A Comparative Study of a Penalty Function, a Repair Heuristic, and Stochastic Operators with the Set-Covering Problem

Thomas Bäck[1] Martin Schütz[2], and Sami Khuri[3],

[1] Informatik Centrum Dortmund,
Joseph-von-Fraunhofer-Straße 20, D–44227 Dortmund
E-mail: baeck@ls11.informatik.uni-dortmund.de

[2] Universität Dortmund
Fachbereich Informatik, LS XI, D–44221 Dortmund
E-mail: schuetz@ls11.informatik.uni-dortmund.de

[3] Department of Mathematics & Computer Science,
San José State University, One Washington Square,
San José, CA 95192-0103, U.S.A.
E-mail: khuri@sjsumcs.sjsu.edu

Abstract. In this paper we compare the effects of using various stochastic operators with the non-unicost set-covering problem. Four different crossover operators are compared to a repair heuristic which consists in transforming infeasible strings into feasible ones. These stochastic operators are incorporated in GENEsYs [2], the genetic algorithm we apply to problem instances of the set-covering problem we draw from well known test problems. GENEsYs uses a simple fitness function that has a graded penalty term to penalize infeasibly bred strings. The results are compared to a non GA-based algorithm based on the greedy technique. Our computational results are then compared, shedding some light on the effects of using different operators, a penalty function, and a repair heuristic on a highly constrained combinatorial optimization problem.

1 Introduction

The set-covering problem is a combinatorial optimization problem that can be used as a model for many practical problems. Applications can be mainly found in resource-selection and committee forming tasks. The problem consists of a finite set $E = \{e_1, e_2, \ldots, e_m\}$ and a family F of subsets S_i of E, $F = \{S_1, S_2, \ldots, S_n\}$, such that every element of E belongs to at least one subset in F, i.e., $\cup_{i=1}^{n} S_i = E$. The unicost minimum set-covering problem consists in finding a minimum-size subset $C \subseteq F$ such that $\cup_{S_i \in C} S_i = E$. Any set $Y \subseteq F$ with $\cup_{S_j \in Y} = E$ is said to cover E. The problem thus consists in finding a cover of E with minimal cardinality. The set-covering problem is NP-complete as can be shown by polynomially transforming the vertex cover problem into it (see [1], p. 392) or by the transformation from the exact cover by 3-sets [15].

In many applications, the subsets have different costs. The non-unicost set covering problem, denoted by mscp, consists in finding a cover of E with cheapest overall cost. In other words, if $c_1, c_2, \ldots, c_n$ denote the costs of $S_1, S_2, \ldots, S_n$, then it is required to find a cover C of E such that $\sum_{S_i \in C} c_i$ has the smallest possible value.

In our work, we study the effects of using different stochastic operators with a penalty-based fitness function and a repair heuristic for infeasible solutions on the mscp. More precisely, the traditional one- and two-point crossover, uniform crossover, and the recently suggested fusion crossover [6] are incorporated in the genetic algorithm GENEsYs. This package is based on Grefenstette's popular software package GENESIS [12]. The fitness function uses a graded penalty term to penalize infeasibly bred strings. Unlike other works, such as Beasley et al. [8], that use a tailor-made fitness function, a variable mutation rate, or Huang et al. [13], whose custom made algorithm targets specific instances of the mcsp, GENEsYs is not augmented with any a priori knowledge of the problem except for our fitness function. We investigate a repair method (following earlier work by Orvosh and Davis [20]) which guarantees feasibility of solutions by changing the genotype of strings that represent infeasible solutions.

Following the formal introduction of the mscp, a greedy heuristic for that problem is introduced. The paper then introduces the fitness function that is used in the subsequent sections and the repair heuristic we use to transform infeasible strings into feasible ones. In order to compare the effects of different genetic operators and the penalty approach vs. the repair method, the paper then focuses on the experimental results. The last section gives the conclusions obtained after the recording of the performance of the different stochastic operators and the penalty function with and without using the repair technique.

2 The Minimum Set-Covering Problem

As mentioned in the introduction, applications of the set-covering problem can be found in resource allocation and committee forming endeavours.

Practical applications of the mscp range from assignment applications such as assigning flying crews to particular flights [22] and fire companies to fire houses [26], to the simplification of boolean expressions [9].

