Journal of Heuristics, 11: 323–357 (1998)
© 1998 Kluwer Academic Publishers

Constraint Handling in Genetic Algorithms: The Set Partitioning Problem

P.C. CHU AND J.E. BEASLEY
The Management School, Imperial College, London SW7 2AZ, England
email: p.chu@ic.ac.uk, http://mscmga.ms.ic.ac.uk/pchu/pchu.html
j.beasley@ic.ac.uk, http://mscmga.ms.ic.ac.uk/jeb/jeb.html

Abstract

In this paper we present a genetic algorithm-based heuristic for solving the set partitioning problem (SPP). The SPP is an important combinatorial optimisation problem used by many airlines as a mathematical model for flight crew scheduling.

A key feature of the SPP is that it is a highly constrained problem, all constraints being equalities. New genetic algorithm (GA) components: separate fitness and unfitness scores, adaptive mutation, matching selection and ranking replacement, are introduced to enable a GA to effectively handle such constraints. These components are generalisable to any GA for constrained problems.

We present a steady-state GA in conjunction with a specialised heuristic improvement operator for solving the SPP. The performance of our algorithm is evaluated on a large set of real-world problems. Computational results show that the genetic algorithm-based heuristic is capable of producing high-quality solutions.

Key Words: combinatorial optimisation, crew scheduling, genetic algorithms, set partitioning

1. Introduction

1.1. *The set partitioning problem*

The set partitioning problem (SPP) is the problem of exactly covering the rows of a m-row, n-column, zero-one matrix (a_{ij}) by a subset of the columns at minimal cost. Defining $x_j = 1$ if column j (with cost c_j) is in the solution and $x_j = 0$ otherwise, the SPP is

$$\text{minimise} \quad \sum_{j=1}^{n} c_j x_j, \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j = 1, \quad i = 1, \ldots, m, \tag{2}$$

$$x_j \in \{0, 1\}, \qquad j = 1, \ldots, n, \tag{3}$$

To ease the notation define $I = \{1, \ldots, m\}$ the set of rows, $J = \{1, \ldots, n\}$ the set of columns, $\alpha_i = \{j \mid a_{ij} = 1,\ j \in J\}$ the set of columns that cover row i, and $\beta_j = \{i \mid a_{ij} = 1, i \in I\}$ the set of rows covered by column j.

The set partitioning problem has been studied extensively over the years because of its many important applications (Balas and Padberg (1979)). The best-known application of the SPP is airline crew scheduling (Arabeyre et al. (1969), Baker and Fisher (1981), Gershkoff (1989), Hoffman and Padberg (1993), Marsten and Shepardson (1981)). In this context, each row ($i = 1, \ldots, m$) represents a flight leg (a takeoff and landing) that must be flown. The columns ($j = 1, \ldots, n$) represent feasible round-trip rotations for a crew (i.e., a sequence of flight legs for a crew that begin and end at individual base locations and that conform to all applicable work rules). Associated with each rotation is a cost c_j. The matrix $(a_{ij})_{m\times n}$ is constructed as $a_{ij} = 1$ if flight leg i is covered by rotation j, $a_{ij} = 0$ otherwise. The objective of the crew scheduling problem is to find the "best" collection of rotations such that each flight leg is covered by *exactly* one rotation.

For some practical crew scheduling problems, due to flight assignments, union rules and other factors, some additional constraints are imposed. These constraints are called *base constraints* and have the form $d^l \leq \sum_{j\in B} d_j x_j \leq d^u$, where $d_j > 0$ and $B \subseteq J$. Also in some applications overcovering (rewriting Eq. (2) as $\geq$) is allowed. This form of the problem is called the set covering problem. In this paper, we will only consider the pure set partitioning problem as defined by Eqs. (1)–(3).

2. Related work

Because of the importance of the SPP, a number of algorithms have been developed. These can be classified into two categories: exact algorithms which attempt to solve the SPP to optimality, and heuristic algorithms which try to find "good" solutions quickly.

The starting point for most exact solution algorithms is to solve the linear programming (LP) relaxation of the SPP (i.e., replace (3) by $0 \leq x_j \leq 1$, $j = 1, \ldots, n$). A number of authors (Gershkoff (1989), Marsten and Shepardson (1981)) have noted that for many "small" SPP problems, the solution to the LP relaxation is either all integer, in which case it is also the optimal integer solution, or has only a few fractional values which can be easily resolved. However, as the size of the problem increases, fractional values occur more frequently and simple round-up methods are likely to fail. A method based on cutting planes was proposed by Balas and Padberg (1976). They noted that cutting plane algorithms were moderately successful even while using general purpose cuts and without taking advantage of any special knowledge of the SPP polytope (see also Balas and Padberg (1979)).

Another type of exact method is the use of the tree search (branch-and-bound). Various bounding strategies, including LP and Lagrangean relaxation, have been exploited. The algorithm described in Marsten (1974) is an LP-based algorithm that exploits the structure of the problem within a branch-and-bound tree. Computational results were reported for several large problems using real data. An improved version of Marsten's algorithm was given by Marsten and Shepardson (1981). As pointed out in both papers, solving the highly degenerate LP relaxation of the SPP was the computational bottleneck at that time. However, recent computational advances in LP algorithms have overcome such difficulties, and the LP relaxation of very large SPP instances can now be solved relatively quickly.

Another tree search method was proposed by Fisher and Kedia (1990), who used continuous adaptations of the greedy and three-opt methods applied to the dual of the LP relaxation of the problem to provide lower bounds. Chan and Yano (1992) presented a tree search algorithm based on a lower bound derived from a multiplier adjustment heuristic for the dual of the LP relaxation of the problem. Harche and Thompson (1994) presented an exact algorithm based on a new method, called the column subtraction (or row sum) method, which is capable of solving large sparse instances of set covering, packing and partitioning problems. The most successful optimal solution algorithm to date appears to be the work of Hoffman and Padberg (1993). They presented an exact algorithm based on branch and cut (which involves solving the LP relaxation of the problem and incorporating cuts derived from polyhedral considerations) and reported optimal solutions for a large set of real-world set partitioning problems, with sizes up to 145 rows and 1,053,137 columns. Other recent published work relating to the SPP appears in (Eben-Chaime, Tovey, and Ammons (1996), Sherali and Lee (1996), Tasi (1995)).

There have been relatively few heuristic solution algorithms for the SPP in the literature. This is because the SPP is a highly constrained problem and thus finding any feasible solution to a SPP is itself a difficult problem. Ryan and Falkner (1988) provided a method of obtaining a good feasible solution by imposing additional structure, derived from the real-world problems but not already implicit in the mathematical model. They observed that this additional structure not only reduces the size of the feasible region, but also improves the integer properties of the LP solution. Atamtürk, Nemhauser, and Savelsberg (1995) presented a heuristic algorithm incorporating problem reduction, linear programming, cutting planes and a Lagrangean dual procedure based upon the work of Wedelin (1995a,b).

Recently, Levine (1994, 1996) experimented with a parallel genetic algorithm and applied it to the SPP. His genetic algorithm (GA) is based on an island model where a GA is used with multiple independent subpopulations (each run on a different processor) and highly fit individuals occasionally migrate between the subpopulations. His computational study was conducted on a subset of the problems used by Hoffman and Padberg (1993). Although his algorithm was capable of finding optimal solutions for some problems having up to a few thousand columns, it had difficulty finding feasible solutions for problems having many rows.

In this paper we attempt to apply a GA to the SPP based on the following motivations. Firstly, we are particularly interested in the design of a GA for highly constrained problems. The SPP, with all constraints being equalities, is a challenging example of such a problem. Secondly, the parallel GA proposed by Levine has not produced completely satisfactory results in comparison with other more successful methods. One of the limitations of his GA was its inability to find feasible solutions for some problems. We recognise that this was most likely due to the use of a penalty function in his GA. We believe that, with a suitable approach, a GA can be an effective heuristic for solving the SPP. Lastly, since heuristic algorithms for the SPP are almost non-existent, we hope to develop GAs as an alternative approach to exact methods for the SPP.

The paper is organised as follows. In Section 3 the basic steps of a simple GA are described. In Section 4 the GA-based heuristic for the SPP is presented. In Section 5 computational results are given. Finally in Section 6 some conclusions are drawn.

3. Genetic algorithms

A genetic algorithm can be understood as an "intelligent" probabilistic search algorithm which can be applied to a variety of combinatorial optimisation problems (Reeves (1993)). The theoretical foundations of GAs were originally developed by Holland (1975). GAs are based on the evolutionary process of biological organisms in nature. During the course of evolution, natural populations evolve according to the principles of natural selection and "survival of the fittest". Individuals which are more successful in adapting to their environment will have a better chance of surviving and reproducing, whilst individuals which are less fit will be eliminated. This means that the *genes* from the highly fit individuals will spread to an increasing number of individuals in each successive generation. The combination of good characteristics from highly adapted ancestors may produce even more fit offspring. In this way, species evolve to become more and more well adapted to their environment.

A GA simulates these processes by taking an initial population of individuals and applying genetic operators in each reproduction. In optimisation terms, each individual in the population is encoded into a string or *chromosome* which represents a possible *solution* to a given problem. The fitness of an individual is evaluated with respect to a given objective function. Highly fit individuals or *solutions* are given opportunities to reproduce by exchanging pieces of their genetic information, in a *crossover* procedure, with other highly fit individuals. This produces new "offspring" solutions (i.e., *children*), which share some characteristics taken from both parents. Mutation is often applied after crossover by altering some genes in the strings. The offspring can either replace the whole population (*generational* approach) or replace less fit individuals (*steady-state* approach). This evaluation-selection-reproduction cycle is repeated until a satisfactory solution is found. The basic steps of a simple GA are shown below.

```
Generate an initial population;
Evaluate fitness of individuals in the population;
repeat
    Select parents from the population;
    Recombine (mate) parents to produce children;
    Mutate children;
    Evaluate fitness of the children;
    Replace some or all of the population by the children;
until a satisfactory solution has been found;
```

A more comprehensive overview of GAs can be found in (Bäck, Fogel, and Michalewicz (1997), Beasley, Bull, and Martin (1993a,b), Goldberg (1989), Mitchell (1996), Reeves (1993)).

4. A GA for the SPP

One of the conclusions we have drawn from our previous work on GAs (Beasley and Chu (1996), Chu (1997), Chu and Beasley (1997, 1998)) is that, in order to successfully apply

GAs to constrained combinatorial optimisation problems (such as the SPP), a GA that uses only the traditional operators (crossover and mutation) would appear to be ineffective because of the limited power of these general-purpose operators to generate feasible solutions. However, we can apply the philosophy of GAs to such problems by choosing a natural representation of solutions, by defining properties based on this representation, and by constructing problem-specific operators which manipulate the solution structures so as to improve the quality and/or feasibility of the solutions. Although the disadvantage of this type of approach is that the GA becomes domain-dependent, we believe that this is the best way, if not the only way, to construct GAs which can be competitive with existing algorithms that utilise domain-specific knowledge. These modifications to the basic GA framework in our algorithm for the SPP are described below. In the following discussion we will use the terms "individuals", "solutions" and "strings" interchangeably.

4.1. Representation and fitness function

The first step in designing a genetic algorithm for a particular problem is to devise a suitable representation scheme. The usual 0-1 binary representation is an obvious choice for the SPP since it represents the underlying 0-1 integer variables. Hence the SPP can be coded as a string of length n over the binary alphabet $\{0, 1\}$, representing the underlying 0-1 integer variables. In this representation, a bit in the string is associated with each column. The jth bit, $S[j]$, is 1 if column j is in the solution and 0 otherwise (see figure 1).

Note here that Levine (1994, 1996) also used the same representation in his GA for the SPP. This binary representation does not ensure the feasibility of individuals when the traditional crossover and mutation operators are applied. Note here that in the SPP, a constraint i is infeasible if and only if row i is not being covered by exactly one column (i.e., $w_i \neq 1$ where $w_i = \sum_{j \in J} a_{ij} S[j]$).

