European Journal of Operational Research 31 (1987) 85–93
North-Holland

An algorithm for set covering problem

J.E. BEASLEY
Department of Management Science, Imperial College, London SW7 2BX, England

Abstract: In this paper we present an algorithm for the set covering problem that combines problem reduction tests with dual ascent, subgradient optimisation and linear programming. Computational results are presented for problems involving up to 400 rows and 4000 columns.

Keywords: Set covering, optimisation

1. Introduction

In this paper we consider the set covering problem (SCP) which is the problem of covering the rows of a m-row, n-column, zero–one matrix (a_{ij}) by a subset of the columns at minimum cost. Formally the problem can be defined as follows: Let

$x_j = 1$ if column j (cost c_j) is in the solution,

$= 0$ otherwise,

then the program is

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

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

$$x_j \in (0, 1), \quad j = 1, \ldots, n. \tag{3}$$

Equation (2) ensures that each row is covered by at least one column and equation (3) is the integrality constraint. If the inequalities in equation (2) are replaced by equalities the resulting problem is called the set partitioning problem (SPP).

Both the SCP and the SPP are well-known problems in the literature and have applications in fields such as airline crew scheduling [1,13,14], location of emergency facilities [16,19], routing problems [11], assembly line balancing [17] and information retrieval [7]. Other application areas are given in [4].

Christofides and Korman [6] presented a computational survey of a number of different methods for the SCP and reported the solution of unicost ($c_j = 1$, $j = 1, \ldots, n$) problems involving up to 35 rows and 1075 columns (density 20%). Note here that the density of a SCP is the percentage of ones in the (a_{ij}) matrix.

Etcheberry [8] presented an algorithm based upon lagrangean relaxation of a subset of the covering constraints (equation (2)) with subgradient optimisation being used to decide the lagrange multipliers. He reported the solution of unicost problems involving up to 50 rows and 100 columns (densities from 6% to 25%).

Paixao [15] presented an algorithm based upon decomposition and state space relaxation and reported the solution of problems involving up to 400 rows and 4000 columns (density 2%).

Balas and Ho [3] reported that a reasonable lower bound for the SCP could be found by a dual ascent procedure together with subgradient optimisation to improve upon the lower bound obtained from the dual ascent procedure. They also reported that a considerable number of columns could be removed from the problem by a simple reduction test.

A key feature of their work was the use of cutting planes. Essentially they added to the original SCP additional constraints (of the same type

The author would like to acknowledge the help of Egon Balas and Maria-Cecilia Carrera of Carnegie-Mellon University who supplied some of the test problems used in this paper.

Received August 1983; revised November 1984 and July 1986

as shown in equation (2)) in order to increase the value of the linear programming relaxation of the (enlarged) SCP so that (eventually) it has the same value as the optimal SCP (integer) solution. They reported the solution of problems involving up to 200 rows and 2000 columns (density 2%).

In this paper we present a three stage algorithm (based upon their work) consisting of:

(1) a dual ascent procedure

(2) a subgradient procedure starting from an initial set of lagrange multipliers equal to the dual variables from stage (1)

(3) solving the dual of the linear programming relaxation of the SCP.

Note here that the third stage is achievable computationally because a comprehensive set of problem reduction tests are used to remove some rows, and a large number of columns, from the problem.

In the next section we outline each of the three stages given above and give the complete lower bound procedure for the problem.

2. Lower bound procedure

2.1. Upper bound

We used the heuristic given by Balas and Ho [3] to generate an upper bound (Z_{UB}) for the problem. They considered a heuristic of the greedy type where at each stage an uncovered row i is chosen and covered by choosing the column j which minimises some function $F(c_j$, number of uncovered rows in column j, j covers $i)$. Balas and Ho [3] considered five different forms for the function F, the details can be found in [3] and will not be repeated here.