As is the case with many NP-hard problems, one of the only two algorithms, short of exhaustive enumeration, that delivers optimum solutions is based on the branch-and-bound technique (the other being dynamic programming). Approximation algorithms include greedy techniques [10], simulated annealing [23], Lagrangian relaxation and sub-gradient optimization techniques [6] and GA-based heuristics [18]. The interested reader can find more references to the unicost and mscp including more applications in [5].

The mscp can be recast by using matrix notation. Elements of E and subsets of F can be represented by an m-row, n-column, zero-one matrix A, with elements a_{ij}, $1 \leq i \leq m$ and $1 \leq j \leq n$. The columns of A represent the subsets while the rows represent the elements of E. Thus, $a_{ij} = 1$ means that e_i is covered by S_j while $a_{ij} = 0$ signifies that e_i is not covered by S_j. The columns have costs $c_1, c_2, \ldots, c_n$. A solution to the mscp is represented by a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ in which $x_j = 1$ means that S_j (i.e., column j) is chosen while $x_j = 0$ signals the omission of S_j in the solution. The objective of the mscp is to cover the rows of A by a subset of columns at minimal cost.

The following is a formal definition of the mscp in which we make use of Stinson's terminology for combinatorial optimization problems [24].

Problem instance:

A set of objects $E = \{e_1, \ldots, e_m\}$ and a family of subsets of E, $F = \{S_1, \ldots, S_n\}$, such that $\cup_{i=1}^{n} S_i = E$. Each S_i costs c_i where $c_i > 0$ for $i = 1, \ldots, n$. Or equivalently, an $m \times n$ matrix

$$A = \begin{pmatrix} a_{11} & \ldots & a_{1n} \\ . & & . \\ . & & . \\ . & & . \\ a_{m1} & \ldots & a_{mn} \end{pmatrix}$$

such that $\sum_{j=1}^{n} a_{ij} \geq 1$ for $i = 1, 2, \ldots, m$.

Feasible solution:

A vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ where $x_i \in \{0, 1\}$, such that

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1$$

for $i = 1, 2, \ldots, m$.

Objective function:

A cost function $P(\mathbf{x}) = \sum_{j=1}^{n} c_j x_j$ where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is a feasible solution.

Optimal solution:

A feasible vector that gives the minimum cost, i.e., a vector that minimizes the objective function.

We proceed by considering the non-GA based greedy heuristic we call "greedy-mscp."

In the process of building the cover C of E, we need to devise a scheme for greedily choosing the next subset (or column), among the remaining competing subsets, to cover uncovered elements (or rows).

The greedy step can be implemented by choosing the least expensive subset that contains uncovered elements of E, or the subset that contains the maximum number of uncovered elements.

The greedy strategy we choose for "greedy-mscp" makes use of the cost-ratio $c_j/|S'_j|$ where $S'_j \subseteq S_j$ is the set of elements of S_j that are still uncovered. The algorithm chooses among the subsets that are still in contention, the one that yields the smallest cost-ratio.

The procedure has to keep updating the subsets that are still competing by removing from each one of them the elements that have already been covered by previously selected subsets. It is thus very possible to have subsets drop out of contention. If U denotes the set of uncovered elements of E, then S_j ceases being a candidate for inclusion in C if $S_j \cap U = \emptyset$ or equivalently $S'_j = \emptyset$ with $j \in J$, where J is the index set of subsets still competing for inclusion in C.