The main design issue of the GA then becomes how should the constraints be embraced, and if infeasible solutions are allowed, how should they be evaluated? There are three standard ways (Davis and Steenstrup (1987), Michalewicz (1995)) of dealing with constraints and infeasible solutions in GAs:

1. to use a representation that automatically ensures that all solutions are feasible,
2. to design a heuristic operator (often called a *repair operator*) which guarantees to transform any infeasible solution into a feasible solution,
3. to apply a penalty function (Goldberg (1989), Levine (1994, 1996), Powell and Skolnick (1993), Richardson et al. (1989), Smith and Tate (1993)) to penalise the fitness of any infeasible solution without distorting the fitness landscape.

column j	1	2	3	4	5	$\cdots$	$n-1$	n
$S[j]$	0	1	0	1	1	$\cdots$	1	0

Figure 1. Binary representation of a SPP solution.

The first of these approaches does not appear to be applicable to the SPP. As to the second of these approaches, in our previous work (Beasley and Chu (1996), Chu and Beasley (1998)) we have considered problems for which effective repair operators that ensure satisfaction of all the constraints can be easily developed, and consequently this simplified the evaluation of solutions. Unfortunately, in the case of the SPP, an efficient (polynomial-time) algorithm which guarantees to transform infeasible solutions into feasible solutions is not known to exist. Hence this approach too does not appear to be applicable to the SPP.

The third of these approaches allows infeasible individuals to be generated but penalises them by adding a penalty term to the fitness function such that the fitness $f(S)$ of a solution S is given by $f(S) = h(S) + p(S)$, where $h(S)$ and $p(S)$ are the objective and penalty terms for S, respectively.

Levine (1994) investigated three penalty terms for the SPP:

- The *countinfz penalty term*: $p(S) = \sum_{i \in I, w_i \neq 1} \lambda_i$, where the scalar weight λ_i is chosen (empirically) to be $\lambda_i = \max_{j \in \alpha_i} \{c_j\}$. This penalty term penalises each violated constraint by a fixed amount, regardless of how much the constraint is violated.
- The *linear penalty term*: $p(S) = \sum_{i \in I, w_i \neq 1} \lambda_i |w_i - 1|$. This penalty term takes into account the magnitude of each constraint violation.
- The *Smith-Tate* (Smith and Tate (1993)) *penalty term*: $p(S) = \sum_{i \in I, w_i \neq 1} (z_f - z_b)/2$ where z_f is the best *feasible* objective function value found so far and z_b is the best (feasible or infeasible) objective function value found so far. The aim of this penalty term is to favour solutions which are near a feasible solution over highly fit solutions which are far from any feasible solution. The distance from feasibility is measured by the number of constraints violated.

Experiments conducted by Levine showed mixed results for these three penalty terms. In fact, the penalty method performed poorly, managing to produce optimal solutions in only a small proportion of trials for small-sized problems, and failing to produce feasible solutions for some large-sized problems. Levine (1994) concluded that much work remained to be done in designing better penalty functions. Michalewicz (1995) too concluded that it is difficult to design penalty functions. He also highlighted the lack of constraint handling techniques in GAs.

Given a problem, such as the SPP, for which a repair operator and appropriate penalty function are difficult to construct, we propose, in this paper, a new alternative approach for processing infeasible solutions. The weakness of the penalty methods is that the optimal penalty coefficient values λ_i may be highly data-dependent. Hence it is unlikely that a set of parameter values will be useful for all problems. Moreover, it is not clear whether, with the penalty approach, the selection/replacement mechanisms should favour unfit but feasible individuals or highly fit but infeasible individuals. Therefore, instead of combining the objective and penalty terms into a single fitness measure, we redefine fitness in a way which involves separating the single fitness measure into two: one measure is still called *fitness* but the other measure is called *unfitness*. Each individual can then be represented by a pair of values that measure the objective function value (fitness) and the degree of infeasibility (unfitness) independently, obviating the need for the scaling factors λ_i.

The fitness $f(S)$ of an individual S for the SPP is chosen to be equal to its objective function value, calculated by $f(S) = \sum_{j\in J} c_j S[j]$. Note that the *lower* the fitness value the *more* fit the solution is.

The unfitness $u(S)$ of an individual S measures the degree of infeasibility (in relative terms) and for the SPP we have chosen to define it as $u(S) = \sum_{i\in I w_i \neq 1} |w_i - 1|$. Recall that w_i measures the number of times row i is being covered. The absolute value here implies that an individual is feasible if and only if $u(S) = 0$ and infeasible if $u(S) > 0$. Defining the unfitness to be at a minimum of zero if and only if the solution is feasible seems a natural approach.

Note that we would stress here that there is no one unique way of defining an unfitness score for the SPP (or for any other problem) and alternative expressions are equally valid, e.g., $u(S) = \sum_{i\in I w_i \neq 1} |w_i - 1|^2$ could be used to measure unfitness.

There are several advantages to using separate fitness and unfitness scores instead of a single penalty-adjusted fitness score. Firstly, it eliminates the difficult task of determining the appropriate λ_i values. Secondly, it transforms the "fitness landscape" from a one-dimensional line (fitness only) into a two-dimensional plane (fitness and unfitness axes), thus allowing the point which a solution represents to be identified more precisely. Finally, based on the fitness and unfitness scores, we can divide the population into several subgroups of distinct characteristics which can be useful in deciding to which search region the GA should be directed during the selection and replacement processes. The details of the design of this scheme are discussed in Section 4.5.

We should stress here that although the idea of separate fitness (objective function) and unfitness (constraint violation) values are being presented in the context of the SPP they are directly applicable to *any* GA for constrained problems.

4.2. Crossover and mutation operators

The crossover operator takes bits from each parent string and "combines" them to create a child string. The idea is that by creating new strings from substrings of fit parent strings, new and promising areas of the search space will be explored. Many crossover techniques exist in the literature. However, GA researchers (Spears and DeJong (1991), Syswerda (1989)) have given evidence to support the claim that uniform crossover has a better recombination potential than do other crossover operators, such as the classical one-point and two-point crossover operators. The uniform crossover operator works as follows. Let P_1 and P_2 be the parent strings $P_1[1], \ldots, P_1[n]$ and $P_2[1], \ldots, P_2[n]$, respectively. Then the child solution C is created by setting $C[j] = P_1[j]$ with probability 0.5, $C[j] = P_2[j]$ otherwise, i.e., for each bit j a random choice is made as to which parent will contribute its bit ($P_1[j]$ or $P_2[j]$) to the child.

Mutation is usually applied to each child after crossover. It works by *inverting M* randomly chosen bits in a string where M is experimentally determined. Mutation is generally seen as an operator which provides a small amount of random search. It also helps to guard against loss of valuable genetic information by reintroducing information lost due to premature convergence and thereby expands the search space.

Two types of mutation procedure are used in our GA. The first type is called *static mutation*, which has a constant mutation rate M_s throughout the GA search. The second type is called *adaptive mutation*, which mutates only certain bits at each iteration based on some heuristic rules. We propose the use of adaptive mutation for the following reasons.

As mentioned earlier, the SPP is a highly constrained problem and a feasible solution is often difficult to construct heuristically. Our initial experimental results, as well as the results reported by Levine, indicated that for some difficult problems, the GA may fail to evolve even a single feasible solution after a large number of iterations. This problem arises because an attempt to satisfy a particular constraint may introduce infeasibilities into other currently feasible constraints. As the population begins to converge, the GA will gradually evolve a "dominating" set of columns which favour certain constraints at the expense of others. We call this outcome *premature column convergence*. The negative effect of this outcome is that some constraints (rows) may never be satisfied, and the likelihood that a feasible solution will emerge is severely diminished. The problem of premature column convergence may also arise naturally due to the fact that some constraints are easier to satisfy than others.

An adaptive mutation procedure is designed to counter these natural biases, hence preventing premature column convergence. The idea is to prevent the loss of certain columns that may result in permanent violation of some constraints. By monitoring population statistics with respect to constraint violation/satisfaction, we could bring back these columns through biased mutation. The details of this procedure are as follows:

1. At each GA iteration and for each constraint (row) i, record the number of individuals (denoted by η_i) in which constraint i is violated. Here $\eta_i = \sum_{k=1}^{N} \min[1, |\sum_{j \in J} a_{ij} S_k[j] - 1|]$, where N is the size of the population and the second term in this minimisation is zero if constraint i is satisfied in individual S_k.
2. For each constraint i, if $\eta_i \geq \epsilon N (0 < \epsilon < 1)$, then set $M_a (1 \leq M_a \leq |\alpha_i|)$ randomly chosen bits (columns) from α_i to one in the child string.

In step 1, information on the extent of constraint violation is collected. In step 2, it is conjectured that if the number of individuals which violate constraint i exceeds some threshold, i.e., $\eta_i \geq \epsilon N$, it is likely that the population is moving toward premature column convergence because individuals which satisfy constraint i are "dying out". In order to re-introduce solutions which satisfy constraint i into the population, columns which cover row i are added into the child string to "compete" with other columns. This is done by setting M_a bits (of those that cover row i) in the child string to one, where M_a is arbitrarily chosen.

Limited computational experience showed that a low static mutation rate ($2 < M_s < 5$) was preferable to a higher mutation rate ($M_s > 5$) and that within the low range the quality of the solution was not particularly sensitive to the rate. We set $M_s = 3$ in our GA. We also set the adaptive mutation parameters ϵ and M_a to be 0.5 and 5, respectively. These mean that when more than 50% of the population violate constraint i, adaptive mutation will take place in the child string with a mutation rate $M_a = 5$. These values have experimentally been shown to be quite satisfactory.

We should stress here that although the idea of adaptive mutation is being presented in the context of the SPP the idea is generalisable to *any* GA for constrained problems. As a general approach we simply (as in step 1 above) collect information on the extent of constraint violations and then (as in step 2 above) introduce appropriate variables into the solution in an attempt to encourage constraint satisfaction.

4.3. Heuristic improvement operator

The child solutions generated by the crossover and mutation operators are likely to be infeasible, i.e., some rows may be under-covered and some rows may be over-covered. This led us to develop an additional improvement operator to be incorporated into the GA. The aim of this operator is to improve the solution by moving to a near feasible, or possibly feasible, solution.

To accomplish this goal, we designed a heuristic improvement operator that includes two basic procedures: one is called DROP and the other is called ADD. The role of the DROP procedure is to identify all *over-covered* rows and randomly remove columns until all rows are covered by *at most* one column. The role of the ADD procedure is to identify all *under-covered* rows and add columns such that as many under-covered rows as possible can be covered without causing any other rows to be over-covered. The heuristic improvement operator is presented in Algorithm 1.

Algorithm 1 Heuristic improvement operator for the SPP

```
Let: { $S =$ the set of columns in a solution,
     { $U =$ the set of uncovered rows,
     { $w_i =$ the number of columns that cover row $i$, $i \in I$ in $S$.

 1: initialise $w_i := |\alpha_i \cap S|, \forall i \in I$;
 2: set $T := S$; /* $T$ is a dummy set */
 3: repeat /* DROP procedure */
 4:     randomly select a column $j$, $j \in T$ and set $T \leftarrow T - j$;
 5:     if $w_i \geq 2$, for any $i \in \beta_j$ then
 6:         set $S \leftarrow S - j$; set $w_i \leftarrow w_i - 1, \forall i \in \beta_j$;
 7:     end if
 8: until $T = \emptyset$;
 9: initialise $U := \{i \mid w_i = 0, \forall i \in I\}$;
10: set $V := U$; /* $V$ is a dummy set */
11: repeat /* ADD procedure */
12:     randomly select a row $i \in V$ and set $V \leftarrow V - i$;
13:     search for the column $j \in \alpha_i$ that satisfies $\beta_j \subseteq U$, and minimises $c_j/|\beta_j|$;
14:     if $j$ exists then
15:     set $S \leftarrow S + j$; set $w_i \leftarrow w_i + 1, \forall i \in \beta_j$; set $U \leftarrow U - \beta_j$ and $V \leftarrow V - \beta_j$;
16:     end if
17: until $V = \emptyset$.
```

In Algorithm 1, steps 3–8 (DROP) attempt to make over-covered rows (i.e., those i such that $w_i \geq 2$) feasible by setting all but one of the columns $j \in \alpha_i \cap S$ to zero. Notice here that columns are inspected for exclusion in a random fashion rather than in any fixed order (such as by considering higher-cost columns first). The reasoning is to minimise the possibility of biasing certain columns that may lead to premature column convergence. In steps 11–17 (ADD), columns are added back to the solution such that as many under-covered rows as possible are being covered without introducing over-coverage into other rows. This procedure examines each under-covered row randomly so that biases toward any particular rows are minimised. But the inclusion of the columns is done in a greedy manner by considering columns in decreasing cost-ratio $c_j/|\beta_j|$. Note that this procedure allows only *under-covered* rows, but not over-covered rows, to exist in the solution. Although slightly over-covered solutions may also be useful, we decided not to further increase the complexity of the procedure by additionally searching for over-covered solutions that may potentially give lower unfitness values.

