A MULTIPLIER ADJUSTMENT APPROACH FOR THE SET PARTITIONING PROBLEM

Thomas Justin Chan
Solution Technoligies Consulting, Ltd.
Vancouver, B.C., Canada V5Z 1E9

and

Candace Arai Yano
Department of Industrial & Operations Engineering
University of Michigan
Ann Arbor, MI 48109-2117

Technical Report 87-12

Revised June 1990

A MULTIPLIER ADJUSTMENT APPROACH FOR THE SET PARTITIONING PROBLEM

by
Thomas Justin Chan*
and
Candace Arai Yano**

*Solution Technologies Consulting, Ltd.
Vancouver, B.C., Canada V5Z 1E9 †

**Department of Industrial and Operations Engineering
The University of Michigan
Ann Arbor, Michigan 48109-2117

December, 1988
Revised June 1990

† Part of this work was done while this author was in the Department of Computer Science and Engineering at Southern Methodist University

Abstract

We introduce an effective branch-and-bound algorithm for solving the set partitioning problem. The new algorithm employs a new multiplier-adjustment-based bounding procedure, and a complementary branching strategy which results in relatively small search trees. Computational results based on 20 moderately sized crew scheduling problems indicate that our new algorithm is on average 16.6 times faster than the popular code, SETPAR. The improvements are mainly due to the bounding procedure, which is fast, easy to use, and provides tight lower bounds. On average, the bounds are 97.6% of the optimal objective value of the linear programming relaxation after only five iterations, and 98.5% after ten iterations. Moreover, the lower bounds are observed to be monotonically nondecreasing. We also apply the technique of variable elimination, which is very effective in reducing the size of the problems. On average, 89% of the variables are eliminated from the problem at the root node.

The set partitioning problem (SPP) is a zero-one integer program formulated as follows:

$$
\begin{array}{lll}
\text{(P)} \quad \text{minimize} & \sum_{j=1}^{n} C_j x_j & \\
\text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j = 1 & i \in I \qquad (1) \\
& x_j = \{0,1\} & j \in J \\
\text{where} & a_{ij} \in \{0,1\} & i \in I,\ j \in J \\
& I = \{1,2,\ldots,m\} & \\
& J = \{1,2,\ldots,n\}. &
\end{array}
$$

The SPP has been the focus of study by many researchers because of its simple structure and numerous practical applications. Among the applications described in the literature are: crew scheduling (Marsten and Shepardson 1981), truck scheduling (Balinski and Quant 1964), information retrieval (Day 1965), circuit design (Root 1964), capacity balancing (Steinman and Schwinn 1969), capital investment (Valenta 1969), facility location (Revelle, Marks and Liebman 1970), political districting (Garfinkel and Nemhauser 1970), and radio communication planning (Thuve 1981). Other applications of the SPP are given in the surveys by Garfinkel and Nemhauser (1972), and Salkin (1975).

The two most published approaches for the SPP are implicit enumeration and simplex-based cutting planes. (A survey of these and other approaches is provided by Balas and Padberg 1979.) Implicit enumeration is the more promising and widely used of the two because it takes full advantage of the special structure of the SPP. Branching is usually performed by either *fixing* a variable $x_j = 1$ to satisfy a particular constraint (Pierce 1968, Garfinkel and Nemhauser 1969, and Fisher and Kedia 1986) or restricting the set from which a variable may be selected to satisfy a particular constraint (Marsten 1974).

One means of enhancing the performance of implicit enumeration is to reduce the size of the search tree. This is usually achieved by calculating a lower bound on the cost of completion for each partial solution. To accomplish this, linear programming (LP) relaxation

is often used (Michaud 1972). The reason for its popularity is that the solution of the LP relaxation is frequently integral or provides a very tight lower bound (Balinski and Quandt 1964). However, solving a LP for each partial solution is computationally impractical for most problems. Moreover, LPs derived from large SPPs are often extremely difficult to solve because they are highly degenerate. (This difficulty has been reported by Marsten 1974 and other researchers.) For these reasons, many researchers have resorted to non-LP methods to obtain lower bounds.

