1982

A class of location, distribution and scheduling problems: modeling and solution methods

Egon Balas
Carnegie Mellon University


Follow this and additional works at: http://repository.cmu.edu/tepper
A CLASS OF LOCATION, DISTRIBUTION AND SCHEDULING PROBLEMS: MODELING AND SOLUTION METHODS

by

E. Balas

Dscambar, 1982

DKC-70-14-32
A CLASS OF LOCATION, DISTRIBUTION
AND SCHEDULING PROBLEMS:
MODELING AND SOLUTION METHODS

by
Egon Balas

Graduate School of Industrial Administration
Carnegie-Mellon University
Pittsburgh, PA., U.S.A.

Paper presented at the Chinese-American Symposium
on Systems Analysis and Engineering, Xian, April 1981

The research underlying this paper was supported by Grant ECS-7902506
of the National Science Foundation, and Contract N00014-75-C-0621 NR 047-048
with the U.S. Office of Naval Research.
A CLASS OF LOCATION, DISTRIBUTION
AND SCHEDULING PROBLEMS:
MODELING AND SOLUTION METHODS
by
Egon Balas
Carnegie-Mellon University
Pittsburgh, Pa., U.S.A.

1. Introduction

A large variety of location, distribution, scheduling and other problems can be formulated as variants of a mathematical model known as the covering or set covering problem. A partial list of real world problems to which this approach has been successfully applied includes the following instances (see the Bibliography on Applications at the end of the paper):

- site selection and facility location-allocation problems
- location of emergency service facilities (fire stations, hospitals etc.)
- choice of size and location of drilling platforms in offshore oilfields
- vehicle routing: truck dispatching problem, tanker fleet and airline fleet scheduling
- crew scheduling for airlines, bus companies, railways
- the minimum test set (diagnostic) problem (in industry, medicine, experimental design)
- switching circuit design (electrical engineering)
- distribution of broadcasting frequencies among radio or TV stations
- information retrieval (from computer files)
- assembly line balancing
- stock cutting
- various capital investment decisions
The (weighted) set covering problem is

\[(SC) \quad \min \{cx | Ax \geq e, x \in \{0, 1\}, \text{jeN} \}\]

where \(A \in \mathbb{R}^{m \times n}\), \(e \in \mathbb{R}^m\), and \(c \in \mathbb{R}^n\). Its name comes from the following interpretation:

if the rows of \(A\) are associated with the elements of the set \(M\), and each column \(a_j\) of \(A\) with the subset \(M_j\) of those \(i \in M\) such that \(a_{ij} \geq 1\), then \((SC)\) is the problem of finding a minimum-weight family of subsets \(M_j, \text{jeN}\), whose union is \(M\), i.e., which "cover" \(M\), each subset \(M_j\) being weighted with \(c_j\).

The special case when \(c_j = 1, \text{jeN}\), is called the simple (unweighted) covering problem.

Another interpretation of \((SC)\) is as follows. Let \(G = (V, E)\) be a bipartite graph, i.e., a graph whose vertex set \(V\) can be partitioned into two subsets, \(V_1\) and \(V_2\) such that \(EC \cap X V_j, i.e., every edge \((i, j) \in E\) is of the form \(i \in V_1, j \in V_2\). We say that a vertex \(j \in V_2\) covers a vertex \(i \in V_1\) if \((i, j) \in E\). If vertex \(j\) has weight \(c_j, \text{jeV_2}\), then \((SC)\) is the problem of covering the vertices of \(V_1\) with a minimum-weight subset of the vertices of \(V_2\), with \(M = V_1, H = V_2, \text{and for jeV_2, M_j = \{icVj|d, j \in E\}}\).