2.2. Dual ascent procedure

Letting u_i ($\geqslant 0$, $i = 1,\ldots,m$) be the dual variables associated with equation (2) then the dual of the linear programming relaxation of the SCP (DLSCP) is given by

$$\text{maximise} \quad \sum_{i=1}^{m} u_i, \tag{4}$$

$$\text{subject to} \quad \sum_{i=1}^{m} u_i a_{ij} \leqslant c_j, \quad j = 1,\ldots,n, \tag{5}$$

$$u_i \geqslant 0, \qquad i = 1,\ldots,m. \tag{6}$$

We adopted a two pass procedure to the problem of generating a good lower bound for the SCP from a feasible solution for DLSCP. This procedure was modelled on the dual ascent procedure of Balas and Ho [3] and was as follows:

(1) Let $N_i = \sum_{j=1}^{n} a_{ij}$, i.e. N_i is the number of columns that cover row i.

(2) Let (u_i) be any set of values to be used as a starting point for the dual variables for DLSCP.

(3) Pass 1: consider the rows in decreasing N_i order and for each row i perform

$$u_i = \max\left[0,\ u_i + \min\left[0,\ \left[c_j - \sum_{k=1}^{m} u_k a_{kj} \,\middle|\, a_{ij} = 1,\ j = 1,\ldots,n\right]\right]\right]. \tag{7}$$

At the end of this pass the (u_i) will be feasible for DLSCP and we then attempt to improve them.

(4) Pass 2: consider the rows in increasing N_i order and for each row i perform

$$u_i = u_i + \min\left[c_j - \sum_{k=1}^{m} u_k a_{kj} \,\middle|\, a_{ij} = 1,\ j = 1,\ldots,n\right]. \tag{8}$$

At the end of this procedure we have a set of dual variables feasible for DLSCP and a corresponding lower bound as given by equation (4).

2.3. Lagrangean lower bound

Letting s_i ($\geqslant 0$, $i = 1,\ldots,m$) be the lagrange multipliers associated with equation (2) then the lagrangean lower bound program (LLBP) is given by

$$\text{minimise} \quad \sum_{j=1}^{n}\left(c_j - \sum_{i=1}^{m} s_i a_{ij}\right)x_j + \sum_{i=1}^{m} s_i, \tag{9}$$

$$\text{subject to} \quad x_j \in (0, 1), \quad j = 1,\ldots,n. \tag{10}$$

This program is easily solved—let C_j represent the coefficient of x_j in the objective function of LLBP (equation (9)) and (X_j) the solution values of the (x_j)—then we have that $X_j = 1$ if $C_j \leqslant 0$ ($= 0$ otherwise) and the lower bound, Z_{LB}, is given by

$$Z_{LB} = \sum_{j=1}^{n} C_j X_j + \sum_{i=1}^{m} s_i. \tag{11}$$

2.4. Subgradient procedure

(1) Set (s_i), the lagrange multipliers, equal to the values of the dual variables (u_i) at the end of the dual ascent procedure.

(2) Solve LLBP with the current set of multipliers (s_i) and let the solution be Z_{LB}, (X_j). Update Z_{max}, the maximum lower bound found, with Z_{LB}.

(3) Apply the heuristic of Balas and Ho [3], using the lagrangean costs (C_j), in an attempt to produce a feasible solution for the SCP. Update Z_{UB} accordingly.

(4) Stop if $Z_{max} = Z_{UB}$ (Z_{UB} is the optimal solution).

(5) Calculate the subgradients

$$G_i = 1 - \sum_{j=1}^{n} a_{ij} X_j, \quad i = 1, \ldots, m. \tag{12}$$

We found it computationally useful to adjust the subgradients, if they were not going to effectively contribute to the update of the multipliers, using

$$G_i = 0 \quad \text{if } s_i = 0 \text{ and } G_i < 0, \quad i = 1, \ldots, m. \tag{13}$$

(6) Define a step size T by

$$T = f(Z_{UB} - Z_{LB}) / \left(\sum_{i=1}^{m} G_i^2 \right), \tag{14}$$

where $0 \leqslant f \leqslant 2$ and update the lagrange multipliers by

$$s_i = \max(0,\ s_i + TG_i), \quad i = 1, \ldots, m. \tag{15}$$

(7) Go to step (2) to resolve the problem unless sufficient subgradient iterations have been performed, in which case stop.

