 1993. Shall We Repair? Genetic Algorithms, Combinatorial Optimization and Feasibility Constraints, in *Proceedings of 5th International Conference on Genetic Algorithms*, S. Forrest (ed.), Morgan Kaufmann, San Mateo, CA, 650.
99. C.C. PALMER and A. KERSHENBAUM, 1995. An Approach to a Problem in Network Design Using Genetic Algorithms, *Networks 26*, 151–163.
100. D.W. PEARSON, N.C. STEELE, and R.F. ALBRECHT (eds.), 1995. *Proceedings of the 2nd International Conference on Artificial Neural Networks and Genetic Algorithms*, Springer-Verlag, Vienna.
101. H. PIRKUL and E. ROLLAND, 1994. New Heuristic Solution Procedures for the Uniform Graph Partitioning Problem: Extensions and Evaluation, *Computers & Operations Research 21*, 895–907.
102. P.W. POON and J.N. CARTER, 1995. Genetic Algorithm Crossover Operators for Ordering Applications, *Computers & Operations Research 22*, 135–147.
103. P. PRINETTO, M. REBAUDENGO, and M. SONZA REORDA, 1993. Hybrid Genetic Algorithms for the Traveling Salesman Problem, in *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, R.F. Albrecht, C.R. Reeves and N.C. Steele (eds.), Springer-Verlag, Vienna, 559–566.
104. P. PROSSER, 1988. A Hybrid Genetic Algorithm for Pallet Loading, in *Proceedings of 8th European Conference on Artificial Intelligence*, Pitman, London.
105. N.J. RADCLIFFE, 1991. Forma Analysis and Random Respectful Recombination, in *Proceedings of 4th International Conference on Genetic Algorithms*, R.K. Belew and L.B. Booker (eds.), Morgan Kaufmann, San Mateo, CA, 222–229.
106. N.J. RADCLIFFE, 1991. Equivalence Class Analysis of Genetic Algorithms, *Complex Systems 5*, 183–205.
107. N.J. RADCLIFFE, 1991. Non-linear Genetic Representations, in *Parallel Problem-Solving from Nature 2*, R. Männer and B. Manderick (eds.), Elsevier Science Publishers, Amsterdam, 259–268.
108. N.J. RADCLIFFE and F.A.W. GEORGE, 1993. A Study in Set Recombination, in *Proceedings of 5th International Conference on Genetic Algorithms*, S. Forrest (ed.), Morgan Kaufmann, San Mateo, CA, 23–30.
109. N.J. RADCLIFFE and P.D. SURRY, 1994. Formal Memetic Algorithms, in *Evolutionary Computing: AISB Workshop, Leeds, UK, April 1994; Selected Papers*, T.C. Fogarty (ed.), Springer-Verlag, Berlin, 1–16.
110. N.J. RADCLIFFE and P. SURRY, 1995. Formae and the Variance of Fitness, in *Foundations of Genetic Algorithms 3*, D. Whitley and M. Vose (eds.), Morgan Kaufmann, San Mateo, CA, 51–72.
111. G.J.E. RAWLINS (ed.), 1991. *Foundations of Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA.
112. V.J. RAYWARD-SMITH, I.H. OSMAN, C.R. REEVES, and G.D. SMITH (eds.), 1996. *Modern Heuristic Search Methods*, John Wiley & Sons, New York.
113. I. RECHENBERG, 1973. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipen der biologischen Evolution*, Frommann-Holzboog Verlag, Stuttgart. (2nd ed., 1993).
114. C.R. REEVES, 1992. A Genetic Algorithm Approach to Stochastic Flowshop Sequencing, in *Proceedings of the IEE Colloquium on Genetic Algorithms for Control and Systems Engineering*, Digest No. 1992/106. IEE, London.
115. C.R. REEVES, 1993. Improving the Efficiency of Tabu Search in Machine Sequencing Problems, *Journal of the Operational Research Society 44*, 375–382.
116. C.R. REEVES (ed.), 1993. *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications, Oxford, UK; re-issued by McGraw-Hill, London, 1995.
117. C.R. REEVES, 1993. Using Genetic Algorithms with Small Populations, in *Proceedings of 5th International Conference on Genetic Algorithms*, S. Forrest (ed.), Morgan Kaufmann, San Mateo, CA, 92–99.