A close relative of \((SC)\) is (weighted) set partitioning (equality-constrained set covering) problem

\[(SP) \quad \min \{cx + ey | Ax - y = e, y \geq 0, x_j \in \{0, 1\}, \text{jeN} \}\]

where \(A, e\), and \(c\) are as before. \((SP)\) can be brought to the form \((SC)\) by writing

\[\min \{cx + ey | Ax - y = e, y \geq 0, x_j \in \{0, 1\}, \text{jeN} \}\]

and then, using \(y \geq Ax - e\),
mint-em + c'x ≥ e, x_j ≥ 0 or 1, \_j \in N

with c' = c + SeA. For sufficiently large 9 (for instance, 9 ≥ c), this problem has the same set of optimal solutions as (SP) whenever the latter is feasible. Both set covering and set partitioning are used in formulating the problems listed above, and sometimes a mixed covering-partitioning problem arises. Also, in many real-world situations a few extra constraints may be needed, which require appropriate modifications of the solution methods.

In the next section we discuss the modeling potential of the set covering approach, illustrating the problem formulation techniques on several important classes of real-world problems. In section 3 we describe a class of algorithms for solving set covering problems, based on cutting planes, heuristics and subgradient optimization. Finally, as an Appendix we provide two bibliographies, one on theory and algorithms, the second one on applications (classified by area) of the set covering and set partitioning models.

2. Modeling Techniques

The high versatility of the model under discussion stems from the fact that all the real world problems listed above, and a great variety of other problems, can be formulated as follows. Given

(i) a finite set M;

(ii) a system of constraints on the elements of M, defining a family F of "acceptable" subsets of M; and

(iii) a function on M defining a cost for every member of the family F; find a minimum-cost collection of members of F which, cover M, i.e., whose union is M.

The applicability of the set covering model to problems amenable to this formulation is based on the simple but important observation that in
most cases problems of this form can be solved with a satisfactory degree of precision by the following two-stage approximation procedure.

Stage 1. Using (ii) and (iii), generate explicitly a subfamily $FCF, F^* = \{F_j \mid j \in \mathbb{N}, c_j \}$, with associated costs $c_j, j \in \mathbb{N}$, for which the probability that $F$ contains an optimal solution is sufficiently large.

Stage 2. Replace the objective function (iii) by $ex$, and the system of constraints (ii) by $Ax \geq e, x_j = 0 \text{ or } 1, j \in \mathbb{N}$, where the columns of $A$ correspond to the elements of $F$ (i.e., $a_{ij} = 1$ if $i \in M$, and $a_{ij} = 0$ otherwise), and solve the resulting set covering problem.

In the following we illustrate this modeling procedure on several examples.

**Offshore Drilling Platforms.** To start the exploration of an offshore oilfield, after fixing the location of the wells to be drilled on the basis of geological data, one has to choose the appropriate size and location of the platforms to be used for the drilling (and later for the exploitation) of the wells. Drilling platforms vary immensely in size and cost. A platform may handle just one or two wells, or as many as 30-40 wells; it can be just a few yards high or as high as the Empire State Building; and it may cost anywhere between a few hundred thousand dollars and 100 million dollars. The best platform/well configuration depends on the distances between the wells, the shape of the seabed, the depth of the water, the depth to which one has to drill to reach the oil, etc. These factors define both the system of constraints on the size and location of the platforms, and the cost function. Rather than trying to write down explicitly these complicated and highly non-linear functions and constraints, one can proceed as follows. Given the location of $m$ wells expressed as a set of coordinate pairs in 2-space,
heuristic rules are defined and put into a computer program, for grouping together wells that might lend themselves to being drilled from a single platform. For each such group of wells, say $M_j$, the cost of the corresponding platform, connecting pipes and other necessary equipment, is estimated and expressed as a single number $c_j$. The wells are grouped in many different ways, and each group $M_j$ corresponds to a candidate platform, i.e., one that may or may not be built. Each candidate platform $j$ is associated with a cost $c_j$ and a column $a_j$ of a 0-1 matrix to be used in a set covering problem, namely $a_{ij} = 1$ if well $i$ is included in the group of candidate platform $i$, $a_{ij} = 0$ otherwise. Solving the set covering (or set partitioning) problem formulated this way will then select an optimal combination of drilling platforms to be built. Although the set covering problem can usually (i.e., up to 1000-2000 columns) be solved to optimality, the solution obtained is not necessarily optimal for the real problem, since some combination of platforms and wells may have been omitted in the Stage 1 procedure of generating platform candidates. But if a sufficiently reliable procedure is used in Stage 1, i.e., one that does not omit any promising candidate, then the optimal solution of the Stage 2 problem should be pretty close to the optimum of the real problem.