In deciding a value for f (equation (14)) we followed the approach of Etcheberry [8] and Fisher [9,10] in setting $f = 2$ initially and then halving f if Z_{max} did not increase within a certain number of iterations. (We halved f if Z_{max} did not increase within 15 iterations in the computational results reported later). The subgradient procedure was halted when f fell below 0.005. We also found it useful to use a target value in equation (14) 5% above the upper bound Z_{UB}.

2.5. Complete lower bound procedure

The complete procedure for the problem is:

(1) generate an upper bound,

(2) use the dual ascent procedure to calculate an initial set of lagrange multipliers,

(3) use the subgradient procedure in an attempt to improve upon the bound derived from the dual ascent procedure,

(4) at the end of the subgradient procedure optimally solve DLSCP using a simplex algorithm.

At the end of this procedure we will have solved the linear programming relaxation of the SCP. We may not, of course, have found the optimal SCP (integer) solution and in the event of this occurring we resolve the problem using a tree search procedure.

We stated before that the effectiveness of the procedure depends upon a comprehensive set of problem reduction tests which can be used to remove some rows, and many columns, from the problem. In the next section we outline these reduction tests.

3. Problem reduction

Reduction tests for the SCP have been given by a number of authors in the literature (e.g. see [12]). In this section we will outline those tests which we found most effective (some of which have not been discussed in the literature before). We classify these reduction tests as relating either to the columns (variables) or to the rows (constraints) of the problem.

3.1. Column reduction

(1) *Column domination*

Any column j whose rows $[i \mid a_{ij} = 1,\ i = 1, \ldots, m]$ can be covered by other columns for a cost less than c_j can be deleted from the problem. A computationally effective version of this is given by defining

$$d_i = \min\left[c_j \mid a_{ij} = 1,\ j = 1, \ldots, n \right], \quad i = 1, \ldots, m. \tag{16}$$

Then any column j for which

$$c_j > \sum_{i=1}^{m} d_i a_{ij} \tag{17}$$

can be deleted from the problem since the rows covered by column j can be covered by a combination of other columns at a lower cost. We used

the Balas and Ho [3] heuristic (using their F function (iii)) to check all columns which were not deleted by equation (17).

Limited computational experience indicated that this reduction test was ineffective for unicost problems (as we might expect) and so, in the computational results reported later, this test was not used for such problems.

(2) *Lagrangean penalties*

As before let C_j be the lagrangean objective function (equation (9)) coefficient for column j for a given set of lagrange multipliers with Z_{LB} the associated lower bound. Then if

$$Z_{LB} + C_j > Z_{UB}, \quad C_j \geqslant 0, \tag{18}$$

we can eliminate column j from the problem (as to include it in the solution would force the lower bound above the best feasible solution found Z_{UB}). Similarly if

$$Z_{LB} - C_j > Z_{UB}, \quad C_j < 0, \tag{19}$$

we must have column j in the optimal solution. Note here that as $C_j \geqslant 0$ ($j = 1, \ldots, n$) in the optimal solution of LLBP (as the dual variables in the optimal solution of DLSCP correspond to optimal lagrange multipliers for LLBP) we would not expect this test (equation (19)) to be particularly effective computationally. Inspection of the output for the computational results reported later confirmed this expectation.

(3) *Dual ascent penalties*

We can use the dual ascent procedure to derive a penalty for any column j being in/out of solution. Let (u_i) be any set of values to be used as a starting point for the dual ascent procedure, (e.g. the set of multipliers associated with the best lower bound Z_{max}), then:

(a) *out-penalty*. Set c_j to infinity and apply the dual ascent procedure—if the bound obtained exceeds Z_{UB} then column j must be in the optimal solution (an out-penalty test since the bound relates to removing column j from the problem).

(b) *in-penalty*. Apply the dual ascent procedure with u_i fixed at zero for all rows i such that $a_{ij} = 1$. If the bound obtained $+c_j$ exceeds Z_{UB} then column j cannot be in the optimal solution and can be deleted from the problem.