Levine (1994, 1996) in his GA for the SPP applied a different heuristic improvement operator to the one we have presented above to a single randomly selected member of the population at each iteration. By contrast we apply our operator to each child solution.

4.4. Parent selection method

Parent selection is the task of assigning reproductive opportunities to each individual in the population based on their relative fitnesses. Since each individual possesses both fitness and unfitness scores, the parent selection criterion may be based on either the fitness, the unfitness, or a combination of both. The difficulties with the first two criteria are that, at the initial stages of the GA, individuals which have lower fitness scores (more fit) tend to have higher unfitness scores (more infeasible) and vice-versa. Thus, if we favour selection on more fit individuals, we are likely to select parents which are mostly infeasible. This may not help the GA in generating feasible offspring. On the other hand, if we favour selection on feasible, but less fit individuals, we may find that whilst feasibility may improve the solution quality will suffer. In order to balance this tradeoff, a combination of the fitness and unfitness values (i.e., $f(S_i) + \lambda u(S_i)$) may be a better selection basis. But again, the problem with this approach is that an appropriate scalar λ, similar to those required by the penalty method, may be difficult to determine.

The shortcomings of these selection methods motivated us to develop an alternative selection method, called the *matching selection* method. The main objective of this method is to select parents such that, combining them, would result in an improvement in feasibility without undermining solution quality. This method incorporates problem-specific knowledge which takes into consideration the row coverage of each candidate parent. In a matching selection, the first parent P_1 is selected using a binary tournament (Goldberg (1989)) based on *fitness*. The second parent P_2 is then selected so as to give a maximum *compatibility* score. A compatibility score between parent P_1 and any candidate second parent S_i ($S_i \neq P_1$) from the population is defined by $\delta(P_1, S_i) = |R_{P_1} \cup R_{S_i}| - |R_{P_1} \cap R_{S_i}|$, where R_{P_1} and R_{S_i} are the set of rows covered by P_1 and S_i, respectively. If the maximum compatibility score is scored by more than one member, then the tie-breaking rule is to select the one with the lowest fitness value.

The logic here is that we would like the two parents together to cover as many rows as possible (i.e., a high $|R_{P_1} \cup R_{S_i}|$) and with as few covered rows in common as possible (i.e., a low $|R_{P_1} \cap R_{S_i}|$).

One exception to the matching rule is when the first parent P_1 is a feasible solution $(u(P_1) = 0)$, then the second parent P_2 is selected using the tournament selection method based on fitness instead of the matching method. This is because computational results indicated that a feasible child can be constructed relatively easily if one or both parents are feasible. So when the population largely consists of feasible individuals, the parent selection strategy is then re-directed toward improving fitness rather than feasibility.

One issue here is that using matching selection we try and identify two parents who together cover the rows "well", as defined by the compatibility score. However the crossover used (uniform crossover see Section 4.2) will, via the columns it chooses, most likely destroy this coverage. The heuristic improvement operator though, which is applied to each child after crossover, is designed to achieve good row coverage, especially if started from a child solution which already contains a set of columns that contributed to good row coverage. This issue is also addressed in Section 5.3.2 below.

We should stress here that although the idea of matching selection is being presented in the context of the SPP the idea is generalisable to *any* GA for constrained problems. As a general approach we simply (as above) select the first parent in a standard way but select the second parent based upon a compatibility score defined with regard to the constraints in the problem under investigation.

4.5. Population replacement scheme

4.5.1. Ranking replacement. In order to insert the newly generated child solution into the population, room must be made for the child, which means we must have a method for selecting a population member to be deleted. The ideal performance of any replacement strategy is that members having both fitness and unfitness values *smaller* than the population average will, on average, be represented in the new population to a greater extent. Likewise, members with above-average fitness and unfitness should decrease in the population. However, this ideal performance may not be achieved by any simple replacement mechanism since a trade-off between fitness and unfitness values generally occurs. To see this consider two simple, but less than ideal, replacement strategies:

1. Replace the member with the worst *fitness*.
2. Replace the member with the worst *unfitness*.

Clearly, the first replacement strategy based on fitness has the effect of favouring the quality (fitness score) over feasibility (unfitness score) of the individuals. This may not be an ideal strategy since individuals which have lower fitness values tend to have higher unfitness values initially. Thus, eliminating unfit (but maybe feasible) individuals early in the run may force the population to converge toward infeasibility prematurely. On the other hand, the second strategy based on unfitness favours feasible individuals over fit individuals. Whilst feasibility of the population may improve, hence increasing the chance of finding

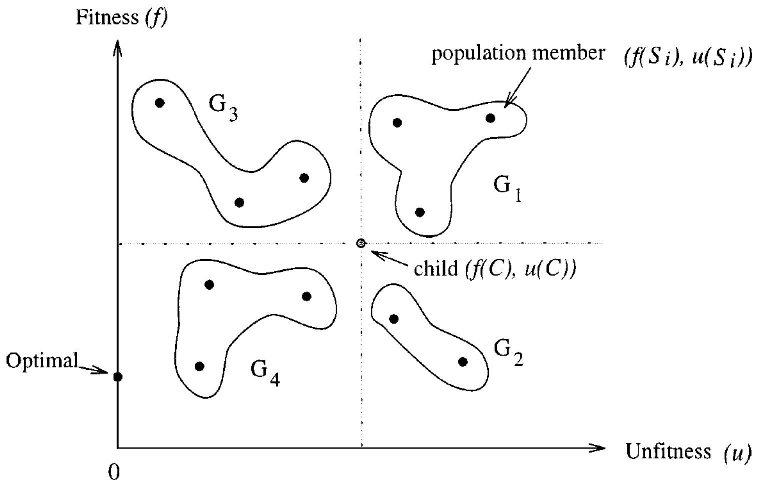

Figure 2. Population subgroups and the fitness-unfitness landscape.

feasible solutions, it may also reduce the probability that the population will converge to high-quality (or optimal) solutions because the average fitness may increase rather than decrease over time.

In order to balance these trade-offs between fitness and unfitness, we designed a new generalised replacement scheme, called the *ranking replacement* method, that takes into account both the fitness and unfitness scores when selecting a population member for deletion. This method is described as follows.

To set up a ranking replacement, the population is first divided into four mutually exclusive subgroups *with respect to the child,* see figure 2. Figure 2 shows the fitness-unfitness landscape of the population where the x and y axes represent unfitness and fitness, respectively. Each point in the plot represents an individual (solution) whose coordinate is provided by the individual's fitness and unfitness scores (note that the higher the fitness and unfitness scores for a solution, the worse the solution is). The population $\mathcal{P}$ is hence separated into four disjoint subgroups with respect to the fitness and unfitness of the child. These four subgroups are defined as follows.

$$G_1 = \{S_i \mid f(S_i) \geq f(C), u(S_i) \geq u(C), S_i \in \mathcal{P}\},$$

$$G_2 = \{S_i \mid f(S_i) < f(C), u(S_i) \geq u(C), S_i \in \mathcal{P}\},$$

$$G_3 = \{S_i \mid f(S_i) \geq f(C), u(S_i) < u(C), S_i \in \mathcal{P}\},$$

$$G_4 = \{S_i \mid f(S_i) < f(C), u(S_i) < u(C), S_i \in \mathcal{P}\}.$$

By considering the subgroups in the order: first G_1 then G_2 then G_3 then G_4, a child in the ranking replacement scheme will replace a selected member of the *first* non-empty subgroup in this order. This implies that a child solution will first try to replace a solution in subgroup G_1, which has higher fitness and unfitness scores than the child's, thus improving both the average fitness and unfitness of the whole population. If subgroup G_1 is empty, then subgroup G_2 is considered next, followed by G_3 and G_4 in that order. The reason for considering subgroup G_2 prior to G_3 is that for problems for which feasible solutions are harder to find, the priority is to first search for a feasible solution before trying to improve the fitness (or the quality) of the solution. Therefore, by eliminating solutions in subgroups G_1 and G_2 first, the average unfitness of the population will be reduced (i.e., the population will shift towards the fitness axis).

In any subgroup, the member selected for replacement by the child is the member with the *worst unfitness* (ties broken by worst fitness). Note that when replacing a solution, care must be taken to prevent a duplicate solution from entering the population. A *duplicate* child is one such that its solution structure is identical to any one of the solution structures in the population. Allowing duplicate solutions to exist in the population may be undesirable because a population could come to consist of all identical solutions, thus severely limiting the ability of the GA to generate new solutions.

The ranking replacement scheme may be effective in improving both the feasibility and the quality of the solutions. To see this, we refer back to figure 2. Figure 2 shows that in order for the GA to evolve good feasible solutions, the initial solutions (denoted by the points on the plot) should move towards the optimal solution (which lies on the fitness axis) as the GA progresses. Any new improved child solutions will eliminate other solutions in the population with high unfitness (subgroups G_1 and G_2) via the ranking replacement scheme. The overall effect is to shift the population appropriately. This can be seen in figure 3 where we predict convergence under different replacement schemes, ranking replacement, worst-fitness replacement and worst-unfitness replacement.

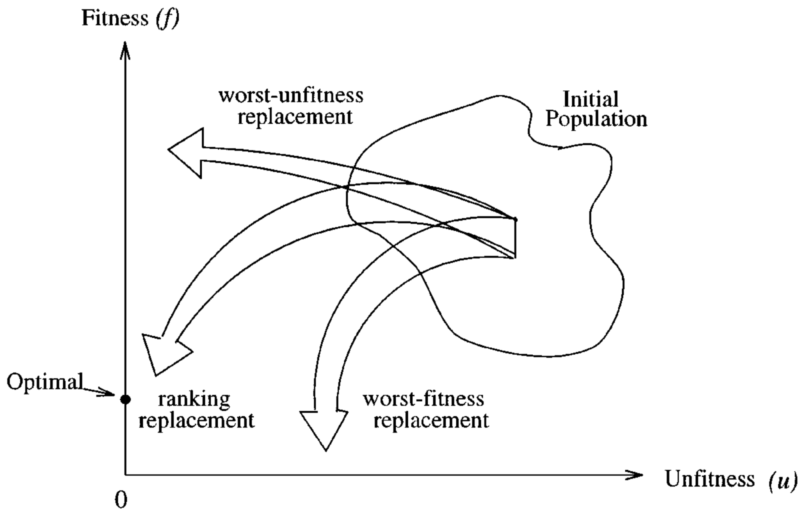

Figure 3. Predicted convergence under different replacement schemes.

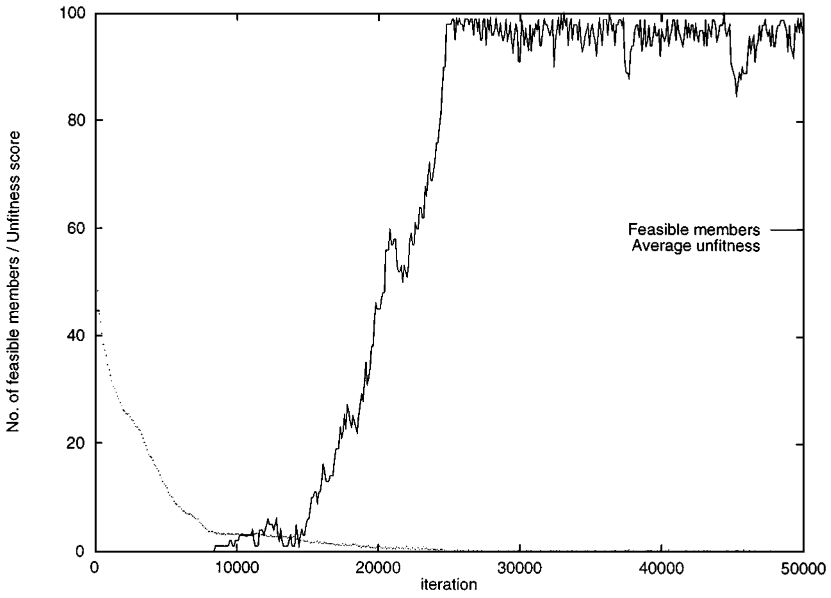

Figure 4. Ranking replacement: population characteristics.