Location and Number of Emergency Service Facilities. In deciding upon the number and location of hospitals, fire stations or other emergency service facilities dedicated to fill the needs of a certain area, one good criterion to use is that each point in the area be reachable from at least one facility in no more than some predetermined time limit $t$. If the points of the area (population centers, villages, quarters of a city, etc.) are represented as vertices of a graph, and the candidates for the location of a service facility
as a subset of those vertices, and if the edges of the graph have lengths associated with them that reflect the time needed to reach an end-vertex of the edge from the other, then Stage 1 consists of determining, for each candidate location \( j \), the set \( M_j \) of vertices reachable from \( j \) within the time limit \( t \). Stage 2 will then solve a set covering problem whose coefficient matrix \( A \) has a column \( a_{ij} \) for each candidate location for a service facility, with \( i \cdot 1 \) if point \( i \) can be reached from candidate facility \( j \) in no more time than \( t \), \( a_{ij} = 0 \) otherwise. The solution gives both the number and location of the facilities needed. Obviously, the result is a function of the time limit \( t \), and solving the problem for the relevant range of values of \( t \) also provides information about the cost of improving the emergency services, or the savings achievable through a relaxation of the service requirements.

**Crew Scheduling.** Airlines, bus companies, railways are facing the problem of scheduling their crews for the flights or trips to be provided in a given time period. To take the case of an airline, crews based in various cities have to be scheduled to man the flights of a given time period, say a week, so as to make the best use of their time. The conditions that have to be met are those of avoiding conflicts in the schedule, providing for reasonable breaks between flights, keeping a limit on the number of hours flown at night as well as a balance between the various crews in this respect, having each crew spend time periodically at its home base, etc. All these and other considerations that have to be taken into account give rise to a highly complex cost function and constraints, hard even to formulate. Instead of trying to do so, however, one usually sets up this problem as a
set covering problem without ever writing down the constraints in functional form. In Stage 1, tentative routes are explicitly generated for each crew, that take into account the requirements, i.e., exclude conflicts, provide for breaks, etc. This is done by many airlines through heuristic programs that examine explicitly a very large number of possible schedules for each crew and retain those among them that are not obviously bad. Each such candidate schedule for a crew generates a column $a_j$ of the 0-1 matrix $A$, where $a_{ij} = 1$ if schedule $j$ (for a given crew) includes flight-leg $i$, $a_{ij} = 0$ otherwise, and a cost $c_j$ which is a synthetic expression of the extent to which schedule $j$ meets (or violates) the above listed requirements. Here a flight-leg is a flight leaving a given city at a given time and reaching another city at a given time. Solving the set covering problem (or set partitioning problem; depending on conditions specific to each airline), provides a schedule for each crew that covers all the flight legs to be covered during the period in question, while minimizing the total cost of the overall schedule (in terms of inconvenience, or sometimes actual money).

The Minimum Test Set Problem. The following problem arises in environments as diverse as product classification and quality control in industry, medical diagnostics, design of experiments, etc. Given a set of objects $Q = \{1, \ldots, q\}$, and a set of attributes (properties) $P = \{1, \ldots, p\}$ of some of these objects, find a minimal set $S$ of properties to enable one to distinguish between the objects; in other words, find a set $S \subseteq P$ such that for every pair of objects $i, j \in Q$, there exists at least one property $k \in S$ such that object $i$ has property $k$ and object $j$ does not have it, or vice versa. In an industrial context, the properties in question are characteristics that make it possible
to classify a product (as belonging or not belonging to a certain class, being or not being admissible, etc.) on the basis of a minimum number of measurements or tests. In a medical context, one is looking for a minimum number of tests that one has to perform in order to safely diagnose a disease, or, which is the same thing, be able to distinguish between diseases showing similar symptoms. Other applications abound, and the problem can also be formulated somewhat more generally by assigning weights to the properties and asking for a minimum-weight (rather than a minimum-cardinality) set of properties to satisfy the required condition. Here the interpretation of the weights may be the cost of the tests or measurements (in industry), the risk involved in the tests (in medicine), etc.