We can strengthen this simple test as follows: we can assume that $c_j > 0$, $j = 1, \ldots, n$, since any column j with $c_j \leqslant 0$ is in the optimal solution and so that column, and the rows that it covers, can be removed from the problem. Then we observe that if a column is in the optimal solution at least one of the rows that it covers is covered by no other solution column (if not then we could remove the column and have a solution of lower cost—contradicting the optimality assumption).

Consider all rows covered by column j—for each such row k if a cover of the set of rows $[i \mid a_{ij} = 1, i \neq k, i = 1, \ldots, m]$ can be found with cost less than c_j then if column j is in the optimal solution row k is covered by no other solution column (by a similar reasoning to that above). Consequently we can temporarily delete all the columns (except j) which cover row k when calculating an in-penalty for column j. Computationally we used the Balas and Ho [3] heuristic (with F function (iii)) to construct covers.

Note here that this observation can also be used to remove columns if we definitely identify a column as being in the optimal solution (either as a result of one of the reduction tests given in this section or explicitly in the tree search (see later)).

(4) *Column inclusion*

If a row i is only covered by a single column in the (reduced) problem then the column that covers i must be in the optimal solution.

Computationally if we identified a column j which had to be in the optimal solution all the rows $[i \mid a_{ij} = 1, i = 1, \ldots, m]$ were eliminated from the problem and column j deleted (note therefore that we need to set $Z_{UB} = Z_{UB} - c_j$ and to recalculate any lower bounds).

3.2. Row reduction

For convenience let $M_i = [j \mid a_{ij} = 1, j = 1, \ldots, n]$ i.e. M_i is the set of columns that cover row i (note here that we exclude any columns deleted from the problem in defining M_i). Then the row reduction tests we used are given below.

(1) *Row domination*

Any row k for which $m_i \subset M_k$ for some row i ($\neq$ k) can be deleted from the problem (since any column used to cover row i automatically covers row k).

(2) *Row redundancy*

Consider the following program $P(k)$:

$$\text{minimise} \quad \sum_{j=1}^{n} c_j x_j, \tag{20}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \geqslant 1, \quad i \neq k, \; i = 1, \ldots, m, \tag{21}$$

$$\sum_{j=1}^{n} a_{kj} x_j = 0, \tag{22}$$

$$x_j \in (0, 1), \quad j = 1, \ldots, n. \tag{23}$$

This program is the same as the original SCP except for a single row k which is uncovered. Then it is clear that if the optimal value of this program $P(k)$ is greater than the optimal SCP solution value then row k can be deleted from the problem (since to have it uncovered is unprofitable and it is better to have it covered).

Note here that this program is related to the work that has been done on identifying redundant constraints in linear programming (see [5,18]).

Whilst it is clear that to optimally solve $P(k)$ would be difficult (since it is itself a SCP) we do have the condition that row k is redundant if we can find some lower bound upon the optimal value of $P(k)$ such that: optimal value of $P(k) \geqslant$ this lower bound $> Z_{UB} \geqslant$ the optimal value of the original SCP. Hence we can show that row k is redundant if we can obtain a lower bound for $P(k)$ which exceeds Z_{UB}.

Now since $P(k)$ is closely related to the original SCP we can use the dual ascent procedure to establish a lower bound for $P(k)$. This can be done as follows:

(a) let (u_i) be any set of values to be used as a starting point for the dual ascent procedure (again the set of multipliers associated with the best lower bound Z_{max} provide a convenient starting point)

(b) set c_j to infinity for all $j \in M_k$ and apply the dual ascent procedure with u_k fixed at zero—if the bound obtained exceeds Z_{UB} then row k can be deleted from the problem.

Note here that we may, in deleting a row, lose some information about the problem which could have been put to better use (e.g. if we deleted a row covered by just a single column where we would otherwise have identified the column as being in the optimal solution). However, examination of the computational results showed that typically we were deleting rows that contained a large number of columns and so we felt that any potential loss of information was outweighed by the advantage of deleting rows from the problem (which would leave us with a smaller problem to solve).