We should stress here that although the idea of ranking replacement is being presented in the context of the SPP it is directly applicable to *any* GA for constrained problems that makes use of separate fitness and unfitness scores.

4.5.2. An example problem. It is instructive to examine whether the predicted convergence shown in figure 3 occurs in practice. Figure 5 shows the population (with a population of size 100) at different stages of the GA for test problem AA02 (see Section 5) using the ranking replacement method. The intersection between the dotted line and the fitness axis is where the optimal solution (known from other work for this test problem) lies. In the initial population (at iteration 0) the points are scattered. As the GA progresses, the points begin to converge and move towards the fitness axis. In the last plot (at iteration 50,000) the points are clustered near the optimal solution. The results are very close to ideal performance.

Figure 4 shows how the number of feasible members of the population (i.e., with unfitness zero) and the average unfitness value changes over the course of the GA shown in figure 5. It is clear that the behaviour shown is highly desirable, with the number of feasible members increasing, and the average unfitness decreasing, as the GA proceeds.

To see how the choice of different replacement strategies can affect the performance of the GA, compare figure 5 with the corresponding figures for worst-fitness replacement (figure 6) and worst-unfitness replacement (figure 7). In figure 6, replacing the member with the worst fitness score at each iteration causes the population to converge to only infeasible solutions. In figure 7, replacing the member with the worst-unfitness fares better, finding feasible solutions but converging prematurely to poor sub-optimal solutions.

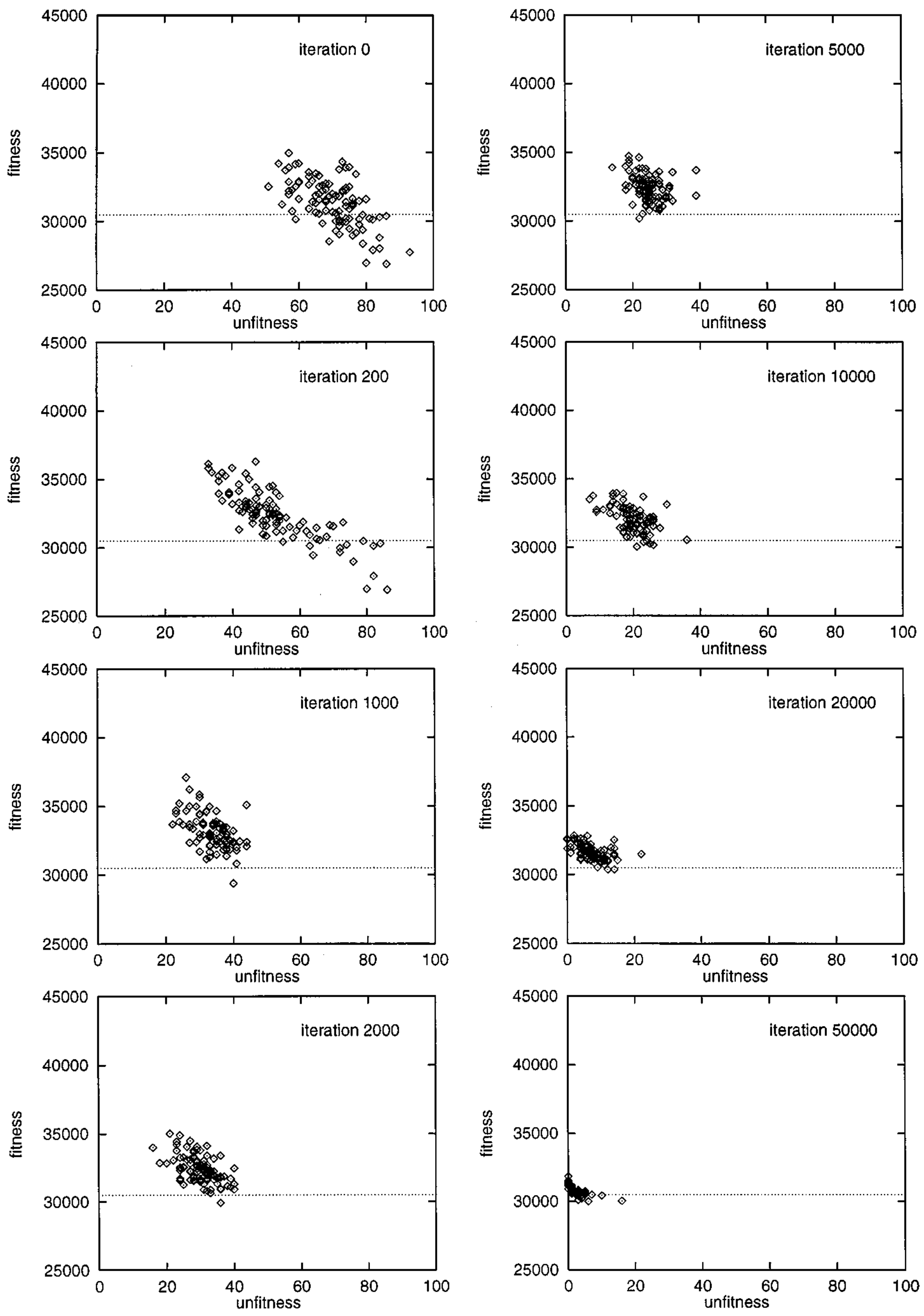

Figure 5. Convergence using the ranking replacement method.

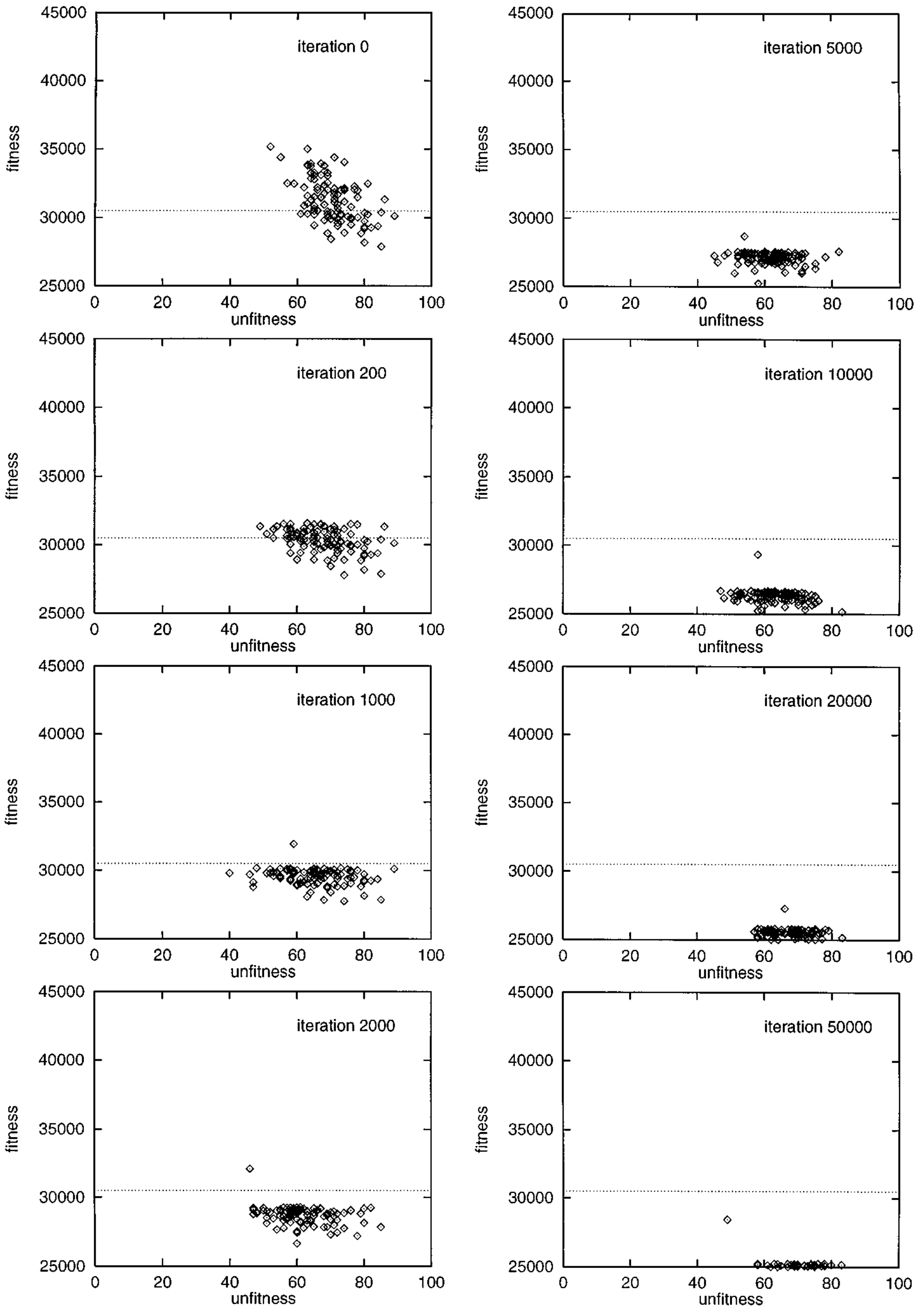

Figure 6. Convergence using the worst-fitness replacement method.

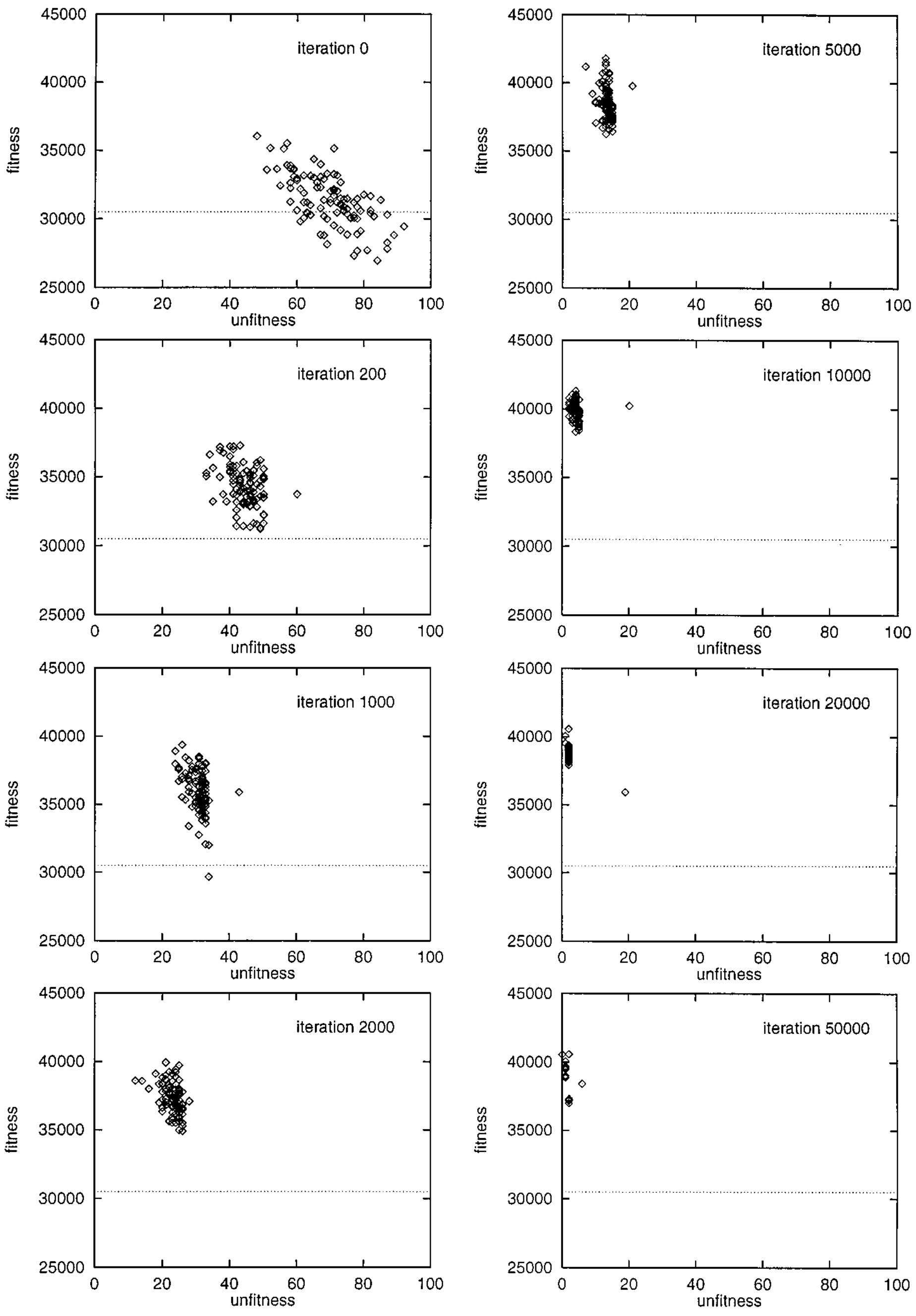

Figure 7. Convergence using the worst-unfitness replacement method.