118. C.R. REEVES, 1994. Genetic Algorithms and Neighbourhood Search, in *Evolutionary Computing: AISB Workshop, Leeds, UK, April 1994; Selected Papers*, T.C. Fogarty (ed.), Springer-Verlag, Berlin, 115–130.
119. C.R. REEVES, 1994. Tabu Search Finds Global Optima of a Class of GA-hard Problem, presented at 15th International Symposium on Mathematical Programming, Ann Arbor, MI, August 1994.
120. C.R. REEVES, 1995. A Genetic Algorithm for Flowstop Sequencing, *Computers & Operations Research 22*, 5–13.
121. C.R. REEVES, 1996. Heuristic Search Methods: A Review, in *Operational Research: Keynote Papers 1996*, D. Johnson and F. O'Brien (eds.), Operational Research Society, Birmingham, UK, 122–149.
122. C.R. REEVES and C.C. WRIGHT, 1995. An Experimental Design Perspective on Genetic Algorithms, in *Foundations of Genetic Algorithms 3*, D. Whitley and M.D. Vose (eds.), Morgan Kaufmann, San Mateo, CA, 7–22.
123. C.R. REEVES, 1996. Hybrid Genetic Algorithms for Bin-packing and Related Problems, *Annals of Operations Research 63*, 371–396.
124. J.T. RICHARDSON, M.R. PALMER, G.E. LIEPINS, and M.R. HILLYARD, 1989. Some Guidelines for Genetic Algorithms with Penalty Functions, in *Proceedings of 3rd International Conference on Genetic Algorithms*, J.D. Schaffer (ed.), Morgan Kaufmann, Los Altos, CA, 191–197.
125. J.D. SCHAFFER (ed.), 1989. *Proceedings of 3rd International Conference on Genetic Algorithms*, Morgan Kaufmann, Los Altos, CA.
126. J.D. SCHAFFER, R.A. CARUANA, L.J. ESHELMAN, and R. DAS, 1989. A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization, in *Proceedings of 3rd International Conference on Genetic Algorithms*, J.D. Schaffer (ed.), Morgan Kaufmann, Los Altos, CA, 51–60.
127. J.D. SCHAFFER and L.J. ESHELMAN, 1996. Combinatorial Optimization by Genetic Algorithms: The Value of the Genotype/Phenotype Distinction, in *Modern Heuristic Search Methods*, V.J. Rayward-Smith, I.H. Osman, C.R. Reeves, and G.D. Smith (eds.), John Wiley & Sons, New York, 85–97.
128. J.D. SCHAFFER, D. WHITLEY, and L.J. ESHELMAN, 1992. Combinations of Genetic Algorithms and Neural Networks: A Survey of the State of the Art, in *Proceedings of COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, L.D. Whitley and J.D. Schaffer (eds.), IEEE Computer Society Press, Los Alamitos, CA, 1–37.

129. N.N. SCHRAUDOLPH and R.K. BELEW, 1992. Dynamic Parameter Encoding for Genetic Algorithms *Machine Learning 9*, 9–21.
130. H-P. SCHWEFEL, 1977. *Numerische Optimierung von Computermodellen mittels der Evolutionsstrategie*, Birkhäuser Verlag, Basel. (English edition: *Numerical Optimization of Computer Models*, John Wiley & Sons, Chichester, UK, 1981.)
131. H-P. SCHWEFEL and R. MÄNNER (eds.), 1991. *Parallel Problem-Solving from Nature*, Springer-Verlag, Berlin.
132. C.G. SHAEFER, 1987. The ARGOT strategy: Adaptive Representation Genetic Optimizer Technique, in *Proceedings of the 2nd International Conference on Genetic Algorithms*, J.J. Grefenstette (ed.), Lawrence Erlbaum Associates, Hillsdale, NJ, 50–58.
133. D. SMITH, 1985. Bin Packing with Adaptive Search, in *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, J.J. Grefenstette (ed.), Lawrence Erlbaum Associates, Hillsdale, NJ, 202–207.
134. A.E. SMITH and D.M. TATE, 1993. Genetic Optimization using a Penalty Function, in *Proceedings of 5th International Conference on Genetic Algorithms*, S. Forrest (ed.), Morgan Kaufmann, San Mateo, CA, 499–505.