The formulation of this problem as a set covering problem is not so straightforward as in the other examples discussed above. Let $D = (d_{ij})$ be the incidence matrix of objects versus properties, i.e., let $D$ have a row for every object and a column for every property, with $d_{ij} = 1$ if object $i$ has property $j$, $d_{ij} = 0$ otherwise. Then our problem can be stated as that of finding a minimum number of columns (or, if the properties are weighted, a minimum-weight subset of the column set) such that the submatrix of $D$ consisting of these columns has no pair of rows that are componentwise equal; in other words, such that for every pair $i, k$ of rows (objects), the submatrix in question contains at least one column (property) $j$ such that $d_{ij} = 1$ and $d_{kj} = 0$, or $d_{ij} = 0$ and $d_{kj} = 1$.

Now define a new $0$-$1$ matrix $A = (a_{ij})$ with $n = p$ columns, one for every column of $D$, and $m = \frac{1}{2} q(q-1)$ rows, one for every distinct pair of rows of $D$ and such that, if row $k$ of $A$ corresponds to the pair of rows $i_k, j_k$ of $D$, then
With this definition, the minimini test set problem can be formulated as the set covering problem

\[
\min \{ cx_j | Ax \geq e, x_j \neq 0 \text{ or } 1, j \in N \},
\]

where e and N are as before, while c_j is the weight assigned to property j (i.e., if we are solving the unweighted problem, c_j = 1, \( \forall j \in N \)).

3. **Solution Methods**

In this section we discuss a class of algorithms for solving set covering problems, based on cutting planes from conditional bounds [1, 2]. Several versions of such an algorithm were implemented jointly with A. Ho [3], and extensively tested on randomly generated and real world problems, with the conclusion that this algorithm is a reliable and efficient tool for solving large, sparse set covering problems of the kind that frequently occurs in practice. With a time limit of 10 minutes on a DEC 20/50, we have solved all but one of a set of 50 randomly generated set covering problems with up to 200 constraints, 2000 variables and 8000 nonzero matrix entries (here "solving" means finding an optimal solution and proving its optimality), never generating a branch and bound tree with more than 50 nodes. For problems that are too large to be solved within a reasonable time limit, the procedure usually finds good feasible solutions, with a bound on the distance from the optimum. (For the one unsolved problem, this bound was 2.3%).
We consider the set covering problem (SC) introduced in section 1, and denote

\[ M_j \triangleq C_i \{ a_{ij} \leq 1 \}, j \in N; \quad H_x \triangleq C_j \{ a_j^\top x \leq 1 \}, i \in M. \]

We also use the pair of dual linear programs

(L) \[ \min \{ c^\top x | A x \geq e, x \geq 0 \} \]
and

(D) \[ \max \{ u^\top e | u A \leq c, u \geq 0 \} \]
associated with (SC).

A 0-1 vector \( x \) satisfying \( A x \geq e \) is called a \textit{cover}, and its \textit{support} \( S(x) = \{ j : N \mid x_j = 1 \} \). A cover whose support is nonredundant is \textit{prime}. For a cover \( x \), we denote \( T(x) = \{ i : M \mid a_i^\top x < 1 \} \), where \( a_i^\top \) is the \( i \)-th row of \( A \).

The theory underlying the family of cutting planes from conditional bounds can be summarized as follows (for proofs of these statements, interpretation of the cuts in terms of conditional bounds, and further elaboration on their properties, see [2]).

Let \( z^- \) be some upper bound on the value of (SC), and let \( u \) be any feasible solution to (D), with \( s \cdot c - u A \), such that the condition

(1) \[ \sum_{j \in S} s_j \geq z^- - u e \]

is satisfied for some \( S \in N \). Let \( S = \{ j(D, \ldots, j(p)) \} \), and let \( Q_1, \ldots, Q_p \), be any collection of subsets of \( N \) satisfying

(2) \[ \sum_{j \in Q_i} s_j \leq 1, j \in N. \]

Then every cover \( z \) such that \( e x < z^- \) satisfies the disjunction

(3) \[ \sum_{i=1}^p \left( \sum_{j \in Q_i} s_j \right) \leq z^- - u e. \]
Further, for any choice of indices \( h(i) \in M, i = 1, \ldots, p \), the disjunction (3) implies the inequality

\[
E r > 1
\]

where

\[
W = \bigcup_{i=1}^{p} (M_i \setminus j_i).
\]