Figures 5, 6 and 7 correspond to the predicted behaviour shown in figure 3 and demonstrate that the choice of which replacement strategy to use can dramatically affect the outcome of a GA.

4.5.3. Penalty term. One question that can be asked is whether the ranking replacement scheme is really just equivalent to using a linear penalty term $\textit{fitness} + \lambda(\textit{unfitness})$, but with the λ not predetermined but deduced from the population in some way. We present a simple counter-example to prove that this is not so.

Letting (fitness, unfitness) represent the fitness and unfitness scores of a solution suppose we have a child (10, 10) and three members in the population, namely: (20, 80), hence in G_1 see figure 2; (5, 100), hence in G_2; and (100, 5), hence in G_3. Under ranking replacement (20, 80) in G_1 will be chosen for replacement.

In order for (20, 80) to be chosen for replacement under a linear penalty term scheme there must exist a $\lambda \geq 0$ such that $20 + 80\lambda \geq 5 + 100\lambda$ and $20 + 80\lambda \geq 100 + 5\lambda$ (recall here that we are attempting to minimise both fitness and unfitness). Simple algebra reveals that λ needs to satisfy $\lambda \leq 0.75$ and $\lambda \geq 1.067$, which is clearly impossible. Hence, ranking replacement is not equivalent to using a linear penalty term scheme.

4.5.4. Related work. There have been a number of papers presented in the literature in which, as in this paper with fitness and unfitness scores, GA solutions have more than one value associated with them and these values are used in a population replacement scheme. The majority of these papers concern *unconstrained multiobjective problems*.

In such problems there are M functions, $F_1, F_2, \dots, F_M$, (each of which should be minimised) and each GA solution k then has a vector of values $\{F_1[k], F_2[k], \dots, F_M[k]\}$ associated with it. The goal is to construct *Pareto-optimal* solutions. Within a known population $\mathcal{P}$ of size N a solution k is Pareto-optimal if there exists $m(1 \leq m \leq M)$ such that $\{F_m[k] < F_m[p]\ \forall p \in \mathcal{P} - k\}$ and $\{F_j[k] \leq F_j[p] j = 1, \dots, M\, \forall p \in \mathcal{P} - k\}$ i.e., a solution k is Pareto-optimal if it is better than all other solutions in the population for one of the M functions and at least as good for the other functions.

Note the difference between an unconstrained multiobjective problem and a single objective constrained problem such as the SPP considered in this paper. In an unconstrained multiobjective problem all solutions are valid in terms of the original problem, some are just better than others. In a single objective constrained problem only solutions which satisfy the constraints (i.e., are feasible) are valid.

Schaffer (1985), based on his doctoral thesis (Schaffer (1984)), presented a vector evaluated genetic algorithm (VEGA) for unconstrained multiobjective problems. In his GA (a generational approach) an "intermediate generation" is first constructed. This intermediate generation contains M subgroups, each of equal size $G = N/M$. Each subgroup $m(1 \leq m \leq M)$ contains the solutions in the population which have the G smallest values of F_m. The usual genetic operators of crossover and mutation are applied to this intermediate generation to produce a new generation. See also Goldberg (1989) for a discussion of this work.

Horn, Nafpliotis, and Goldberg (1994) presented a niched Pareto GA for unconstrained multiobjective problems utilising some suggestions of Goldberg (1989). In their GA the parent selection mechanism is modified to reflect the mutiobjective nature of the problem and fitness sharing (Deb and Goldberg (1989), Goldberg, Deb, and Horn (1992), Goldberg and Richardson (1987), Oei, Goldberg and Chang (1991)) is used. Similar approaches have also been explored by Fonseca and Fleming (1993) and Horn and Nafpliotis (1993).

Richardson et al (1989) briefly suggested that it should be possible to apply both VEGA and Pareto-optimal approaches to constrained problems. As far as we are aware the only work done in this direction has been done by Surry, Radcliffe, and Boyd (1995) who considered a constrained single objective problem that arises in gas pipeline design. They presented a GA approach to the problem based upon a method they call COMOGA, denoting Constrained Optimisation by Multi-Objective Genetic Algorithms, which draws on a number of the ideas considered above for unconstrained multiobjective problems.

In COMOGA each individual, as in this paper, has two values: an objective function value and a value summarising constraint violations based upon Pareto-ranking the individual's constraint violations against the population using the ranking technique of Fonseca and Fleming (1993). COMOGA is a generational GA where parents are selected by the standard binary tournament scheme (Goldberg (1989)), each tournament that is held being based on the objective function value with probability p_{cost}, otherwise being based on the Pareto-rank value. Such an approach is similar (but not identical) to the VEGA approach of Schaffer (1985) applied to a problem with two objectives but with different-sized "intermediate generations" for each objective. In COMOGA the parameter p_{cost} is varied from generation to generation in order to attempt to achieve a target proportion of feasible solutions in the population.

4.6. Population size and initial population

A population size of $N = 100$ was used. Limited computational experience indicted that the quality of the solution is not very sensitive to the size of the population. The initial population was generated in a heuristic (pseudo-random) manner. The algorithm for generating an initial population is described in Algorithm 2.

Algorithm 2 Initialise the population $\mathcal{P}(0)$ for the SPP

```
 1: for $k = 1$ to $N$ do
 2:   set $S_k := \emptyset$; set $U := I$;
 3:   repeat
 4:     randomly select a row $i \in U$;
 5:     randomly select a column $j \in \alpha_i$ such that $\beta_j \cap (I - U) = \emptyset$;
 6:     if $j$ exists then
 7:       set $S_k \leftarrow S_k + j$; set $U \leftarrow U - i, \forall i \in \beta_j$;
 8:     else
 9:       set $U \leftarrow U - i$;
10:     end if
11:   until $U = \emptyset$.
12: end for
```

The heuristic used here to generate an initial solution is a generalisation of the ADD heuristic used in the heuristic improvement operator described in Algorithm 1. For each uncovered row (step 4), the idea is to find a column that can cover that row without over-covering other rows (step 5). If such a column is found, the column is added to the solution (step 7). If no such column is found, then that row is left uncovered (step 9) and the procedure is repeated (steps 3–11) for another uncovered row yet to be examined until all rows have been checked. The “random” strategy we employed here (steps 4 and 5) in picking rows and columns for consideration is adopted to avoid bias towards any particular rows or columns.

4.7. Algorithmic summary

Our GA for the SPP is summarised in Algorithm 3. Note that the GA cannot guarantee to return a feasible solution. If no feasible solution has been found when the maximum allowed iteration is reached, the solution with the lowest unfitness score is returned.

There are many significant differences between our GA for the SPP (as summarised in Algorithm 3) and the GA for the SPP presented by Levine (1994), for example in our use of separate fitness/unfitness scores and ranking replacement. Even where similarities exist between our work and his, differences remain.

For example in steps 10–12 we discard any duplicate children. Levine also found it beneficial to avoid duplicates, however in contrast to the approach adopted in this paper he mutated a duplicate until it was distinct from the population. We, in step 9, apply the heuristic improvement operator to each newly generated child at each iteration. Levine applied a (different) heuristic improvement operator to a single randomly selected member of the population at each iteration.

Algorithm 3 A GA for the SPP

```
 1: set the iteration counter $t := 0$;
 2: initialise the population $\mathcal{P}(t) := \{S_1, \dots, S_N\}$, $S_i \in \{0, 1\}^n$;
 3: evaluate $\mathcal{P}(t) : \{f(S_1), u(S_1)\}, \dots, \{f(S_N), u(S_N)\}$;
 4: find $\min_{S \in \mathcal{P}(t), u(S)=0}\{f(S)\}$ or $\min_{S \in \mathcal{P}(t), u(S)>0}\{u(S)\}$, and set $S^* \leftarrow S$;
 5: while $t < t_{max}$ do
 6:    select $\{P_1, P_2\} := \Phi(\mathcal{P}(t))$; /* $\Phi$ = matching selection method */
 7:    crossover $C := \Omega_f(P_1, P_2)$; /* $\Omega_f$ = uniform crossover operator */
 8:    mutate $C \leftarrow \Omega_m(C, M_s, M_a, \epsilon)$; /* $\Omega_m$ = static and adaptive mutation operators */
 9:    $C \leftarrow \Omega_{improve}(C)$; /* $\Omega_{improve}$ = heuristic improvement operator */
10:    if $C \equiv$ any $S \in \mathcal{P}(t)$ then /* $C$ is a duplicate of a member of the population */
11:       discard $C$ and go to 6;
12:    end if
13:    evaluate $f(C), u(C)$;
14:    find a $S' \in \mathcal{P}(t)$ using ranking replacement and replace $S' \leftarrow C$; /* steady-state
       replacement */
```

```
15:   if $(u(C) = u(S^*) = 0$ and $f(C) < f(S^*))$ or $(u(S^*) > 0$ and $u(C) < u(S^*))$ then
16:     $S^* \leftarrow C$;
17:   end if /* update best solution $S^*$ found */
18:   $t \leftarrow t + 1$;
19: end while
20: return $S^*$, $f(S^*)$ and $u(S^*)$.
```

5. Computational study

5.1. Test problems and preprocessing

The GA presented in this paper was coded in C and run on a Silicon Graphics Indigo workstation (R4000, 100 MHz). To test our GA, fifty-five real-world set-partitioning problems originating from the airline industry were solved. These problems were used by Hoffman and Padberg (1993) to test their branch-and-cut algorithm, and a subset of them were used by Levine (1994, 1996) to test his GA. The optimal solutions of these problems are known (see Hoffman and Padberg (1993)). For details of how to electronically obtain these test problems see Beasley (1990, 1996), or email the message *sppinfo to o.rlibrary@ic.ac.uk* or see the WWW address *http://mscmga.ms.ic.ac.uk/jeb/orlib/sppinfo.html.*

Because of the special structures of the SPPs associated with these airline problems, it is often possible to reduce the size of the problems by applying some logical rules. This "preprocessing" of the data will not only reduce the memory requirement burden for some problems of very large-scale, it can also make the optimisation task considerably easier (i.e., having less variables and constraints to consider).

Here we briefly list several reduction procedures (Garfinkel and Nemhauser (1972), Hoffman and Padberg (1993)). Some of the procedures, namely Reductions 1 and 2, seem obvious and trivial to apply, but we mention them because we found them to be very useful for these test problems.

Reduction 1: If $\beta_j = \beta_k$ for any pair of $(j, k) \in J$, $j \neq k$ and if $c_j \leq c_k$, then delete column k (i.e., column k is a duplicate column).

Reduction 2: If $|\alpha_i| = 1, i \in I$, then the column j in α_i must be in the optimal solution ($x_j = 1$). Delete j and all rows $k \in \beta_j$, as well as all columns in $\alpha_k, k \in \beta_j$.

Reduction 3: If $\alpha_i \subseteq \alpha_k$ for any pair of $(i, k) \in I, i \neq k$, then delete all columns $j \in (\alpha_k - \alpha_i)$ and row k. This reduction follows because any column that covers row i also covers row k.

Reduction 4: If $|\alpha_i - (\alpha_i \cap \alpha_k)| = |\alpha_k - (\alpha_i \cap \alpha_k)| = 1$ for any pair of $(i, k) \in I, i \neq k$, then let column $j = \alpha_i - (\alpha_i \cap \alpha_k)$ and column $p = \alpha_k - (\alpha_i \cap \alpha_k)$.