Note also that this reduction test cannot always be applied. For example, suppose that in the tree search to resolve the problem we have branched in such a way that at the current tree node the optimal solution to the SCP defined at that tree node is greater than Z_{UB}. In such a case applying the above reduction test may result in the removal of some rows that are *not* redundant in terms of the SCP defined at the current tree node. To avoid this problem we only apply this reduction test if the solution corresponding to the upper bound Z_{UB} is still feasible (and of value Z_{UB}).

(3) *Row splitting*

Split M_i—the set of columns that cover row i—into two sets, J_1 and J_2 (say) where $|J_1| \neq 0$, $|J_2| \neq 0$, $J_1 \cup J_2 = M_i$ and $J_1 \cap J_2 = \emptyset$. Let (u_i) be any set of values to be used as a starting point for the dual ascent procedure. Set c_j to infinity for all $j \in J_1$ and apply the dual ascent procedure—if the bound obtained exceeds Z_{UB} then we know that at least one column in J_1 must be in the optimal solution. Hence we can remove all the columns in J_2 from row i (but not from the entire problem) by setting $a_{ij} = 0$ for all $j \in J_2$ since they are obviously superfluous to the covering of row i. [Note here that this setting of elements of (a_{ij}) to zero may cause any dual feasible solution for the problem to become infeasible.]

Computationally we defined the sets J_1 and J_2 for each row i by considering the columns in M_i in ascending cost order and placing the first $|M_i|/2$ of these columns into J_1 and the remainder into J_2. We repeated this row splitting (splitting the columns for each row) operation until no further reduction for row i could be found—the rows being considered in descending $|M_i|$ order.

Note here that this test is a generalisation of the dual ascent out-penalty test from a single column to a set of columns and can be applied to any set of columns. The specific advantage of row

splitting is that if we identify a set of columns J_1 at least one of which must be in the optimal solution then we can make an immediate reduction in problem size.

Note too that since the inequality $\sum_{j \in J_1} x_j \geq 1$ is a member of the family of cuts defined by Balas [2] this test can be regarded as a special case of the cutting plane approach adopted by Balas [2] and Balas and Ho [3]. [The author is grateful to one of the referees for pointing out this connection].

4. Tree search procedure

As mentioned previously, in the event of the lower bound procedure terminating without finding the optimal SCP (integer) solution we resolve the problem using a tree search procedure. We used a binary depth-first tree search procedure computing at each tree node a lower bound for the optimal completion of the node. We indicate below how we structured the initial tree node and the tree search nodes with respect to the calculation of the bound and the reduction tests we carried out.

4.1. Initial tree node

(1) *Reduction*

We first reduced the problem using the column domination test.

(2) *Upper bound*

We found an initial feasible solution Z_{UB}, corresponding to an upper bound on the problem, by using the Balas and Ho [3] greedy heuristic.

(3) *Dual ascent*

The dual ascent procedure was used starting from initial values of $u_i = 0$, $i = 1, \ldots, m$, to obtain an initial lower bound.

(4) *Subgradient ascent*

The subgradient procedure was then carried out with, at each subgradient iteration, the reduction test relating to lagrangean penalties being performed. Each time f (equation (14)) was halved we recalled the multipliers associated with the best lower bound (Z_{max}) found so far, resolved LLBP with that set of multipliers and performed the reduction tests relating to column domination and column inclusion. At the end of the subgradient procedure the set of multipliers associated with the best lower bound found were recalled and LLBP resolved with this set of multipliers.

(5) *Reduction*

The reduction tests given before, except for the test relating to dual ascent penalties, were then all performed (together with the Balas and Ho [3] procedure for generating an upper bound) until no further reduction could be achieved. We found that the test relating to dual ascent penalties was relatively expensive computationally and so this test was only performed once at this step.