To avoid solving a LP for each partial solution, Pierce and Lasky (1973) and Lu (1970) solve a knapsack problem obtained by adding up all the unsatisfied constraints, while Christofides and Korman (1973) solve an auxiliary problem using dynamic programming techniques. Etcheberry (1977) develops a more successful bounding technique by using Lagrangian relaxation (Fisher 1981 and Geoffrion 1974) and subgradient optimization (Sandi 1979). Fisher and Kedia (1986) introduce a dual-based procedure applied to the LP relaxation. They use a greedy heuristic followed by an improvement heuristic, which is similar to local exchange heuristics used in many combinatorial optimization problems.

In an airline crew scheduling application, Marsten and Shepardson (1981) combine Marsten's branching strategy with Etcheberry's bounding strategy. This hybrid algorithm is found to perform better than Marsten's original algorithm (Gerbracht 1978). However, SETPAR (Marsten, Muller and Killion 1979), a LP-based branch-and-bound computer code, is still the most popular means of solving the SPP among practitioners.

In this paper, we introduce an effective branch-and-bound algorithm for solving the SPP. New terms and notation are defined in section 1. We introduce a new bounding procedure in section 2 and the branching strategy in section 3. In section 4, we summarize the overall branch-and-bound algorithm and describe how the concept of variable elimination can be applied to SPP. In section 5, we compare the computational results for the new algorithm and SETPAR using actual data for a crew scheduling application.

1. Notation and Overview

$F = \{j \in J : \ x_j \text{ is fixed to be 1 } \}$

$I' = \{i \in I : \text{constraint } i \text{ is a free constraint } \}$

$J' = \{j \in J : \ \text{variable } j \text{ is a free variable } \}$

$I_j = \{i \in I' : a_{ij} = 1 \ \}$

$J_i = \{j \in J' : a_{ij} = 1 \ \}$

$LIST =$ list of candidate problems to be investigated

$U = (u_1, u_2, \ldots, u_m) =$ dual solution vector

$LB =$ lower bound of current subproblem

$SOL = \{j \in J' : \ x_j = 1 \ \}$

$UB =$ upper bound (objective value of incumbent)

$n_i = |SOL \cap J_i|$

$S_0 = \{i \in I' : \ n_i = 0 \ \}$

$S_1 = \{i \in I' : \ n_i = 1 \ \}$

$S_2 = \{i \in I' : \ n_i \geq 2 \ \}$

At each branching step of the branch-and-bound algorithm, certain variables are *fixed* to one or zero. When a variable is fixed to one, a new subproblem is created with fewer variables and constraints than the subproblem from which it branches. These remaining variables and constraints are referred to as the *free* variables and constraints, respectively.

The index set F represents the variables which are fixed to one for the current subproblem. Sets I' and J' contain the indices of the free constraints and variables, respectively, for the current subproblem. Subproblems corresponding to the unfathomed terminal nodes of the search tree are referred to as candidate problems. Subproblems created during branching are referred to as candidate subproblems. If a candidate subproblem is not fathomed, then it is stored in *LIST* and may be investigated at a future branching. To store a subproblem, only the values of F, J', and U need to be saved. (For computational efficiency, SOL is also stored.) When a new subproblem is created, the values of F, J', U and SOL are reset to their corresponding values in the parent candidate problem.

The bounding component is an iterative algorithm, which attempts to obtain a tight lower bound to the candidate subproblems. At each iteration, a feasible dual solution (U) is obtained for the LP relaxation of the subproblem. This dual solution is used to determine the corresponding lower bound (LB) and primal integer solution (SOL). If the integer solution is feasible to the subproblem, then $SOL \cup F$ represents a feasible solution to the original problem P. The current best feasible solution of P is referred to as the incumbent. The objective value of the incumbent provides the current upper bound, UB.

If $i \in I_j$ and $j \in SOL$, then we say constraint i is *covered* by variable j. The number of variables covering constraint i is indicated by n_i. Given an integer solution to a subproblem, the free constraints can be classified into three groups: 1) those which are *under-covered* (i.e., not covered by any variable); 2) those which are *tight* (i.e., covered by exactly one variable); and 3) those which are *over-covered* (i.e., covered by two or more variables). These three groups of constraints are represented by the index sets S_0, S_1 and S_2, respectively.

For notational simplicity, we also assume the constraints are indexed such that $|J_{i_1}| \leq |J_{i_2}| \; \forall \; i_1, i_2 \in I : i_1 < i_2$ at the beginning of the algorithm, where $J' = J$.