1. If $\beta_j \cap \beta_p = \emptyset$, then columns j and p are merged into a single column having a cost $c_j + c_p$. Delete row k.
2. If $\beta_j \cap \beta_p \neq \emptyset$, then delete columns j and p. Delete row k.

Reduction 5: (*a new procedure*): For each $j \in J$, suppose $x_j = 1$, then $x_k = 0, \forall k \in T = (\{\cup_{i\in\beta_j}\alpha_i\} - j)$. If $U = \{i \mid \alpha_i \subseteq T, i \in I - \beta_j\} \neq \emptyset$, then the problem is infeasible (since row $i \in U$ cannot be covered). Therefore, we can deduce that x_j cannot have the value one. So x_j must be zero and hence column j can be deleted from the problem since no feasible solution can contain column j.

Collectively we refer to these procedures as PREP. Reduction 5 (a new reduction procedure) is particularly helpful to the GA since it can reduce the chance that the GA may converge into infeasible solutions. These reduction procedures are applied to the original problem repeatedly until no further reduction can be achieved. The original problem sizes, the reduced problem sizes given by Hoffman and Padberg (1993) and the reduced problem sizes using PREP are given in Table 1. The first column of Table 1 gives the name of the test problems, which are categorised into four sets (with different characteristics) according to the prefix: `NW`, `AA`, `US` and `KL`. The characteristics of these problems are given by number of rows, number of columns, and density (percentage of ones in the a_{ij} matrix). Table 1 shows that by using PREP, we were able to reduce the problems to smaller sizes than those reported by Hoffman and Padberg for most of the problems. Note also here that substantial problem reductions can occur, much larger reductions then would be expected for randomly generated problems. The computational effort required to carry out these reductions was reasonable, ranging from a few seconds for small problems up to less than an hour for the largest-sized problem.

5.2. Computational results

In our computational study, 10 trials of the GA heuristic (each with a different random seed) were generated for each of the 55 test problems (after PREP reductions, see Section 5.1). Each trial terminated when $t_{\max} = 100,000$ non-duplicate children had been generated. The default parameter settings for all problems were population size $N = 100$, static mutation rate $M_s = 3$ and adaptive mutation rate $M_a = 5$. To compare the results against the traditional operational research method, we used the `CPLEX` Mixed Integer solver, Version 3.0 (with all the default settings) to solve the problems to optimality. Computational results are shown in Tables 2 and 3.

In Table 2 we give, for each problem:

- the LP optimal value using the `CPLEX` LP solver,
- the integer optimal value from Hoffman and Padberg (1993),
- the best GA solution value for each of the 10 trials,

In Table 3 we give, for each problem:

- the best solution reported by Levine (1994) (over all trials),
- the best solution found by our GA (over all trials),
- over the 10 trials the average percentage deviation (σ) from integer optimal of the best GA solution value,
- over the 10 trials the average number of feasible (non-duplicate) children generated,
- over the 10 trials the average solution time (in CPU seconds), which is the time that the GA takes to first reach the final best solution,

Table 1. SPP test problems details and reduction comparison.

	Original			Hoffman & Padberg			PREP		
Problem	Rows	Columns	Density	Rows	Columns	Density	Rows	Columns	Density
NW01	135	51975	5.86	135	50069	5.87	135	49903	5.87
NW02	145	87879	5.66	145	85258	5.68	145	85256	5.68
NW03	59	43749	14.13	59	38964	14.12	53	38956	15.45
NW04	36	87482	20.22	36	46190	20.20	35	46189	20.49
NW05	71	288507	10.06	71	202603	10.07	62	202594	11.35
NW06	50	6774	18.17	50	5977	18.27	38	5956	19.81
NW07	36	5172	22.12	36	3108	21.86	34	3105	22.71
NW08	24	434	22.39	34	356	22.36	21	352	25.55
NW09	40	3103	16.20	40	2305	16.05	38	2301	16.64
NW10	24	853	21.18	24	659	21.25	21	655	23.34
NW11	39	8820	16.64	39	6488	16.81	34	6482	18.75
NW12	27	626	20.00	27	454	19.06	25	451	14.66
NW13	51	16043	12.78	51	10905	12.57	50	10903	12.68
NW14	73	123409	10.04	73	95178	10.11	70	95172	10.44
NW15	31	467	19.55	29	463	21.00	29	405	20.56
NW16	139	148633	7.27	139	138951	7.23	135	138947	7.39
NW17	61	118607	13.96	61	78186	13.96	54	78179	15.34
NW18	124	10757	6.82	124	8460	6.83	110	8439	6.96
NW19	40	2879	21.88	40	2145	21.59	32	2134	21.92
NW20	22	685	24.70	22	566	25.00	22	536	25.00
NW21	25	577	24.89	25	426	24.33	25	421	24.32
NW22	23	619	23.87	23	531	24.10	23	520	24.09
NW23	19	711	24.80	18	473	24.87	18	423	24.38
NW24	19	1366	33.20	19	925	33.19	19	926	33.22
NW25	20	1217	30.16	20	844	30.15	20	844	30.15
NW26	23	771	23.77	18	473	23.12	21	464	25.02
NW27	22	1355	31.55	22	926	30.76	22	817	30.22
NW28	18	1210	39.27	18	825	38.43	18	580	35.95
NW29	18	2540	31.04	18	2034	30.99	18	2034	30.99
NW30	26	2653	29.63	26	1884	29.80	26	1877	29.78
NW31	26	2662	28.86	26	1823	29.21	26	1728	28.86
NW32	19	294	24.30	18	251	25.81	18	251	25.81
NW33	23	3068	30.76	23	2415	30.75	23	2308	30.47
NW34	20	899	28.06	20	750	28.16	20	718	27.70
NW35	23	1709	26.70	23	1403	27.02	23	1132	26.32
NW36	20	1783	36.90	20	1408	36.14	20	1204	35.17

(*Continued on next page*.)

Table 1. (*Continued*).

NW37	19	770	25.83	19	639	25.89	19	639	25.89
NW38	23	1220	32.33	23	911	31.44	21	690	33.45
NW39	25	677	26.55	25	567	26.25	25	565	26.28
NW40	19	404	26.95	19	336	26.83	19	336	26.86
NW41	17	197	22.10	17	177	22.27	17	177	22.33
NW42	23	1079	26.33	23	895	26.05	23	795	25.92
NW43	18	1072	25.18	17	982	26.43	17	982	26.43
AA01	823	8904	1.00	607	7532	1.00	605	7399	1.03
AA02	531	5198	1.32	360	3846	1.54	360	3837	1.54
AA03	825	8627	1.00	537	6694	1.32	536	6657	1.13
AA04	426	7195	1.70	342	6122	1.80	342	6118	1.80
AA05	801	8308	0.99	521	6235	1.12	520	6206	1.12
AA06	646	7292	1.10	488	5862	1.21	485	5807	1.22
US01	145	1053137	9.10	90	370642	9.80	86	351018	10.47
US02	100	13635	14.13	45	9022	16.57	45	4617	14.82
US03	77	85552	18.40	53	27084	21.42	50	20171	20.10
US04	163	28016	6.52	112	6564	7.48	91	3732	8.45
KL01	55	7479	13.67	50	5957	13.47	47	5915	13.44
KL02	71	36699	8.16	69	16542	8.34	69	16542	8.34

- over the 10 trials the average execution time (in CPU seconds), which is the time that the GA takes to generate 100,000 non-duplicate child solutions,
- the best solution found using CPLEX,
- the number of nodes searched by CPLEX in its branch-and-bound search tree,
- the total running time (in CPU seconds) for CPLEX.

Examining these tables, we observe that:

1. The GA is able to find at least one feasible solution for all but four problems (NW01, AA01, AA03 and AA05). In fact, in 43 out of the 55 problems the GA is able to find the optimal solution in at least one trial. The consistency of this performance is demonstrated in 34 problems, in which the optimal solution is obtained in *every* trial.
2. Comparison with the results reported by Levine (1994) is difficult as the results for his parallel GA using an island model vary according to the number of subpopulations. Moreover, he only attempted a subset of 40 of the 55 test problems solved in this paper. The figures shown in Table 3 are the best solutions found by Levine across eight different values for the number of subpopulations (each run of his GA was also for 100,000 iterations). It is clear that the solution provided by our GA is *in all cases* at least as good as the solution found by Levine. This is *clear evidence* that our GA outperforms the GA presented by Levine.

Table 2. Computational results for test problems.

Problem	LP optimal	Integer optimal	Best GA solution in each of the 10 trials									
NW01	114852.0	114852	n/f	n/f	n/f	n/f	n/f	n/f	n/f	n/f	n/f	n/f
NW02	105444.0	105444	109368	109125	109560	109713	108816	109443	110184	110148	109677	110151
NW03	24447.0	24492	25095	24510	o	24495	24561	o	26448	24501	25086	25785
NW04	16310.7	16862	o	16876	16986	16970	16970	o	16970	o	17004	o
NW05	132878.0	132878	138150	138890	138878	138330	138636	138240	138888	137602	134170	135780
NW06	7640.0	7810	o	o	o	o	o	o	o	o	o	o
NW07	5476.0	5476	o	o	o	o	o	o	o	o	o	o
NW08	35894.0	35894	o	o	o	o	o	o	o	o	o	o
NW09	67760.0	67760	o	o	o	o	o	o	o	o	o	o
NW10	68271.0	68271	o	o	o	o	o	o	o	o	o	o
NW11	116254.5	116256	o	o	o	o	o	117585	o	o	o	o
NW12	14118.0	14118	o	o	o	o	o	o	o	o	o	o
NW13	50132.0	50146	o	50650	50152	o	50152	o	50332	50164	50152	50158
NW14	61844.0	61844	62532	62356	62724	62304	62696	62388	62918	62932	62262	62532
NW15	67743.0	67743	o	o	o	o	o	o	o	o	o	o
NW16	1181590.0	1181590	o	o	o	o	o	o	o	o	o	o
NW17	10875.7	11115	o	11133	o	11196	11133	o	o	o	o	11133
NW18	338864.3	340160	363820	345762	359160	345130	365398	358484	357646	359148	358550	385596
NW19	10898.0	10898	o	o	o	o	o	o	o	o	o	o
NW20	16626.0	16812	o	o	o	o	o	o	o	o	o	o
NW21	7380.0	7408	o	o	o	o	o	o	o	o	o	o
NW22	6942.0	6984	o	o	o	o	o	o	o	o	o	o
NW23	12317.0	12534	o	o	o	o	o	o	o	o	o	o

(*Continued on next page*.)

Table 2. (*Continued*).

NW24	5843.0	6314	o	o	o	o	o	o	o	o	o	o
NW25	5852.0	5960	o	o	o	o	o	o	o	o	o	o
NW26	6743.0	6796	o	o	o	o	o	o	o	o	o	o
NW27	9877.5	9933	o	o	o	o	o	o	o	o	o	o
NW28	8169.0	8298	o	o	o	o	o	o	o	o	o	o
NW29	4185.3	4274	o	o	o	o	o	o	o	o	o	o
NW30	3726.8	3942	o	o	o	o	o	o	o	o	o	o
NW31	7980.0	8038	o	o	o	o	o	o	o	o	o	o
NW32	14570.0	14877	o	o	o	o	o	o	o	o	o	o
NW33	6484.0	6678	o	6724	o	o	o	o	o	o	o	o
NW34	10453.5	10488	o	o	o	o	o	o	o	o	o	o
NW35	7206.0	7216	o	o	o	o	o	o	o	o	o	o
NW36	7260.0	7314	o	o	o	o	o	o	o	7322	o	o
NW37	9961.5	10068	o	o	o	o	o	o	o	o	o	o
NW38	5553.0	5558	o	o	o	o	o	o	o	o	o	o
NW39	9868.5	10080	o	o	o	o	o	o	o	o	o	o
NW40	10658.3	10809	o	o	o	o	o	o	o	o	o	o
NW41	10972.5	11307	o	o	o	o	o	o	o	o	o	o
NW42	7485.0	7656	o	o	o	o	o	o	o	o	o	o
NW43	8897.0	8904	o	o	o	o	o	o	o	o	o	o
AA01	55535.4	56138	n/f	n/f	n/f	n/f	n/f	n/f	n/f	n/f	n/f	n/f
AA02	30494.0	30494	30601	30704	30500	30639	31161	31616	30726	32165	31056	30601
AA03	49616.4	49649	n/f	n/f	n/f	n/f	n/f	n/f	n/f	n/f	n/f	n/f

(*Continued on next page*.)