(6) *DLSCP solution*

The dual of the linear programming relaxation of the remaining (reduced) SCP was solved using a suitable algorithm. In the computational results reported later we used the NAG library routine E04MBF. Note here that because of the reduction tests the value obtained may not coincide with the value of the dual of the linear programming relaxation of the original (unreduced) SCP.

(7) *Reduction*

LLBP was resolved using the optimal set of lagrange multipliers obtained from the linear programming solution and step (5) above repeated.

4.2. Other tree nodes

(1) *Lower bound*

At each tree node we started with an initial set of lagrange multipliers equal to the set associated with the best lower bound found at the predecessor tree node. Subgradient ascent was carried out in the same manner as at the initial tree node except that:

(a) 30 subgradient iterations were carried out, with f being halved every five iterations, at tree nodes associated with forward branching, but both these figures were doubled for tree nodes associated with backtracking,

(b) we did not perform the reduction test relating to column domination (but did carry out a test in its place to eliminate columns that were no longer needed in that they did not cover any uncovered row),

(c) we only performed the reduction test relating to column inclusion twice at each tree node,

before the ascent and at the end of the ascent,

(d) we only used the heuristic of Balas and Ho [3] (with lagrangean costs (C_j)) at the end of the ascent at each tree node.

(2) *Branching*

In forward branching we first chose the (uncovered) row i with the maximum s_i value for non-unicost problems and the (uncovered) row i with the minimum $|M_i|$ value for unicost problems (as suggested by Paixao [15]).

We then branched by choosing the column j (from M_i) with the minimum C_j value. Intuitively this corresponds to choosing the column that is most likely to cover row i in the optimal completion of the current tree node.

(3) *Backtracking*

We can backtrack in the tree when, at any tree node, $Z_{max} > Z_{UB}$ or when the reduction tests indicate that there is no feasible completion of the current tree node (e.g. as may happen if there is no column available to cover some row).

5. Computational results

The algorithm presented in this paper was programmed in FORTRAN and run on a Cray-1S using the CFT compiler (with maximum optimisation) for a number of test problems drawn from the literature and some randomly generated problems.

The Cray-1S is an example of what is called a 'supercomputer' in that, for certain calculations, it is much faster than conventional computers. For example, adding together two one-dimensional vectors, each of length h, would involve $O(h)$ operations on a conventional computer but would involve only (essentially) O(1) operations on the Cray-1S. This ability to substantially speed up certain vector calculations is known as the vector processing capability of the Cray-1S.

The primary use made of this vector processing ability in the algorithm presented in this paper was in relation to the calculation of the lower bound and subgradients.

Table 1 presents a summary of the test problems, with problem sets 4, 5 and 6 being taken from Balas and Ho [3] and problem sets A to E being randomly generated.

The test problems were produced using the scheme of Balas and Ho [3], namely the column costs (c_j) were integers randomly generated from [1, K] (some K (shown in Table 1)); every column covered at least one row; and every row was covered by at least two columns. Note here that problems of this size are as large, if not larger, than those solved by other authors [3,6,8,15].

Table 2 gives the results for problems that terminated at the initial tree node and so did not require branching and Table 3 the results for problems that required branching. As in Balas and Ho [3] we use 4.1 to refer to the first problem in problem set 4, 4.2 to refer to the second problem, etc.

In Table 3 we give, for each problem, the best

Table 1
Test problem details

Problem set	Number of rows (m)	Number of columns (n)	Density	K	Number of problems in problem set
4	200	1000	2%	100	10
5	200	2000	2%	100	10
6	200	1000	5%	100	5
A	300	3000	2%	100	5
B	300	3000	5%	100	5
C	400	4000	2%	100	5
D	400	4000	5%	100	5
E	50	500	20%	1 (unicost)	5

Table 2
Results for problems not requiring branching

Problem number	Number of subgradient iterations	Optimal value	Total time Cray-1S seconds
4.1	206	429	2.3
4.2	169	512	2.7
4.3	149	516	2.5
4.5	107	512	1.9
4.7	172	430	2.2
4.9	385	641	6.3
4.10	131	514	1.8
5.3	126	226	2.7
5.5	145	211	2.5
5.6	116	213	2.4
5.8	466	288	5.9
5.9	169	279	2.9
5.10	142	265	2.9