2. Bounding Strategy

Our goal is to develop a procedure which is fast, easy to apply, and provides a tight lower bound. The result is the multiplier adjustment method, MAM. When applied to a subproblem, MAM attempts to solve the dual of the LP relaxation:

$$\begin{aligned} \text{maximize} \quad & \sum_{i \in I'} u_i \\ \text{s.t.} \quad & \sum_{i \in I'} a_{ij} u_i \leq C_j \qquad j \in J' \qquad (2) \\ & u_i \quad \text{unrestricted} \quad i \in I \end{aligned}$$

The objective value of any feasible solution to this LP dual provides a lower bound to the subproblem.

Hereafter, we refer to the dual variables ($u_i s$) as multipliers to avoid confusion with the integer variables ($x_j s$) The constraints in (1) will be referred to as the primal constraints, while the constraints in (2) will be referred to as the dual constraints.

At each iteration, MAM increases certain multipliers to improve the lower bound, while other multipliers may be decreased to maintain dual feasibility. The criteria for selecting the multipliers will be described shortly. Before stating MAM formally, we need to introduce the following additional notation, which is relevant to the discussion of MAM only:

t = current iteration number

T = maximum number of iterations allowed

$$SLACK_j = C_j - \sum_{i \in I_j} u_i$$

$$SOL = \{j \in J' : SLACK_j = 0\}$$

$$LB = \sum_{i \in I'} u_i + \sum_{j \in F} C_j$$

MAM Algorithm

In the following statement of the algorithm, the variables $SOL, S_0, S_1, S_2, n_i s$ and $SLACK_j s$ always contain their current values. Explicit statements of their updates are omitted for simplicity. U is set initially to 0, and SOL, S_0, S_1, and S_2 are initialized accordingly.

When MAM is applied to a subproblem, the following steps are performed:

Step (0):

a) Set $t = 1$.

b) If $S_0 = S_2 = \emptyset, SOL$ is feasible.
Terminate MAM and go to step 5.

c) If $S_0 \neq \emptyset$, go to step 3.

Step (1): *Increase selected multipliers.*

a) Set $UP = \bigcup_{j \in SOL'} I_j \cap S_1$
where $SOL' = \{j \in SOL : \ I_j \cap S_1 \neq \emptyset, I_j \cap S_2 = \emptyset\}$.

b) If $UP = \emptyset$, terminate MAM.

c) Set $u_i = u_i + \min_{i \in J_i \setminus SOL} \{SLACK_j / |I_j \cap UP|\}, \quad i \in UP.$

Step (2): *Decrease selected multipliers in violated dual constraints.*

a) For each $j \in SOL'$ in turn:
If $SLACK_j < 0$,
set $u_j = u_i + \left(n_i / \sum_{i \in I_j \cap S_2} n_i \right) SLACK_j, \quad i \in I_j \cap S_2.$

b) If $S_0 = \emptyset$, go to step 4.

Step (3): *Ensure all primal constraints are covered.*

For each $i \in I'$ in turn:

If $i \in S_0$,

a) Set $u_i = u_i + \min_{j \in J_i}\{SLACK_j\}$.

b) Set $S_0 = S_0 \backslash I_q$ where $q = \text{argmin}_{j \in J_i}\{SLACK_j\}$.

c) If $S_0 = \emptyset$, go to step 4.

Step (4): *Check termination criteria.*

a) $LB \geq UB$, terminate MAM and fathom subproblem.

b) If $S_0 = S_2 = \emptyset$, go to step 5.

c) Set $t = t + 1$. If $t = T$, terminate MAM.

d) Go to step 1.

Step (5): *Check whether incumbent should be updated.*

a) If $LB < UB$, set $UB = LB$ and update incumbent.

b) Terminate MAM and fathom subproblem.

Further description of MAM

The basic idea of MAM is to adjust the multipliers to improve primal feasibility while remaining dual feasible. Whenever SOL is feasible (i.e., $S_0 = S_2 = \emptyset$), it becomes optimal to the subproblem because all complementary slackness conditions are satisfied. At each iteration, certain multipliers corresponding to over-covered primal constraints are decreased, while multipliers corresponding to under-covered primal constraints are increased.

The process of decreasing multipliers corresponding to over-covered primal constraints is accomplished indirectly by first increasing the multipliers in UP. (See step 1a.) The rationale for the selection is as follows. Each variable $j \in SOL'$ covers the tight primal constraints in $I_j \cap S_1$ and the over-covered primal constraints in $I_j \cap S_2$. When the multipliers in $I_j \cap S_1$ are increased, dual constraint j becomes violated. In step 2, this violation is corrected by decreasing the multipliers in $I_j \cap S_2$.