procedure *greedy-mscp*
  $U := E$; (* *set of uncovered elements* *)
  $C := \emptyset$; (* *cover to be constructed* *)
  $J := \{1, 2, \ldots, n\}$; (* *index set of unused subsets* *)
  for all $i \in \{1, \ldots, n\}$
    $S'_i := S_i$; (* S'_i *contains uncovered elements of* S_i *)
  $K := \emptyset$; (* *index set of subsets in* C *)
  while $U \neq \emptyset$
    choose $S_j \in F$ *where* $j \in J$ *such that*
      $\frac{c_j}{|S'_j|} = \min_{i \in J} \left\{ \frac{c_i}{|S'_i|} \right\}$;
    $C := C \cup \{S_j\}$; (* *include* S_j *in the cover set* *)
    $K := K \cup \{j\}$;
    $U := U - S_j$; (* *remove elements of* E *covered by* S_j *)
    $J := J - \{j\}$; (* S_j *has been chosen* *)
    for all $i \in J$
      $S'_i := S'_i - S_j$; (* *remove from* S'_i *elements that are covered by* S_j *)
      if $S'_i = \emptyset$ **then**
        $J := J - \{i\}$; (* *elements of* S'_i *are already covered* *)
    for all $i \in K$
      $S'_i = \cup_{t \in K, t \neq i} S_t$ (* S'_i *contains the covered elements not in* S_i *)
      if $S'_i = U$ **then** (* S_i *is a redundant subset* *)
        $C := C - \{S_i\}$;
        $K := K - \{i\}$;
return C

It is very possible to have a subset in C, $S_k \in C$ for instance, have all its elements covered by the union of other subsets in C. The subset is then redundant and should be removed from C. In other words, if $S_k = \cup_{i \in K, i \neq k} S_i$ where K is the index set for subsets that are in the cover C, then S_k is redundant and should be dropped out of C. Thus, each time a subset S_j with $j \in J$ is chosen to join C, every subset in the cover should be checked to see whether its elements are covered by the union of the other subsets in the cover. We point out that procedure *greedy-mscp* also checks the newly added subset for redundancy, which is not necessary. We did not delete this step from the algorithm for the sake of clarity. We note that when S_j with $j \in J$ is chosen to join C, its corresponding prime set, S'_j, is not needed anymore. Rather than discarding the prime sets, the heuristic uses them for detecting redundant subsets. More precisely, each prime set corresponding to a subset in the cover will contain the union of all other subsets in C excluding its own elements, i.e. $S'_i = \cup_{t \in K, t \neq i} S_t$ for every $i \in K$. The prime sets thus play a dual role; S'_i with $i \in J$ is the set of elements in S_i that are still uncovered by C. On the other hand, S'_i with $i \in K$ is the union of all other subsets in the cover.

As mentioned earlier, the procedure uses two index sets, J and K. The first contains the indices of the subsets that are still competing for inclusion in C, while K contains the indices of the subsets that are already in C. It is important to realize that due to the dropping of subsets from contention, and due to redundant subsets that are removed from the cover C, the union of J and K rarely yields $\{1, \ldots, n\}$.

To get a better understanding of the procedure, we make use of the problem instance of the unicost set-cover problem found in "Introduction to Algorithms" by Cormen et al. [10] (pp. 974–975). The problem instance will also be used in the next section to demonstrate the workings of the graded penalty we use in the fitness function. The example is first converted to a non-unicost problem instance by adding costs to the various subsets, and we then recast it by using the matrix notation.

Example: Consider the following problem instance with 12 elements and 6 subsets: $E = \{e_1, e_2, \ldots, e_{12}\}$ and $F = \{S_1, \ldots, S_6\}$, where $S_1 = \{e_1, e_2, e_5, e_6, e_9, e_{10}\}$, $S_2 = \{e_6, e_7, e_{10}, e_{11}\}$, $S_3 = \{e_1, e_2, e_3, e_4\}$, $S_4 = \{e_3, e_5, e_6, e_7, e_8\}$, $S_5 = \{e_9, e_{10}, e_{11}, e_{12}\}$ and $S_6 = \{e_4, e_8\}$. Suppose that $c_1 = 10$, $c_2 = 5$, $c_3 = 11$, $c_4 = 10$, $c_5 = 8$ and $c_6 = 2$ are the costs we assign to the subset in order to convert the original instance [10] into a non-unicost problem instance. The following 12×6 matrix represents the problem, where the first column costs 10, the second 5, and so on.

$$A^T = \begin{pmatrix} 1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0 \\ 0\,0\,0\,0\,0\,1\,1\,0\,0\,1\,1\,0 \\ 1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,1\,0\,1\,1\,1\,1\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1 \\ 0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0\,0 \end{pmatrix}$$

After the initialization phase of the procedure, S_6 is the first chosen subset since:

$$2/2 = \min\{10/6, 5/4, 11/4, 10/5, 8/4, 2/2\}\,.$$

At the end of the updating remaining steps in the while loop of the procedure we have:

$S_1' = \{e_1, e_2, e_5, e_6, e_9, e_{10}\}$, $S_2' = \{e_6, e_7, e_{10}, e_{11}\}$, $S_3' = \{e_1, e_2, e_3\}$, $S_4' = \{e_3, e_5, e_6, e_7\}$, $S_5' = \{e_9, e_{10}, e_{11}, e_{12}\}$, and $C = \{S_6\}$.