Table 2. (*Continued*).

AA04	25877.6	26402	30115	28261	29779	28397	28585	29440	28442	28986	29791	29400
AA05	53735.9	53839	n/f	n/f	n/f	n/f	n/f	n/f	n/f	n/f	n/f	n/f
AA06	26977.2	27040	27932	28060	28118	28048	28022	28004	28608	28500	28355	27883
US01	9949.5	10022	12627	12317	11614	12237	10557	11640	11144	11875	11496	10921
US02	5965.0	5965	o	o	o	o	o	o	o	o	o	o
US03	5338.0	5338	o	o	o	o	o	o	o	o	o	o
US04	17731.7	17854	o	o	o	o	o	o	o	o	o	o
KL01	1084.0	1086	1088	o	o	o	1088	o	o	1090	o	o
KL02	215.3	219	o	220	o	220	226	o	220	220	o	220

o = optimal solution value.
n/f = no feasible solution found.

Table 3. Performance comparison of our GA versus Levine and CPLEX.

		GA					CPLEX		
Problem	Levine best solution	Best solution	Avg. % σ	Avg. no. of feasible soln's generated	Average solution time	Average execution time	Best solution	No. of nodes	Solution time
NW01	n/a	n/f	n/a	0	n/a	10435.9	o	0	127.4
NW02	n/a	108816	3.96	24703	15132.0	19392.4	o	0	143.6
NW03	25671	o	1.86	100000	2782.0	6398.9	o	2	27.1
NW04	n/a	o	0.36	52446	3458.6	8700.2	o	997	850.2
NW05	n/a	134170	3.67	100000	24914.5	42391.9	–	–	–
NW06	o	o	0	100000	326.4	1030.6	o	16	6.6
NW07	o	o	0	100000	47.6	442.6	o	0	1.2
NW08	o	o	0	100000	0.7	99.9	o	0	0.1
NW09	o	o	0	100000	14.2	369.0	o	0	0.7
NW10	n/f	o	0	100000	1.8	132.9	o	0	0.2
NW11	n/f	o	0.11	100000	69.8	905.3	o	1	2.5
NW12	o	o	0	100000	3.5	74.6	o	0	0.1
NW13	n/a	o	0.15	100000	389.4	1187.9	o	9	5.9
NW14	n/a	62262	1.16	100000	9782.0	20072.6	o	0	102.3
NW15	o	o	0	1802	1.4	101.1	o	0	0.1
NW16	n/a	o	0	100000	11810.5	116675.7	o	0	340.2
NW17	n/a	o	0.12	100000	5650.7	13999.1	o	23	150.8
NW18	n/f	345130	5.79	89336	1540.8	2033.4	o	3	11.1
NW19	o	o	0	100000	69.7	404.5	o	0	0.6
NW20	o	o	0	85814	3.2	119.7	o	5	0.2
NW21	o	o	0	95380	1.1	80.0	o	1	0.2
NW22	o	o	0	49026	0.5	90.2	o	3	0.2
NW23	o	o	0	19431	1.5	105.3	o	6	0.2
NW24	o	o	0	99438	4.9	144.1	o	5	0.3
NW25	o	o	0	100000	3.6	113.9	o	4	0.3
NW26	o	o	0	52931	4.4	87.4	o	2	0.1
NW27	o	o	0	81581	2.6	135.6	o	2	0.3
NW28	o	o	0	74176	3.0	121.2	o	4	0.3
NW29	o	o	0	54395	48.8	243.0	o	8	0.7
NW30	o	o	0	72783	28.1	260.1	o	4	0.8
NW31	o	o	0	73493	42.2	261.1	o	4	0.7
NW32	o	o	0	100000	1.9	56.3	o	18	0.2
NW33	o	o	0.07	63661	5.5	315.5	o	2	0.7
NW34	o	o	0	69252	2.2	141.6	o	1	0.2

(*Continued on next page.*)

Table 3. (*Continued*).

NW35	o	o	0	36175	4.5	178.3	o	2	0.4
NW36	o	o	0.01	11016	46.2	212.3	o	32	1.8
NW37	o	o	0	99094	2.3	108.0	o	2	0.2
NW38	o	o	0	33348	6.7	132.4	o	2	0.3
NW39	o	o	0	84923	1.1	106.7	o	4	0.2
NW40	o	o	0	100000	1.4	68.8	o	5	0.1
NW41	o	o	0	100000	0.9	44.7	o	2	0.1
NW42	o	o	0	15498	16.3	163.6	o	6	0.4
NW43	o	o	0	61399	6.5	123.2	o	1	0.2
AA01	n/f	n/f	n/a	0	n/a	3101.3	o	2373	10109.3
AA02	n/a	30500	1.58	5105	1145.4	1567.4	o	0	63.4
AA03	n/a	n/f	n/a	0	n/a	2810.3	o	12	341.2
AA04	n/f	28261	10.29	14854	1731.7	1890.4	26521	3645	6923.2
AA05	n/f	n/f	n/a	0	n/a	2670.1	o	136	276.3
AA06	n/a	27883	4.12	15651	2114.8	2439.4	o	40	209.5
US01	n/a	10557	16.17	12814	43683.5	74728.4	—	—	—
US02	n/a	o	0	44522	76.8	667.0	o	0	6.5
US03	n/a	o	0	27197	1350.2	3859.5	o	0	27.5
US04	n/a	o	0	20528	135.1	789.7	o	8	13.6
KL01	1095	o	0.07	32601	159.6	909.9	o	66	10.7
KL02	220	o	0.55	20288	485.7	2099.7	o	95	66.5

o = optimal solution value.
n/f = no feasible solution found.
— = see text for discussion.
n/a = not available.

3. The GA is capable of generating a large percentage of child solutions that are feasible for many problems. This indicates that the heuristic improvement operator is effective in constructing feasible solutions.
4. The problems (particularly the AA problems) with low density are generally more difficult for the GA in terms of its ability to generate feasible solutions. The GA failed to find any feasible solution in three out of six AA problems. The difficulty of the AA problems is largely due to the relative large number of rows (constraints) compared with other problems. The AA problems were also found to be more difficult using the CPLEX solver in terms of the number of nodes searched and the running time.
5. The CPLEX mixed integer solver finds optimal solutions for all the problems except NW05, US01 and AA04. For CPLEX many of the smaller problems are fairly easy to solve, with the integer optimal solution being found after only a small branch-and-bound tree search (indicated by the number of nodes searched). What makes these problems easy for CPLEX is that the number of integer values in the LP relaxation solution is

relatively high and the gap between the LP lower bound and the optimal value is relatively small (compare the LP and the integer optimal values in Table 2). CPLEX was unable to obtain any solution for NW05 and US01 because their memory requirements exceeded the memory capacity (48 MB) of our machine. For AA04, a feasible solution was found before the memory was exhausted.

5.3. Experimentation with other settings

5.3.1. Replacement strategy. Other computational experience showed that the feasibility of the solution could be improved for the difficult AA problems if a different replacement strategy was used. The convergence behaviour of the GA using worst-unfitness replacement, shown in figure 7, suggests that feasible solutions may be found more easily if this replacement strategy, rather than the ranking replacement strategy, is used even though it is likely to converge prematurely to poor-quality feasible solutions. To investigate this we experimented with the GA using worst-unfitness replacement on the six difficult AA problems. The results are shown in Table 4.

Comparing these results with those in Tables 2 and 3, we observe a marked improvement in the number of feasible solutions generated when the worst-unfitness replacement method is used. Feasible solution were found for problems AA03 and AA05 in all trials. But AA01 remains intractable, although we observed that the average best unfitness value was significantly reduced from 13 in ranking replacement to 3 in worst-unfitness replacement. This result seems to suggest that feasible, or near feasible, children are more likely to be generated from feasible, or near feasible parents. If this hypothesis is true, then in cases where feasible solutions are harder to find, the priority should be to focus the search on feasible solutions (i.e., by decreasing the average unfitness of the population). Once one feasible solution is found, it would be easier to find others. However, it is also not surprising to see that, for the problems for which both replacement methods always find feasible solutions (AA02, AA04 and AA06), the quality of the solutions obtained by using the worst-unfitness replacement method are inferior to those obtained by using the ranking replacement method.

Table 4. Results of the GA using worst-unfitness replacement.

Problem	Best GA solution in each of the 10 trials										Average $\% \sigma$	Average no. of feasible solutions generated
AA01	n/f	n/f	n/f	n/f	n/f	n/f	n/f	n/f	n/f	n/f	n/a	0
AA02	34428	33811	33522	32377	32914	32702	32630	33294	36398	32748	9.80	12058
AA03	57438	62395	58567	53822	57978	57721	56255	54932	51098	57671	14.38	4751
AA04	35758	35199	35869	32279	30454	31853	32597	33261	32334	33990	26.35	19211
AA05	58110	58969	64168	61277	56328	57685	59654	57379	59418	56175	9.43	3849
AA06	30355	32589	32165	32202	32090	31217	30891	29896	29255	30254	14.98	20154

n/f = no feasible solution found.

Table 5. Performance comparison of different crossover operators.

	Average % σ			
Problem	Uniform	One-point	Two-point	OR
NW11	0.11	0.20	0.44	0
NW13	0.15	0.80	0.73	0.80
NW18	5.79	7.15	5.30	10.20
NW33	0.07	0.13	0	0
NW36	0.01	0	0	0
AA02	1.58	3.76	2.41	1.62
AA04	10.29	13.89	12.12	10.11
KL01	0.07	0.03	0.09	0.20
KL02	0.55	0.73	1.51	1.65
Average	2.07	2.97	2.51	2.73

5.3.2. Crossover. Experimentation was carried out to compare the effectiveness of the GA using different crossover operators. For computational reasons this was done using a subset of nine of the problems considered in Tables 2 and 3 that are relatively difficult in terms of the ability of the GA to consistently find optimal solutions. We compared the uniform crossover operator, as in the results presented in Tables 2 and 3, with the restricted (Beasley and Chu (1996)) one-point and two-point crossover operators, and the OR crossover operator (where $C[j] = \max[P_1[j], P_2[j]]$).

In Section 4.4 we discussed the issue of the uniform crossover operator disrupting a good row coverage achieved by two parents chosen using matching selection. We might expect therefore the OR operator, where the child contains a column if it is in either of the parents, to work particularly well. This is not so as can be seen from the results given in Table 5.

On a problem by problem basis these results show no clear winner among any of these crossover techniques. Note however that the uniform crossover operator performs best on average.

5.3.3. GA components. We also experimented with our standard GA (as given in Algorithm 3) but with certain components omitted in an attempt to ascertain the importance of these components. Specifically we considered the standard GA: with static mutation but no adaptive mutation (step 8 in Algorithm 3); with no mutation (static or dynamic) at all (step 8 in Algorithm 3); and with no heuristic improvement operator (step 9 in Algorithm 3). The results are shown in Table 6 for the same subset of test problems as were considered in Table 5. As in Tables 2 and 3 we performed 10 trials in each case.

We also investigated whether the performance of our standard GA could be attributed solely to our heuristic improvement operator. In order to do this we modified our standard GA by replacing steps 6–8 in Algorithm 3 by $\{C \leftarrow \textit{a random child}\}$ where by random child we mean that each bit in the child has a probability of m/n of being set to one. As for

Table 6. Performance comparison of GA components.

	Average % σ				
Problem	GA	GA-ADMUT	GA-ALLMUT	GA-HEUR	HEUR
NW11	0.11	0.07	0.06	−(0)	21.33
NW13	0.15	2.09	2.61	−(0)	7.57
NW18	5.79	13.03	34.91	−(0)	41.19
NW33	0.07	0.14	0.02	−(0)	5.12
NW36	0.01	0.01	0.11	−(0)	0.06
AA02	1.58	2.70(8)	5.94(2)	−(0)	−(0)
AA04	10.29	11.37(6)	23.05(3)	−(0)	−(0)
KL01	0.07	0.18	0.57	−(0)	1.77
KL02	0.55	1.46	2.65	−(0)	13.06
Average	2.07(90)	3.09(84)	6.54(75)	−(0)	12.87(70)

GA: standard GA, as in Algorithm 3.
GA-ADMUT: standard GA, static mutation but no adaptive mutation.
GA-ALLMUT: standard GA, no mutation at all.
GA-HEUR: standard GA, no heuristic improvement operator.
HEUR: heuristic improvement operator only as applied to random children.