When the multipliers are increased in step 1c, only the dual constraints represented by SOL' become violated and need to be examined in step 2a. We refer to the operation described by step 2a as "coverage scaling". The more a primal constraint is over-covered, the more the corresponding multiplier is reduced. Each "scaling" operation causes a violated dual constraint to be satisfied exactly. Other methods of scaling are described by Chan (1987).

In step 3, certain multipliers in S_0 are increased to ensure that all primal constraints are covered. Each time a multiplier is increased in step 3b, at least one under-covered primal constraint becomes tight or over-covered. The amount of increase is chosen as the maximum increase permitted without violating any dual constraints. Hence, the dual solution is guaranteed to be feasible at the end of each iteration. In addition, SOL is a feasible solution to the corresponding set covering problem. This means that the objective value of this feasible solution also provides a valid upper bound for many applications such as vehicle routing. Unfortunately, this upper bound is not valid for our application.

There are five termination criteria. The first appears in step 0b. When a new subproblem is created, J' is reduced by the branching process. (See section 3.) As a result, the updated SOL may be feasible to the subproblem. This condition is checked in step 0b. The second criterion is found in step 1b and is satisfied when $UP = \emptyset$. This condition occurs either when the dual solution is optimal, or when the algorithm cycles. (A cycle-prevention method which is applicable to the coverage scaling technique is discussed in Chan 1987.) From our

experience, cycling occurs infrequently and only after many iterations (i.e., greater than 50), when LB is close to the optimal LP objective value. Furthermore, we have found that it is more than adequate to apply MAM for at most five iterations to each subproblem. Hence, cycling does not pose any serious problems, especially since MAM is being used as a bounding procedure.

The other three termination criteria are found in step 4. Step 4a is a global criterion which allows the subproblem to be fathomed. The feasibility criterion in step 4b is the same as that in step 0b. When a "duality gap" exists, this criterion cannot be satisfied. The maximum iterations criterion in step 4c is necessary because MAM is not an optimal procedure. In our implementation, we apply MAM for a maximum of ten iterations at the root node and a maximum of five iterations for all subproblems. After MAM terminates, if the subproblem has not been fathomed, then it is stored in *LIST*.

3. Branching Strategy

Our branching strategy was developed to complement MAM. It is similar to the one used by Martello and Toth (1981) for the generalized assignment problem. The branching strategy was motivated by the following observations. When MAM is applied, the variables which cover only tight primal constraints usually equal one in the optimal solution, even after only one iteration. Moreover, for each $i \in S_2, \bar{j}_i$ (the variable which covers constraint i in the optimal solution) is often contained in SOL, and is an element of $SOL_i = SOL \cap J_i$. If $\bar{j}_i$ is not in SOL_i, then it usually becomes an element of SOL when MAM is applied to the subproblem with SOL_i deleted from J'.

During the branching process, the algorithm first identifies the primal constraint having the smallest value of $|J_i|$ among the least over-covered constraints. (For computational efficiency, we use the cardinality at the beginning of the algorithm instead, as indicated in the formal statement below.) Indexing this constraint by p, the algorithm next creates $q+1$

candidate subproblems, where $q = |SOL_p|$. The first q candidate subproblems are created by fixing a unique variable in SOL_p to one. Fixing variable x_j to one reduces a candidate subproblem by $|I_j|$ constraints and $|\cup_{i \in I_j} J_i|$ variables. Candidate subproblem $q+1$ is created by fixing all variables in SOL_p to zero. A formal statement of the branching process is given below:

Step B1.

a) Let $p = \min_{i \in S_2} \{i : n_i = q\}$ where $q = \min_{i \in S_2} \{n_i\}$.

b) Let $SOL_p = \{j_1, j_2, j_3, \ldots, j_q\}$.

Step B2. For candidate subproblem $k \in \{1, 2, 3, \ldots, q\}$:

a) Set I', J', F, SOL and U to the corresponding values in the parent candidate problem.

b) Set $F = F \cup \{j_k\}, I' = I' \backslash I_{j_k}$ and $J' = J' \backslash (\cup_{i \in I_{j_k}} J_i)$.