Note that $S_1' = S_1$, $S_2' = S_2$, and $S_5' = S_5$ since none of these subsets contain either e_4 or e_8, and $S_6' = \emptyset$.

The second pass through the while loop will choose S_2 since

$$5/4 = \min\{10/6, 5/4, 11/3, 10/4, 8/4\}$$

Thus, at the end of the of the second pass of the while loop we have:

$S_1' = \{e_1, e_2, e_5, e_9\}$, $S_3' = \{e_1, e_2, e_3\}$, $S_4' = \{e_3, e_5\}$, $S_5' = \{e_9, e_{12}\}$, and $U = \{e_1, e_2, e_3, e_5, e_9, e_{12}\}$. The cover consists of $C = \{S_2, S_6\}$ and $S_2' = \{e_4, e_8\}$ and $S_6' = \{e_6, e_7, e_{10}, e_{11}\}$.

The procedure then proceeds by including S_1, S_5 and S_4 in C, in that order. Each inclusion is followed by updating S_i', for $i = 1, \ldots, 6$, U, C, and the index sets J and K. The inclusion of S_4 in C in the last time through the while loop triggers the removal of e_3 from S_3', and e_7 from S_2'. But since these were the only elements remaining in S_3' and S_2', respectively, both sets are now empty. S_3 drops out of contention and is not any more considered for inclusion in the cover, while S_2 becomes a redundant subset since the other elements of C, namely S_1, S_4, S_5 and S_6, cover all the elements in S_2. Thus, S_2 is removed from the cover and the procedure returns $C = \{S_1, S_4, S_5, S_6\}$ which has a cost of 30 (notice that this is not the optimum because the cost of the cover $\{S_3, S_4, S_5\}$ is 29).

In the next section we discuss the various stochastic operators including the fitness function we use in our experimental runs.

3 Fitness Functions and Stochastic Operators

Prior to the application of the genetic algorithm to problem instances of the mscp, one needs to encode the problem. Recall that a solution to the mscp can be represented by $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ in which $x_j = 1$ means that S_j is in the cover, and $x_j = 0$ signals the omission of S_j from the cover. Each binary string of length n in the population can be thought of as being the components of a vector $\mathbf{x}$.

3.1 Fitness Functions

The fitness function we use in our comparisons makes use of graded penalty terms to penalize infeasibly bred strings. We remark that a string $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ might represent an infeasible solution, i.e., a collection of subsets that does not cover all elements of E. Rather than discarding the latter, we follow Richardson et al.'s conclusions [21], and allow infeasible strings to be part of the population but at a certain price. The infeasible string's strength relative to the other strings in the population is reduced by adding a penalty term to its fitness. The farther away from feasibility the string is, the higher its penalty term is. We also make sure that the best infeasible string is never superior to the weakest feasible one. We note that in the worst case, we need all the subsets to cover the elements of E. In other words, $B = \sum_{i=1}^{n} c_i$ is the least upper bound on all possible costs. Hence, B is added to any infeasible string's fitness. By using B as an offset term invoked for infeasible strings, we are sure that infeasible strings will never outperform feasible ones. We would like to point out that this approach has worked quite well with other combinatorial optimization problems [4, 16].