Table 3
Results for problems requiring branching

Problem number	Setup/initial subgradient ascent/reduction						Linear program/reduction				Optimal value	Number of tree nodes	Total time Cray-1S seconds
	Number of subgradient iterations	Rows	Columns	Maximum lower bound (Z_{max})	Best upper bound (Z_{UB})	Time Cray-1S seconds	DLSCP solution value	Rows	Columns	Time Cray-1S seconds			
4.4	300	43	42	493.82	496	3.5	494.00	43	42	0.2	494	9	3.8
4.6	392	87	74	557.04	561	5.3	558.50	43	38	0.5	560	5	5.9
4.8	452	122	99	488.56	495	5.7	488.67	122	98	1.6	492	11	7.6
5.1	426	170	161	251.14	257	9.2	251.23	169	159	3.3	253	21	13.3
5.2	460	193	248	299.56	313	11.9	299.76	193	248	4.6	302	95	21.1
5.4	289	119	121	240.48	245	6.5	240.50	115	117	1.4	242	19	8.4
5.7	411	63	62	291.67	294	5.3	292.11	48	44	0.4	293	7	5.8
6.1	401	200	173	133.04	145	12.7	133.14	200	173	3.9	138	195	25.4
6.2	362	200	194	140.34	157	13.2	140.46	200	192	5.9	146	171	27.6
6.3	380	175	95	139.56	145	8.0	141.72	149	75	2.6	145	15	11.1
6.4	331	96	61	128.86	131	5.5	129.00	78	53	0.7	131	3	6.2
6.5	350	199	205	152.85	170	15.2	153.35	199	201	6.5	161	271	35.7
A.1	382	300	381	246.49	273	22.0	246.84	300	381	12.3	253	515	77.6
A.2	404	298	344	247.23	259	26.8	247.50	298	340	14.4	252	809	104.9
A.3	403	300	372	227.81	245	28.9	228.00	300	372	11.4	232	857	110.8
A.4	432	275	218	231.11	236	19.5	231.40	274	209	8.3	234	63	31.1
A.5	382	250	221	234.82	240	17.4	234.89	250	219	7.0	236	79	28.4
B.1	457	300	267	64.27	72	34.3	64.54	300	263	16.0	69	411	82.1
B.2	394	300	370	69.10	80	43.3	69.30	300	367	23.7	76	3273	396.8
B.3	388	300	254	73.93	81	30.7	74.16	300	248	14.1	80	1383	149.6
B.4	428	300	429	70.96	85	57.9	71.22	300	428	28.5	79	4381	564.0
B.5	429	300	234	67.57	74	29.5	67.67	300	233	13.0	72	661	90.5
C.1	348	400	503	223.60	242	44.2	223.80	400	503	20.5	227	1265	203.4
C.2	388	400	545	212.52	231	49.2	212.85	400	545	22.4	219	275	113.1
C.3	492	400	557	234.31	252	62.2	234.58	400	555	33.6	243	15757	2042.5
C.4	381	400	542	213.48	233	48.2	213.85	400	542	23.6	219	493	130.0
C.5	371	400	422	211.36	220	43.8	211.63	399	415	23.4	215	375	107.6
D.1	375	400	465	55.09	66	81.7	55.31	400	460	45.4	60	3393	587.0
D.2	366	400	482	59.02	70	82.8	59.35	400	469	48.3	66	6739	1056.2
D.3	479	400	529	64.88	77	106.4	65.07	400	523	56.2	72	19707	2971.4
D.4	415	400	524	55.55	68	94.6	55.84	400	518	55.7	62	12389	1947.9
D.5	423	400	226	58.49	63	42.5	58.62	400	223	20.7	61	147	76.1
E.1	376	50	221	3.46	5	18.3	3.48	50	200	4.8	5	69	28.1
E.2	351	50	374	3.37	5	21.9	3.38	50	365	10.1	5	145	46.5
E.3	353	50	457	3.28	5	23.5	3.30	50	451	13.6	5	231	62.9
E.4	358	50	306	3.43	5	19.5	3.45	50	297	7.6	5	129	38.7
E.5	290	50	375	3.37	5	19.9	3.39	50	357	11.0	5	167	47.8