For candidate subproblem $q + 1$:

a) Set I', J', F, SOL and U to the corresponding values in the parent candidate problem.

b) Set $J' = J' \backslash SOL_p$.

4. The Branch-and-Bound Algorithm

The new algorithm consists of the following five steps:

Step 0. $LIST = \{\}$.

Step 1. a) Set $I' = I, J' = J, F = \emptyset$, and $UB = \infty$.

b) i. Set $u_i = \min_{j \in J_i} \{C_j\}, \quad i \in I$.

ii. For each $j \in J$ in turn:

If $SLACK_j < 0$, set $u_i = u_i + SLACK_j / |I_j|, \qquad i \in I_j$.

c) Apply MAM.

d) If SOL is feasible, terminate.

Otherwise, go to step 3.

Step 2. a) If $LIST$ is empty, terminate.

Otherwise, retrieve candidate problem with smallest LB.

b) If $LB \geq UB$, terminate.

Step 3. Determine branching parameters. (See step B1 in section 3.)

Step 4. Create new candidate subproblems. (See step B2 in section 3.)

Step 5. a) For each candidate subproblem:

i. Apply feasibility and reduction tests.

ii. Apply MAM.

iii. If subproblem is not fathomed during i or ii,

then store it in $LIST$.

b) Go to step 2.

In step 1, all variables are initialized and MAM is applied to problem P. The multipliers are initialized at step 1b. Step 1bii is a variation of "scaling" (see section 2) which performs better than coverage scaling for the root node. If the primal integer solution obtained by MAM is feasible, it is also optimal and the algorithm terminates. Otherwise, the algorithm goes to step 3 to begin the branching process. The algorithm retrieves the next candidate problem using the "breadth-first" strategy. When the algorithm terminates, if an incumbent exists, then it is optimal; otherwise, P is infeasible .

For each new candidate subproblem, problem feasibility and reduction tests are applied. These tests basically check the cardinalities of the J_is. If $|J_i| = 0$ for any $i \in I'$, then the subproblem is infeasible and fathomed. If $|J_i| = 1$ for any $i \in I'$, then the "singleton" variable is fixed to one and I' and J' must be further reduced. In the latter case, the feasibility test

must be repeated. When these tests are completed, it is possible that SOL becomes feasible. If the subproblem is not fathomed, then MAM is applied to obtain a tighter lower bound.

Variable Elimination

To further improve the efficiency of the algorithm, we have also extended the concept of variable elimination to SPP. Variable elimination was formally presented by Sweeney and Murphy (1979) for the Multiple Choice Integer Program and implemented within a branch-and-bound framework for the multi-item scheduling problem (Sweeney and Murphy 1981). It has also been used in many reduction procedures for other combinatorial optimization problems.

To apply variable elimination, we use the following theorem: If $SLACK_j > UB - LB$, then $x_j = 0$ in any optimal solution. The proof is obvious from LP duality theory. If the multipliers are dual feasible, $SLACK_j$ (which equals $C_j - \sum_{i \in I'} a_{ij} u_i$) is the "reduced cost" and represents the minimum increase of the current objective value ($LB = \sum_{i \in I'} u_i$) if x_j is fixed to one. Hence, if $LB + SLACK_j > UB$, x_j must be zero in the optimal solution.

Although variable elimination can be applied to any subproblem, our experience indicates that it is sufficient to apply it to the root node only. After any iteration of MAM, all $j \in J' : SLACK_j > UB - LB$ can be deleted. If an initial feasible solution is not available, then the UB can be estimated by $LB + \gamma$. (See Chan 1987 for a detailed discussion on how γ may be determined using a statistical method, based on solutions of historical problems for a specific application.)

If the value of γ is chosen appropriately, a significant fraction of the variables can be eliminated without loss of optimality. This can result in a significant reduction in the size of the search tree and the computation time for the MAM. However, if γ is too large, only a few variables are eliminated and the effect on the algorithm is negligible; if γ is too small, the optimal solution may be eliminated. (See Chan 1987 for a discussion on how to deal

with the latter case.)

In our implementation, we apply variable elimination within MAM at the root node. At the end of iteration 5, we estimate UB to be $LB/0.9$ and reduce the problem accordingly. If the reduced problem contains fewer than 1000 columns, then we terminate MAM. At the end of iteration 10, we estimate UB to be $LB/0.95$ and reduce the problem further. These estimates were chosen conservatively enough so that the optimal solution is not eliminated in our test problems. The values can be selected to suit the particular application.