A number in brackets is the number of trials in which a feasible solution was returned. If no number in brackets is given then all 10 trials returned a feasible solution.

the results presented in Tables 2 and 3 we performed 100,000 iterations in each trial and performed 10 trials in all. The results are also shown in Table 6.

Table 6 clearly demonstrates that each of the components considered is necessary for the success of our GA. In particular note that the GA-HEUR and HEUR columns in that table indicate that whilst the heuristic improvement operator is essential for the GA to find feasible solutions it is not sufficient in itself to account for the success of our GA.

6. Conclusions

In this paper, we have developed a heuristic for the set partitioning problem based on genetic algorithms. Our main interest was to investigate the use of GAs for solving highly constrained problems, in which case we chose the SPP as the input to our GA. We proposed separate fitness and unfitness scores, an adaptive mutation scheme, a heuristic improvement operator, a new parent selection method and a new population replacement scheme to improve the performance of our GA. Experimental results have been presented to show that these approaches are indeed very promising. Optimal or near-optimal solutions can be obtained for many problems of even large size.

The heuristic improvement operator is specialised for the SPP, and although computationally expensive, it is an indispensable component and is highly effective in constructing feasible solutions. Of the fifty-five real-world problems we tested, only four problems failed to obtain a feasible solution in one of ten random trials. The GAs ability to generate feasible solutions is attributed to the special constraint handling technique we developed, in particular, the notion of separate fitness and unfitness scores and the ranking replacement method. These techniques are generalisable for the application of GAs to any constrained problem.

The adaptive mutation procedure is designed to prevent the population from being trapped in an infeasible region due to premature column convergence. Our experiments indicated that this additional mutation procedure can significantly enhance performance. This technique is also generalisable for the application of GAs to any constrained problem.

We have presented a matching selection scheme for parent selection. This scheme attempts to choose the second of the two parents so as to work towards constraint feasibility. This technique too is generalisable for the application of GAs to any constrained problem.

A preprocessing routine was performed to reduce the size of the problem in order to increase the computational efficiency of the GA for the SPP. Results indicated that considerable reduction can be made for these real-world test problems.

Despite some very encouraging results, we conclude that our GA, in its present form, is not competitive with the existing exact solvers, such as `CPLEX`, in both speed and quality for the problems tested. However, we believe that the GA may gain more advantages in cases when the gap between the LP relaxation value and the optimal integer value is large. This hypothesis is based upon the relative success of GAs as demonstrated on the problems considered in our other work (Beasley and Chu (1996), Chu (1997), Chu and Beasley (1997, 1998)), and the fact that in those cases the effectiveness of the branch-and-bound approach is limited due to the relative large gap between the LP and the optimal integer values.

References

Arabeyre, J.P., J. Fearnley, F.C. Steiger, and W. Teather. (1969). “The Airline Crew Scheduling Problem: A Survey,” *Transportation Science* 3, 140–163.

Atamtürk, A., G.L. Nemhauser, and M.W.P. Savelsbergh. (1995). “A Combined Lagrangian, Linear Programming and Implication Heuristic for Large-Scale Set Partitioning Problems,” *Journal of Heuristics* 1, 247–259.

Bäck, T., D.B. Fogel, and Z. Michalewicz (eds.). (1997). *Handbook of Evolutionary Computation*. Oxford University Press.

Baker, E.K. and M. Fisher. (1981). “Computational Results for Very Large Air Crew Scheduling Problems,” *OMEGA* 9, 613–618.

Balas, E. and M.W. Padberg. (1976). “Set Partitioning: A Survey,” *SIAM Review* 18, 710–760.

Balas, E. and M.W. Padberg. (1979). Set Partitioning—A Survey.” In N. Christofides, A. Mingozzi, P. Toth and C. Sandi (eds.), *Combinatorial Optimization*. John Wiley, pp. 151–210.

Beasley, D., D.R. Bull, and R.R. Martin. (1993a). “An Overview of Genetic Algorithms: Part I, Fundamentals,” *University Computing* 15, 58–69.

Beasley, D., D.R. Bull, and R.R. Martin. (1993b). “An Overview of Genetic Algorithms: Part II, Research Topics,” *University Computing* 15, 170–181.

Beasley, J.E. (1990). “OR-Library: Distributing Test Problems by Electronic Mail,” *Journal of the Operational Research Society* 41, 1069–1072.

Beasley, J.E. (1996). “Obtaining Test Problems via Internet,” *Journal of Global Optimization* 8, 429–433.

Beasley, J.E. and P.C. Chu. (1996). “A Genetic Algorithm for the Set Covering Problem,” *European Journal of Operational Research* 94, 392–404.

Chan, T.J. and C.A. Yano. (1992). “A Multiplier Adjustment Approach for the Set Partitioning Problem,” *Operations Research* 40, S40–S47.

Chu, P.C. (1997). “A Genetic Algorithm Approach for Combinatorial Optimisation Problems.” Ph.D. Thesis, University of London.

Chu, P.C. and J.E. Beasley. (1997). “A Genetic Algorithm for the Generalised Assignment Problem,” *Computers and Operations Research* 24, 17–23.

Chu, P.C. and J.E. Beasley. (1998). “A Genetic Algorithm for the Multidimensional Knapsack Problem.” *Journal of Heuristics* 4, 63–86.

Davis, L. and M. Steenstrup. (1987). “Genetic Algorithms and Simulated Annealing: An Overview.” In L. Davis (ed.), *Genetic Algorithms and Simulated Annealing*. Morgan Kaufmann, pp. 1–11.

Deb, K. and D.E. Goldberg. (1989). “An Investigation of Niche and Species Formation in Genetic Function Optimization.” In J.D. Schaffer (ed.), *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann, pp. 42–50.

Eben-Chaime, M., C.A. Tovey, and J.C. Ammons. (1996). “Circuit Partitioning via Set Partitioning and Column Generation,” *Operations Research* 44, 65–76.

Fisher, M.L. and P. Kedia. (1990). “Optimal Solution of Set Covering/Partitioning Problems Using Dual Heuristics,” *Management Science* 36, 674–688.

Fonseca, C.M. and P.J. Fleming. (1993). “Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization.” In S. Forrest (ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms.* Morgan Kaufmann, pp. 416–423.

Garfinkel, R.S. and G.L. Nemhauser. (1972). *Integer Programming.* Chap. 8, John Wiley & Sons Inc., pp. 302–304.

Gershkoff, I. (1989). “Optimizing Flight Crew Schedules,” *Interfaces* 19(4), 29–43.

Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

Goldberg, D.E., K. Deb, and J. Horn. (1992). “Massive Multimodality, Deception and Genetic Algorithms.” In R. Männer and B. Manderick (eds.), *Proceedings of the Second International Conference on Parallel Problem Solving from Nature (PPSN).* North-Holland, pp. 37–46.

Goldberg, D.E. and J. Richardson. (1987). “Genetic Algorithms with Sharing for Multimodal Function Optimization.” In J. J. Grefenstette (ed.), *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms.* Lawrence Erlbaum, pp. 41–49.

Harche, F. and G.L. Thompson. (1994). “The Column Subtraction Algorithm: An Exact Method for Solving Weighted Set Covering, Packing and Partitioning Problems,” *Computers and Operations Research* 21, 689–705.

Hoffman, K.L. and M. Padberg. (1993). “Solving Airline Crew Scheduling Problems by Branch-and-Cut,” *Management Science* 39, 657–682.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press.

Horn, J. and N. Nafpliotis. (1993). “Multiobjective Optimization Using the Niched Pareto Genetic Algorithm.” Report Number 93005, Illinois Genetic Algorithms Laboratory, University of Illinois, Urbana-Champaign.

Horn, J., N. Nafpliotis, and D.E. Goldberg. (1994). “A Niched Pareto Genetic Algorithm for Multiobjective Optimization.” In Z. Michalewicz, J.D. Schaffer, H.-P. Schwefel, H. Kitano, and D. Fogel (eds.), *Proceedings of the First IEEE Conference on Evolutionary Computation*. IEEE, pp. 82–87.

Levine, D. (1994). “A Parallel Genetic Algorithm for the Set Partitioning Problem.” Ph.D. Thesis, Department of Computer Science, Illinois Institute of Technology.

Levine, D. (1996). “A Parallel Genetic Algorithm for the Set Partitioning Problem.” In I.H. Osman and J.P. Kelly (eds.), *Meta-Heuristics: Theory and Applications.* Kluwer Academic Publishers, pp. 23–35.

Marsten, R.E. (1974). “An Algorithm for Large Set Partitioning Problems,” *Management Science* 20, 774–787.

Marsten, R.E. and F. Shepardson. (1981). “Exact Solution of Crew Scheduling Problems Using the Set Partitioning Model: Recent Successful Applications,” *Networks* 11, 165–177.

Michalewicz, Z. (1995). “A Perspective on Evolutionary Computation.” In X. Yao, (ed.), *Progress in Evolutionary Computation.* Springer-Verlag, pp. 73–89.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. MIT Press.

Oei, C.K., D.E. Goldberg, and S.J. Chang. (1991). "Tournament Selection, Niching, and the Preservation of Diversity." Report Number 91011, Illinois Genetic Algorithms Laboratory, University of Illinois, Urbana-Champaign.

Powell, D. and M.M. Skolnick. (1993). "Using Genetic Algorithms in Engineering Design Optimization with Nonlinear Constraints." In S. Forrest (ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms.* Morgan Kaufmann, pp. 424–431.

Reeves, C.R. (1993). *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific.

Richardson, J.T., M.R. Palmer, G. Liepins, and M. Hilliard. (1989). "Some Guidelines for Genetic Algorithms with Penalty Functions." In J.D. Schaffer (ed.), *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann, pp. 191–197.

Ryan, D.M. and J.C. Falkner. (1988). "On the Integer Properties of Scheduling Set Partitioning Models," *European Journal of Operational Research* 35, 422–456.

Schaffer, J.D. (1984). "Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms." Ph.D. Thesis, Department of Electrical Engineering, Vanderbilt University.

Schaffer, J.D. (1985). "Multiple Objective Optimization with Vector Evaluated Genetic Algorithms." In J.J. Grefenstette (ed.), *Proceedings of the First International Conference on Genetic Algorithms and their Applications.* Lawrence Erlbaum, pp. 93–100.

Sherali, H.D. and Y.H. Lee. (1996). "Tighter Representations for Set Partitioning Problems," *Discrete Applied Mathematics* 68, 153–167.

Smith, A.E. and D.M. Tate. (1993). "Genetic Optimization Using a Penalty-Function." In S. Forrest (ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms.* Morgan Kaufmann, pp. 499–505.

Spears, W.M. and K.A. DeJong. (1991). "On the Virtues of Parametized Uniform Crossover." In R. Belew and L. Booker (eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms.* Morgan Kaufmann, pp. 230–236.

Surry, P.D., N.J. Radcliffe, and I.D. Boyd. (1995). "A Multi-Objective Approach to Constrained Optimisation of Gas Supply Networks: The COMOGA Method." In T.C. Fogarty (ed.), *Evolutionary Computing: AISB Workshop*. pp. 166–180. Lecture Notes in Computer Science 993. Springer-Verlag.

Syswerda, G. (1989). "Uniform Crossover in Genetic Algorithms." In J.D. Schaffer (ed.), *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann, pp. 2–9.

Tasi, L.H. (1995). "The Modified Differencing Method for the Set Partitioning Problem with Cardinality Constraints," *Discrete Applied Mathematics* 63, 175–180.

Wedelin, D. (1995a). "An Algorithm for Large Scale 0-1 Integer Programming with Application to Airline Crew Scheduling," *Annals of Operations Research* 57, 283–301.

Wedelin, D. (1995b). "The Design of a 0-1 Integer Optimizer and its Application in the Carmen System," *European Journal of Operational Research* 87, 722–730.