In our fitness function, U is the set of elements from E for which the subsets represented by $\mathbf{x}$ have failed to cover. Obviously, U is empty when $\mathbf{x}$ is feasible. The function considers the minimum cost required to transform the infeasible string into a feasible one:

$$f(\mathbf{x}) = \sum_{i=1}^{n} c_i x_i + s(B + \sum_{i=1}^{m} c_i^*) \tag{1}$$

where

$$s = \begin{cases} 0 & , \text{ if } \mathbf{x} \text{ is feasible} \\ 1 & , \text{ if } \mathbf{x} \text{ is infeasible} \end{cases}$$

and

$$c_i^* = \begin{cases} \min_{e_i \in U}\{c_j \mid e_i \in S_j\} & , \text{ if } e_i \in U \\ 0 & , \text{ if } e_i \notin U \end{cases}$$

This fitness function, which is based on the cheapest cost of covering every single uncovered element of E, is more graded than the one which uses the naive penalty term $s(B + c_{\max}|U|)$, where $c_{\max} = \max\{c_1, c_2, \ldots, c_n\}$ and $|U|$ is the cardinality of E (our first experiments were based on this simpler fitness function and demonstrated the need for a better penalty term).

While evaluating the graded penalty term, it is easy to transform an infeasible string into a feasible one by changing $x_j = 0$ to $x_j = 1$ when $c_i^* = \min_{e_i \in U}\{c_j \mid e_i \in S_j\}$ is calculated. This can be interpreted as a kind of repair heuristic and is also compared here with the greedy heuristic and the fitness function given above. Since the repaired string is always feasible, the fitness function in this case is simply $f'(\mathbf{x}) = \sum_{i=1}^{n} c_i x_i$.
Example: Consider the previous problem instance with $\mathbf{x} = (1, 0, 1, 0, 1, 0)$.

$$A \cdot (1, 0, 1, 0, 1, 0)^T = (2, 2, 1, 1, 1, 1, 0, 0, 2, 2, 1, 1)^T \quad .$$

So $\mathbf{x}$ is an infeasible string since e_7 and e_8 are not covered by $\{S_1, S_3, S_5\}$ and $U = \{e_7, e_8\}$. Now $e_7 \in S_2$ and $e_7 \in S_4$, so $c_7^* = \min\{c_2, c_4\} = 5$. Similarly, $e_8 \in S_4$ and $e_8 \in S_6$, i.e., $c_8^* = \min\{c_4, c_6\} = 2$. We compute $B = \sum_{i=1}^{6} c_i = 46$. Thus $f(\mathbf{x}) = 29 + 46 + (5 + 2) = 82$, while the naive penalty term $s(B + c_{\max}|U|)$ would yield $29 + 46 + 2 \cdot 11 = 97$. The repair heuristic would incorporate S_2 and S_6 into the solution and thus change $\mathbf{x}$ into $\mathbf{x}' = (1, 1, 1, 0, 1, 1)$, with $f'(\mathbf{x}) = 36$.

Next we consider the crossover operator we make use in our experimental runs.

3.2 Stochastic Operators

The first three crossover operators we use, one- and two-point [11, 14] and uniform crossover [25], need no introduction. The fusion crossover introduced by Beasley [8] was designed to generate an offspring which represented a *newer* solution when compared to one or two-point crossover. It is based on the "differences" of the parents and as pointed out by Beasley, is advantageous especially when both parents are quite similar. If $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ and $\mathbf{b} = (b_1, b_2, \ldots, b_n)$ are the parent strings with fitness $f_\mathbf{a}$ and $f_\mathbf{b}$, then $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ is the offspring where

for all $i \in \{1, \ldots, n\}$
  if $a_i = b_i$
  then $c_i := a_i$;
  else $c_i := \begin{cases} a_i \textit{ with probability } p = \frac{f_\mathbf{b}}{f_\mathbf{a} + f_\mathbf{b}} \\ b_i \textit{ with probability } 1 - p \end{cases}$

We note that when $a_i \neq b_i$, the offspring inherits the bit from the better of the parents with higher probability (because smaller fitness values correspond with better solutions).

4 Experimental Results

The genetic algorithm is used with the one-point, two-point, uniform, and fusion crossover operators and a crossover rate of 0.6. As suggested independently by Mühlenbein [19] and Bäck [3], the mutation rate $p_m = 1/n$ is determined by the problem dimension n. A population size $\lambda = 100$ and proportional selection with linear dynamic scaling (and a scaling window of five generations) complete the parameter settings of this standard genetic algorithm.

We apply the algorithm with the penalty-based fitness function (1) and the repair method to 50 problem instances from Beasley's OR-library [7]. The characteristics (problem sizes, matrix densities: percentage of ones in the matrix) of these problems are summarized in table 1.