5. Computational Results and Discussion

Computational results are reported for a set of twenty crew scheduling problems provided by American Airlines, Inc. The size, density and optimal objective value of each problem are given in Table I. For all problems, the maximum value for $|I_j|$ is 17. These problems are available to any researcher interested in obtaining them.

Our computational experiment is performed on the Sequent Symmetry (S81) multiprocessor computer using only one of the twelve CPUs, each of which is rated at three million operations per second (or approximately four times faster than a VAX 750). We implemented our new algorithm in FORTRAN and refer to the computer code as MASP (Multiplier Adjustment for Set Partitioning) hereafter. We believe that the most accurate way to compare any algorithms is to test them simultaneously in the same environment. Hence, we compiled both SETPAR and MASP with the Balance FORTRAN Compiler, and solved the problems using each algorithm when the computer is otherwise completely idle. The results are shown in Table II. (For better readability, all real numbers are rounded to the nearest decimal point in Table II.)

Computational times for SETPAR and MASP are given in seconds in columns 2 and 4, respectively. These times exclude the time required to read the data and to initialize the variables. The input and initialization times are not reported; but on average, SETPAR takes

11% longer than MASP for these steps. For our application, the results indicate that MASP is significantly faster than SETPAR in all cases. The CPU time ratio (SETPAR/MASP) for each problem is given in column 5. On average, MASP is 16.6 times faster than SETPAR. This translates into a tremendous savings in computing time, since tens of thousands of similar problems are being solved on a daily basis at American Airlines.

The total number of pivots required by SETPAR for each problem is given in column 3. For MASP, the total number of nodes examined, the node at which the optimal solution is found, and the maximum depth of the search tree for each problem are shown in columns 6, 7, and 8, respectively. (Node 0 refers to the root node.) The total number of iterations executed for each problem is indicated in column 9. The results indicate that both the branching strategy and MAM are very effective.

Considering that the average number of constraints for the problem set is 159, the resultant search trees are fairly small. For all cases, only three subproblems are created during each branching step. Moreover, the optimal solutions are generally found very quickly, and the maximum depth of the search trees is only seven. Of the twenty problems, only problem 7 has a duality gap. Its LP optimal objective value $v(\bar{P})$ is 11170. The maximum depth of the search tree for problem 7 is 44 for SETPAR and only 4 for MASP. Unfortunately, we are not able to obtain any additional problems with a positive duality gap.

In addition, MAM is able to find the optimal solution within ten iterations for five of the problems. In fact, the optimal solution is found at iteration 3 for problem 14 and iteration 4 for problem 16. In general, the lower bounds obtained by MAM converge very quickly to $v(\bar{P})$. Although the lower bounds obtained by MAM can theoretically decrease, all but two of the iterations required to solve the twenty problems provide a lower bound with a positive improvement. The lower bounds from the remaining two iterations have zero improvement. (See Chan 1987 for a variation of MAM which is guaranteed to be monotonically nondecreasing.)

Columns 10 and 11 indicate how close the lower bound is to $v(\bar{P})$ (in terms of percentage of $v(\bar{P})$ after five and ten iterations, respectively. The dashes for problems 10, 14, 16, 19 and 20 indicate that the optimal solution are found before iteration 5 or 10, respectively. For problem 7, MAM terminates after five iterations at the root node because the reduced problem contains only 773 columns. On average, the lower bounds are within 2.4% of $v(\bar{P})$ after five iterations and within 1.5% after ten iterations.

Variable elimination was also very effective for our application. As revealed by columns 12 and 13, the average percentages of variables left are 28.1% and 11.0% at the end of iteration 5 and 10, respectively. (Again, the dashes indicate that MAM is terminated before reaching iteration 5 or 10.) The reductions in the sizes of the problems are significant in view of the observation that our estimates for UB are rather conservative. In our estimates, we assume the lower bounds are within 10% of $v(\bar{P})$ at the end of iteration 5 and within 5% at the end of iteration 10. However, columns 10 and 11 show that the lower bounds are at most 5% less than $v(\bar{P})$ at the end of iteration 5 and at most 3.6% at the end of iteration 10.

An alternate variable elimination approach is to apply it only when LB improves by at least a threshold amount during an iteration. For large-scale problems, applying variable elimination to all subproblems would result in improved performance. Chan (1987) describes several acceleration techniques which may reduce the computation time of MASP on large problems. When dealing with large-scale real-life applications, these strategies should be considered.