135. T. STARKWEATHER, S. MCDANIEL, K. MATHIAS, D. WHITLEY, and C. WHITLEY, 1991. A Comparison of Genetic Sequencing Operators, in *Proceedings of 4th International Conference on Genetic Algorithms*, R.K. Belew and L.B. Booker (eds.), Morgan Kaufmann, San Mateo, CA, 69–76.
136. G. SYSWERDA and J. PALMUCCI, 1991. The Application of Genetic Algorithms to Resource Scheduling, in *Proceedings of 4th International Conference on Genetic Algorithms*, R.K. Belew and L.B. Booker (eds.), Morgan Kaufmann, San Mateo, CA, 502–508.
137. K.Y. TAM, 1992. Genetic Algorithms, Function Optimization and Layout Design, *European Journal of Operational Research 63*, 322–346.
138. D.M. TATE and A.E. SMITH, 1995. A Genetic Approach to the Quadratic Assignment Problem, *Computers & Operations Research 22*, 73–83.
139. S. THANGIAH, R. VINAYAGAMOORTHY, and A.V. GUBBI, 1993. Vehicle Routing and Time Deadlines Using Genetic and Local Algorithms, in *Proceedings of 5th International Conference on Genetic Algorithms*, S. Forrest (ed.), Morgan Kaufmann, San Mateo, CA, 506–513.
140. N.L.J. ULDER, E.H.L. AARTS, H-J. BANDELT, P.J.M. LAARHOVEN, and E. PESCH, 1991. Genetic Local Search Algorithms for the Traveling Salesman Problem, in *Parallel Problem-Solving from Nature*, H-P. Schwefel and R. Männer (eds.), Springer-Verlag, Berlin, 109–116.
141. R.J.M. VAESSENS, E.H.L. AARTS, and J.K. LENSTRA, 1992. A Local Search Template, in *Parallel Problem-Solving from Nature 2*, R. Männer and B. Manderick (eds.), Elsevier Science Publishers, Amsterdam, 65–74.
142. H-M. VOIGT, W. EBELING, I. RECHENBERG, and H-P. SCHWEFEL (eds.), 1996. *Parallel Problem-Solving from Nature 4*, Springer-Verlag, Berlin.
143. M.D. VOSE, 1993. Modeling Simple Genetic Algorithms, in *Foundations of Genetic Algorithms 2*, D. Whitley (ed.), Morgan Kaufmann, San Mateo, CA, 63–74.
144. D. WHITLEY, 1989. The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best, in *Proceedings of 3rd International Conference on Genetic Algorithms*, J.D. Schaffer (ed.), Morgan Kaufmann, Los Altos, CA, 116–121.
145. D. WHITLEY, K. MATHIAS, and P. FITZHORN, 1991. Delta Coding: An Iterative Search Strategy, in *Proceedings of 4th International Conference on Genetic Algorithms*, R.K. Belew and L.B. Booker (eds.), Morgan Kaufmann, San Mateo, CA, 77–84.
146. D. WHITLEY, T. STARKWEATHER, and D. SHANER, 1991. The Traveling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination, in *Handbook of Genetic Algorithms*, L. Davis (ed.), Van Nostrand Reinhold, New York, 350–372.
147. D. WHITLEY, 1993. An Executable Model of a Simple Genetic Algorithm, in *Foundations of Genetic Algorithms 2*, D. Whitley (ed.), Morgan Kaufmann, San Mateo, CA, 45–62.
148. D. WHITLEY (ed.), 1993. *Foundations of Genetic Algorithms 2*, Morgan Kaufmann, San Mateo, CA.
149. D.H. WOLPERT and W.G. MACREADY, 1995. *No Free Lunch Theorems for Search*, Technical Report SFI-TR-95-02-010, Santa Fe Institute, Santa Fe, NM.
150. A. WREN and D.O. WREN, 1995. A Genetic Algorithm for Public Transport Scheduling, *Computers & Operations Research 22*, 101–110.
151. T. YAMADA and R. NAKANO, 1996. Job-Shop Scheduling, in *Genetic Algorithms in Engineering Systems*, P.J. Fleming and A. Zalzala (eds.), Peter Peregrinus, London.
152. G. ZWEIG, 1995. An Effective Tour Construction and Improvement Procedure for the Traveling Salesman Problem, *Operations Research 43*, 1049–1057.