Independently of the problem size, we applied each genetic algorithm configuration (crossover operator and fitness function) a total of ten experiments to each single problem instance, and each experiment was allowed to run for $5 \cdot 10^5$ function evaluations (i.e., 5,000 generations). The best final objective function value found within these ten experiments is reported in tables 2 (fitness function f without repairing) and 3 (fitness function f' with repairing). The results in these tables are also compared to the solution quality as obtained by our greedy heuristic as well as the best solution value known so far (columns 6 and 7 of the tables). From the experimental results, the following observations can be made:

Problem	m	n	Density (%)
4.1–4.10	200	1000	2
5.1–5.10	200	2000	2
6.1–6.5	200	1000	5
a.1–a.5	300	3000	2
b.1–b.5	300	3000	5
c.1–c.5	400	4000	2
d.1–d.5	400	4000	5
e.1–e.5	50	500	20

Table 1. Characteristics of test problems

- The fitness function f without repairing infeasible strings always yields a feasible solution to the set covering problem and thus confirms the usefulness of the graded penalty function approach. The final solution quality, however, always remains far from the best known objective function value and can neither compete with the solution quality reached by genetic heuristics using more problem-specific operators [8, 17] nor with the greedy heuristic.
- Repairing infeasible strings according to the minimum cost principle improves the results remarkably, such that final objective function values close to the best known solution are obtained. The repair procedure competes well with the greedy heuristic and yields better solutions in most cases
- The choice of the crossover operator has no significant impact on the quality of the results.
- Our simple greedy heuristic always outperforms the genetic algorithm without repairing.

The multimodal fitness function of the set covering problem includes many local optima, which are characterized by a relatively small number of ones distributed over a long bit string. As table 1 shows, the densities of most problem instances are very low. Different local optima are separated by a large Hamming distance and a variety of infeasible strings (i.e., the feasible region is not connected), and the objective function values of local optima might differ a lot. Consequently, the genetic algorithm sooner or later concentrates its search in the region of attraction of a local optimum and is not able to leave this region by means of the mutation operator. This might give the reason for which the genetic algorithm without repairing quickly gets stuck in relatively bad local optima.

In combination with a repair heuristic, which certainly represents a greedy technique, the volume-oriented search principle of a genetic algorithm is able to identify much better local optima than without repairing. This combination of global search and locally optimal repair performs even better than our greedy technique.

5 Discussion

The set covering problem certainly represents a very complicated, highly multimodal optimization problem with a complex, non-connected feasible region. Genetic algorithms normally require the incorporation of certain problem-specific heuristics to yield solutions of acceptable quality; see e.g. [8, 13, 18].

The simple penalty function approach to the set covering problem as presented in this paper confirmed this general impression by yielding solutions which, although being feasible in all cases, are far from the solution quality which is easily obtained by our greedy heuristic. A variation of the main search operator of genetic algorithms, the crossover operator, did not produce substantial improvements.

Incorporating a simple repair method into the evaluation of solutions, however, improved the solution quality remarkably and caused the genetic algorithm to encounter better solutions than the greedy heuristic in most cases. Although repairing infeasible solutions is almost a trivial operator, this technique received some attention in genetic algorithm research only recently [20]. Little is known about the frequency of repairing and further research is required to determine which portion of infeasible strings should be repaired.

Acknowledgements

The first two authors gratefully acknowledge support by the BMFT under project EVOALG, grant 01 IB 403 A (Thomas Bäck) and the DFG, grant Schw 361/5-2 (Martin Schütz). The third author would like to thank the California State University system for their research award GS850-851 that allowed him to conduct some preliminary work on set-covering problems. We are also very grateful for the help of Jörg Heitkötter, who organized the data files from OR-library and helped us with formatting the article.