In this study, we have shown the practical effectiveness of variable elimination and the potential of specialized multiplier adjustment methods. In addition to providing tight lower bounds quickly, our new bounding procedure, MAM, is also easily applied. Unlike more general methods such as subgradient optimization, MAM does not have any parameters to fine tune.

Acknowledgment

We sincerely thank the three referees for their thorough and insightful reviews; the two associate editors for their thoughtful and constructive comments; American Airlines Decision Technologies for providing us with the problem set for this study; and XMP Software, Inc., for permitting us to compare SETPAR with MASP on the Sequent computer at Southern Methodist University.

References

Balas, E. and M.W. Padberg (1979), "Set Partitioning - A Survey," in N. Christofides, A. Mingozzi, P. Toth, and C. Sandi (editors), *Combinatorial Optimization*, Wiley, Chichester, England .

Balinski, M.L. and R.E. Quandt (1964), "On and Integer Program for a Delivery Problem," *Operations Research*, 12, 300-304.

Chan, T.J. (1987), "A Multiplier-Adjustment-Based Branch-and-Bound Algorithm for Solving the Set Partitioning Problem," Ph.D. Dissertation, The University of Michigan.

Christofides, N. and S. Korman (1973), "A Computational Survey of Methods for the Set Covering Problem," Report No. 73/2, Imperial College of Science and Technology, April 1973.

Day, R.H. (1965), "On Optimal Extracting from a Multiple File Data Storage System: An Application of Integer Programming," *Operations Research*, 13, 3, 489-494.

Etcheberry, J. (1977), "The Set Covering Problem: A New Implicit Enumeration Algorithm," *Operations Research*, 25, 760-772.

Fisher, M.L. (1981), "The Lagrangian Relaxation Method for Solving Integer Programming Problems", *Management Science*, 27, 1, 1-18.

Fisher, M.L. and P. Kedia (1986), "A Dual Algorithm for Large Scale Set Partitioning," Purdue University, Krannert Graduate School of Management, Working Paper No. 894.

Garfinkel, R.S. and G.L. Nemhauser (1969), "The Set Partitioning Problem: Set Covering with Equality Constraints," *Operations Research*, 17, 848-856.

Garfinkel, R.S. and G.L. Nemhauser (1970), "Optimal Political Districting by Implicit Enumeration Techniques," *Management Science*, 16, B495-B508.

Garfinkel, R.S. and G.L. Nemhauser (1972), "Optimal Set Covering: A Survey," in A. Geoffrion (editor), *Perspectives on Optimization*, Addison-Wesley, Reading, Massachusetts.

Geoffrion, A.M. (1974), "Lagrangian Relaxation for Integer Programming," *Mathematical Programming Study*, 2, 82-114.

Gerbracht, R. (1978), "A New Algorithm for Very Large Crew Pairing Problems," 18th AGIFORS Symposium, Vancouver, British Columbia, Canada, September, 1978.

Lu, Ming-Te (1970), "A Computerized Airline Crew Scheduling System," Ph.D. Thesis, University of Minnesota.

Marsten, R.E. (1974), "An Algorithm for Large Set Partitioning Problems," *Management Science*, 20, 779-787.

Marsten, R.E., M.R. Muller and C.L. Killion (1979), "Crew Planning at Flying Tiger: A Successful Application of Integer Programming," *Management Science*, 25, 12, 1175-1183.

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

Martello S. and P. Toth (1981), "An Algorithm for the Generalized Assignment Problem," in J.P. Brans (editor), *Operational Research '81*, North-Holland, Amsterdam, 589-603.

Michaud, P. (1972), "Exact Implicit Enumeration Method for Solving the Set Partitioning Problem," *IBM Journal of Research and Development*, 16, 573-578.

Pierce, J.F. (1968), "Application of Combinatorial Programming to a Class of All-Zero-One Integer Programming Problems," *Management Science*, 14, 191-209.

Pierce, J.F. and J.S. Lasky (1973), "Improved Combinational Programming Algorithms for a Class of All Zero-One Integer Programming Problems," *Management Science*, 19, 528-543.

Revelle, C., D. Marks and J.C. Liebman (1970), "An Analysis of Private and Public Sector Location Models," *Management Science*, 16, 12, 692-707.