lower bound (Z_{max}) and the best upper bound (Z_{UB}) found at the end of the initial subgradient ascent/reduction, together with the number of rows remaining (excluding those already covered or eliminated as dominated or redundant) and the number of columns remaining. The value of the dual of the linear programming relaxation of this remaining (reduced) SCP is given together with the number of rows and columns left after the reductions based on the optimal set of lagrange multipliers obtained from this linear programming solution have been performed. The optimal SCP (integer) solution value, together with the number of tree nodes and the total time, in Cray-1S seconds, is also given. Note here that the time presented includes input/output time and the time needed to generate the problem.

Comparing Tables 2 and 3 with the corresponding tables in Balas and Ho [3] for the 25 test problems in problem sets 4, 5 and 6 then it is clear that the algorithm presented in this paper has an equal, or superior, performance to the algorithm of Balas and Ho [3] on the majority of these test problems.

References

[1] Baker, E.K., Bodin, L.D., Finnegan, W.F., and Ponder, R.J. (1979), "Efficient heuristic solutions to an airline crew scheduling problem", *AIIE Transactions* 11, 79–85.

[2] Balas, E. (1980), "Cutting planes from conditional bounds: a new approach to set covering", *Mathematical Programming Study* 12, 19–36.

[3] Balas, E., and Ho, A. (1980), "Set covering algorithms using cutting planes, heuristics, and subgradient optimisation: A computational study", *Mathematical Programming Study* 12, 37–60.

[4] Balas, E., and Padberg, M.W. (1979), "Set partitioning—A survey", in: N. Christofides, (ed.), *Combinatorial Optimisation*, Wiley, New York.

[5] Boot, J.C.G. (1962), "On trivial and binding constraints in programming problems", *Management Science* 8, 419–441.

[6] Crhistofides, N., and Korman, S. (1975), "A computational survey of methods for the set covering problem", *Management Science* 21, 591–599.

[7] Day, R.H. (1965), "On optimal extracting from a multiple file data storage system: an application of integer programming", *Operations Research* 13, 482–494.

[8] Etcheberry, J. (1977), "The set-covering problem: A new implicit enumeration algorithm", *Operations Research* 25, 760–772.

[9] Fisher, M.L. (1981), "The lagrangean relaxation method for solving integer programming problems", *Management Science* 27, 1–18.

[10] Fisher, M.L. (1985), "An applications oriented guide to lagrangean relaxation", *Interfaces* 15 (2), 10–21.

[11] Foster, B.A., and Ryan, D.M. (1976), "An integer programming approach to the vehicle scheduling problem", *Operational Research Quarterly* 27, 367–384.

[12] Garfinkel, R.S., and Nemhauser, G.L. (1972), *Integer Programming*, Wiley, New York.

[13] Marsten, R.E. (1974), "An algorithm for large set partitioning problems", *Management Science* 20, 774–787.

[14] Marsten, R.E., and Shepardson, F. (1981), "Exact solutions of crew scheduling problems using the set partitioning problem: Recent successful applications", *Networks* 11, 165–177.

[15] Paixao, J. (1984), "Algorithms for large scale set covering problems", PhD. thesis, Department of Management Science, Imperial College, London SW7 2BX, England.

[16] Revelle, C., Marks, D., and Liebman, J.C. (1970), "An analysis of private and public sector location models", *Management Science* 16, 692–707.

[17] Salveson, M.E. (1955), "The assembly line balancing problem", *Journal of Industrial Engineering* 6, 18–25.

[18] Telgen, J. (1983), "Identifying redundant constraints and implicit equalities in systems of linear constraints", *Management Science* 29, 1209–1222.

[19] Walker, W. (1974), "Application of the set covering problem to the assignment of ladder trucks to fire houses", *Operations Research* 22, 275–277.