References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison Wesley, Reading, MA, 1974.
2. Th. Bäck. GENEsYs 1.0. Software distribution and installation notes, Systems Analysis Research Group, LSXI, Department of Computer Science, University of Dortmund, Germany, July 1992. (Available via anonymous ftp to `lumpi.informatik.uni-dortmund.de` as file GENEsYs-1.0.tar.Z in /pub/GA/src).
3. Th. Bäck. The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature 2*, pages 85–94. Elsevier, Amsterdam, 1992.
4. Th. Bäck and S. Khuri. An evolutionary heuristic for the maximum independent set problem. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 531–535. IEEE Press, 1994.
5. E. Balas and S. M. Ng. On the set covering polytype: I. All the facets with coefficients in $\{0, 1, 2\}$. *Mathematical Programming*, 43:57–69, 1989.
6. J. E. Beasley. A lagrangian heuristic for set-covering problems. *Naval Research Logistics*, 37:151–164, 1990.
7. J. E. Beasley. OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.
8. J. E. Beasley and P. C. Chu. A genetic algorithm for the set covering problem. Submitted to *European Journal of Operational Research* for publication, 1994.
9. M. A. Breuer. Simplification of the covering problem with application ot boolean expressions. *Journal of the Association of Computing Machinery*, 17:166–181, 1970.
10. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, Cambridge, MA, 1990.
11. D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning.* Addison Wesley, Reading, MA, 1989.
12. J. J. Grefenstette. GENESIS: A system for using genetic search procedures. In *Proceedings of the 1984 Conference on Intelligent Systems and Machines*, pages 161–165, 1984.
13. Wen-Chih Huang, Cheng-Yan Kao, and Jorng-Tzong Horng. A genetic algorithm approach for set covering problems. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 569–574. IEEE Press, 1994.
14. K. A. De Jong. *An analysis of the behaviour of a class of genetic adaptive systems.* PhD thesis, University of Michigan, 1975. Diss. Abstr. Int. 36(10), 5140B, University Microfilms No. 76–9381.
15. R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–104. Plenum Press, New York, 1972.
16. S. Khuri, Th. Bäck, and J. Heitkötter. An evolutionary approach to combinatorial optimization problems. In D. Cizmar, editor, *Proceedings of the 22nd Annual ACM Computer Science Conference*, pages 66–73. ACM Press, New York, 1994.
17. D. M. Levine. A genetic algorithm for the set partitioning problem. In S. Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 481–487. Morgan Kaufmann Publishers, San Mateo, CA, 1993.
18. G. E. Liepins, M. R. Hilliard, J. Richardson, and M. Palmer. Genetic algorithms applications to set covering and traveling salesman problems. In Donald E. Brown and Chelsea C. White III., editors, *Operations Research and Artificial Intelligence: The Integration of Problem-Solving Strategies*, pages 29–57. Kluwer Academic Publishers, 1990.
19. H. Mühlenbein. How genetic algorithms really work: I. mutation and hillclimbing. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature 2*, pages 15–25. Elsevier, Amsterdam, 1992.
20. D. Orvosh and L. Davis. Shall we repair ? Genetic algorithms, combinatorial optimization, and feasibility constraints. In S. Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, page 650. Morgan Kaufmann Publishers, San Mateo, CA, 1993.
21. J. T. Richardson, M. R. Palmer, G. Liepins, and M. Hilliard. Some guidelines for genetic algorithms with penalty functions. In J. D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 191–197. Morgan Kaufmann Publishers, San Mateo, CA, 1989.
22. J. Rubin. A technique for the solution of massive set-covering problems with applications to airline crew scheduling. *Transportation Science*, 7:34–48, 1973.
23. S. Sen. Minimal cost set covering using probabilistic methods. In *Proceedings 1993 ACM/SIGAPP Symposium on Applied Computing*, pages 157–164, 1993.
24. D. R. Stinson. *An Introduction to the Design and Analysis of Algorithms.* The Charles Babbage Research Center, Winnipeg, Manitoba, Canada, 2nd edition, 1987.
25. G. Syswerda. Uniform crossover in genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann Publishers, San Mateo, CA, 1989.
26. W. Walker. Using the set-covering problem to assign fire companies to fire houses. *Operations Research*, 22:275–277, 1974.