Root, J.G. (1964), "An Application of Symbolic Logic to a Selection Problem," *Operations Research*, 12, 4, 519-526.

Salkin, H.M. (1975), *Integer Programming*, Addison-Wesley, Reading, Massachusetts.

Sandi, C. (1979), "Subgradient Optimization", in N. Christofides, A. Mingozzi, P. Toth and C. Sandi (editors), *Combinatorial Optimization*, Wiley, Chichester, England.

Steinman, H. and R. Schwinn (1969), "Computational Experience with a Zero-One Programming Problem," *Operations Research*, 17, 5, 917-920.

Sweeney, D.J. and R.A. Murphy (1979), "A Method of Decomposition for Integer Programs," *Operations Research*, 27, 1128-1141.

Sweeney, D.J. and R.A. Murphy (1981), "Branch and Bound Methods for Multi-item Scheduling," *Operations Research*, 29, 853-864.

Thuve, H. (1981), "Frequency Planning as a Set Partitioning Problem," *European Journal of Operational Research*, 6, 29-37.

Valenta, J.R. (1969), "Capital Equipment Decisions: A Model for Optimal Systems Interfacing," M.S. Thesis, M.I.T., June 1969.

TABLE I. Description of Data Set.

Problem #	# of rows (m)	# of columns (n)	density (%)	v(P)
1	162	5218	5.47	10348
2	167	5014	5.47	10976
3	162	9612	5.79	9965
4	155	6803	5.87	10233
5	157	4691	6.26	9052
6	156	7496	5.49	10104
7	124	6313	6.26	11250
8	150	4292	5.80	9484
9	166	7005	5.70	11274
10	157	7199	5.53	8540
11	173	4706	5.69	11156
12	171	8406	5.18	11061
13	173	3874	5.57	11247
14	150	7107	5.56	9681
15	158	4127	5.44	12119
16	171	7256	5.31	12528
17	165	6786	5.41	10129
18	153	8223	5.89	9480
19	159	7957	5.37	10739
20	159	9238	5.59	8821

Table II. Computational Results

Problem #	SETPAR (sec.)	# pivots	MASP (sec.)	CPU Time Ratio	# nodes	opt. node	max. depth	# iter.	% of v(P̄) @ 5 iter.	% of v(P̄) @ 10 iter.	(|J'|/|J|) (in %) @ 5 iter.	(|J'|/|J|) (in %) @ 10 iter.
1	269.2	210	15.0	18.0	24	5	4	55	97.4	98.3	20.5	7.7
2	328.6	256	22.2	14.8	15	7	4	46	95.7	97.7	49.9	19.8
3	272.2	120	19.3	14.1	27	11	5	63	96.9	98.1	11.3	3.9
4	233.2	152	24.4	9.6	24	11	6	58	96.4	96.4	35.5	13.1
5	97.8	92	11.1	8.9	3	1	1	12	99.0	99.7	32.5	9.9
6	276.2	163	17.1	16.1	30	12	5	82	95.2	96.4	16.4	5.7
7	325.7	207	15.4	21.1	18	13	4	69	96.4	—	12.2	—
8	136.8	142	10.5	13.0	9	4	2	24	96.8	97.4	25.7	9.3
9	597.5	328	47.4	12.6	54	31	6	140	95.6	96.8	44.7	16.1
10	163.3	99	11.2	14.6	0	0	0	7	100.0	—	15.3	—
11	223.8	177	14.3	15.6	3	1	1	14	98.8	99.8	61.4	11.4
12	252.5	149	21.5	11.7	39	19	5	97	97.9	98.4	28.2	8.3
13	266.2	268	18.4	14.5	30	18	6	76	97.0	98.4	44.3	18.2
14	67.8	54	4.4	15.6	0	0	0	3	—	—	—	—
15	221.9	233	18.0	12.4	30	24	5	75	96.4	97.0	40.9	16.2
16	261.2	155	8.3	31.4	0	0	0	4	—	—	—	—
17	513.8	304	30.7	16.8	57	39	7	139	95.0	96.4	29.3	11.3
18	526.0	261	17.0	31.0	9	7	3	21	97.9	99.1	14.1	3.4
19	284.8	162	10.9	26.0	0	0	0	6	99.8	—	14.7	—
20	224.8	110	15.2	14.8	0	0	0	8	99.6	—	19.7	—