Problem	1-point	2-point	uniform	fusion	greedy	Best known
4.1	1,030	1,030	1,003	829	436	429
4.2	1,208	1,152	1,008	880	533	512
4.3	1,135	1,079	1,025	962	540	516
4.4	1,042	1,064	981	1,003	512	494
4.5	1,006	1,058	1,034	1,018	520	512
4.6	1,130	1,224	993	1,106	605	560
4.7	912	943	847	1,026	447	430
4.8	1,159	1,090	861	1,002	525	492
4.9	1,404	1,246	1,257	1,263	671	641
4.10	1,046	1,153	1,079	1,083	533	514
5.1	714	757	630	656	269	253
5.2	827	809	663	696	330	302
5.3	741	638	606	540	232	226
5.4	806	664	717	585	250	242
5.5	850	784	600	637	218	211
5.6	717	743	673	663	227	213
5.7	786	836	710	845	310	293
5.8	806	799	680	756	311	288
5.9	776	805	686	822	292	279
5.10	912	777	723	835	278	265
6.1	317	338	310	324	144	138
6.2	294	309	299	351	157	146
6.3	323	365	268	291	157	145
6.4	344	302	298	303	141	131
6.5	383	376	355	355	186	161
a.1	897	955	760	886	264	253
a.2	842	868	745	786	272	252
a.3	787	823	681	730	245	232
a.4	848	704	694	821	251	234
a.5	773	746	705	875	248	236
b.1	342	253	245	290	74	69
b.2	262	278	253	267	78	76
b.3	291	272	282	275	85	80
b.4	303	324	290	262	83	79
b.5	311	296	259	279	75	72
c.1	1054	927	650	782	239	227
c.2	809	854	729	829	225	219
c.3	885	896	767	841	259	243
c.4	839	861	772	703	240	219
c.5	892	845	735	834	219	215
d.1	310	294	325	280	68	60
d.2	285	319	293	296	71	66
d.3	298	246	281	286	80	72
d.4	280	305	256	284	67	62
d.5	318	340	308	331	66	61
e.1	6	7	7	7	5	5
e.2	7	7	7	7	5	5
e.3	7	6	7	6	5	5
e.4	7	7	7	7	6	5
e.5	8	7	7	7	5	5

Table 2. Experimental results for the penalty-based fitness function and the greedy heuristic.

Problem	1-point	2-point	uniform	fusion	greedy	Best known
4.1	439	440	440	438	436	429
4.2	536	551	525	556	533	512
4.3	529	532	532	532	540	516
4.4	528	517	508	512	512	494
4.5	522	526	518	527	520	512
4.6	568	579	572	568	605	560
4.7	444	437	437	437	447	430
4.8	504	504	509	499	525	492
4.9	678	689	686	675	671	641
4.10	549	538	553	553	533	514
5.1	274	271	270	274	269	253
5.2	318	319	319	320	330	302
5.3	238	235	235	239	232	226
5.4	247	245	245	244	250	242
5.5	219	219	219	219	218	211
5.6	229	236	238	232	227	213
5.7	313	310	309	309	310	293
5.8	302	309	304	306	311	288
5.9	301	304	298	292	292	279
5.10	275	276	276	277	278	265
6.1	146	145	147	147	144	138
6.2	154	154	153	154	157	146
6.3	154	150	151	151	157	145
6.4	140	133	136	133	141	131
6.5	180	180	177	179	186	161
a.1	255	261	260	260	264	253
a.2	270	268	267	267	272	252
a.3	247	247	244	245	245	232
a.4	244	246	249	247	251	234
a.5	240	239	239	239	248	236
b.1	77	77	79	75	74	69
b.2	87	83	84	81	78	76
b.3	84	85	83	83	85	80
b.4	84	82	84	84	83	79
b.5	72	75	72	77	75	72
c.1	234	235	234	235	239	227
c.2	231	232	226	233	225	219
c.3	266	261	263	264	259	243
c.4	237	234	238	237	240	219
c.5	220	219	222	219	219	215
d.1	62	61	61	62	68	60
d.2	72	72	70	70	71	66
d.3	77	79	79	79	80	72
d.4	66	64	66	66	67	62
d.5	64	65	65	64	66	61
e.1	8	7	7	8	5	5
e.2	7	6	6	7	5	5
e.3	7	7	8	6	5	5
e.4	6	7	7	7	6	5
e.5	7	8	7	7	5	5

Table 3. Experimental results for the repair heuristic and the greedy heuristic.