Finally, if \( j(i) \in Q, i = 1, \ldots, p \), and if \( \bar{x} \) is a cover such that \( S \cap S(\bar{x}) \), and \( h(i) \in T(\bar{x}) \setminus j_i, i = 1, \ldots, p \), then the inequality (4) cuts off \( \bar{x} \) and defines a facet of

\[
P = \text{conv}\{x \in \mathbb{R}^n[\text{Ax} \geq b, Z x \geq 1, x \geq 0, x \text{ integer, jctf}]\},
\]

where \( \text{conv} V \) means the convex hull of the set \( V \).

Using the above results, one can generate a sequence of cutting planes that are all distinct from each other, by generating a sequence of covers \( x \) and feasible solutions \( u \) to (D). The covers \( x \) provide upper bounds, while the vectors \( u \) provide lower bounds on the value of (SC). Since every inequality that is generated cuts off a cover satisfying all previously generated inequalities, and the number of distinct covers is finite, the procedure ends in a finite number of iterations, with an optimal cover at hand.

The algorithm alternates between two sets of heuristics, one of which finds a "good" prime cover \( z \) for the current problem and a (possibly improved) upper bound, while the other generates a feasible solution to (D) satisfying condition (1) for \( S \cap S(x) \), and from it a cutting plane (4) that cuts off \( x \), as well as a (possibly improved) lower bound. Whenever a disjunction (3) is obtained with \( p = 1 \),
all the variables indexed by \( Q_L \) are set to 0. The second set of heuristics is periodically supplemented by subgradient optimization to obtain sharper lower bounds.

Though this procedure in itself is guaranteed to find an optimal cover in a finite number of iterations, for large problems this may take too many cuts. Therefore, as soon as the rate of improvement in the bounds decreases beyond a certain value, the algorithm branches.

A schematic flowchart of the algorithm is shown in Fig. 1. PRIMAL designates the set of heuristics used for finding prime covers, DUAL the heuristics used for finding feasible dual solutions. TEST is the routine for fixing variables at 0. CUT generates a cutting plane violated by the current cover. SGBAD uses subgradient optimization in an attempt to find an improved dual solution and lower bound. BRANCH is the branching routine which breaks up the current problem into a number of subproblems, while SELECT chooses a new subproblem to be processed.

The four decision boxes of the flowchart can be described as follows. Let \( z_U \) and \( z_L \) be the current upper bound and lower bound, respectively, on the value of (SC).

1. If \( z_T > z_{TT} \), the current subproblem is fathomed (1.1). If \( z_T < z_{TT} \) and some variable belonging to the last prime cover has been fixed at 0, a new cover has to be found (1.2). Otherwise, a cut is generated (1.3).

2. After adding a cut, the algorithm returns to PRIMAL (2.1) unless the iteration counter is a multiple of some constant \( \alpha \), in which case (2.2) it uses subgradient optimization in an attempt to improve upon \( z_T \). Based on some experimentation, the value of \( \alpha \) is chosen such that \(|M|/10 < 3 < (|M|/20)\).
Fig. 1
3. If $z_L \geq z_U$, the current subproblem is fathomed (3.1). If $z_L < z_U$ but the gap $z_U - z_L$ has decreased by at least $\varepsilon > 0$ during the last $3$ iterations for some constant $\beta$, we continue the iterative process (3.2). Otherwise, we branch (3.3). Again, based on some experimentation, we use $\varepsilon = 0.5$ and $\beta = 4\alpha$, with $\alpha$ as defined in 2.

4. If there are no active subproblems, the algorithm stops: the cover associated with $z_U$ is optimal (4.1). Otherwise, it applies the iterative procedure to the selected subproblem (4.2).

Next we briefly discuss the various ingredients of the algorithm and their role in making the procedure efficient.

**Primal heuristics.** Most of the procedures we use to generate prime covers are of the "greedy" type, in that they construct a cover by a sequence of steps, each of which consists of the selection of a variable $x_j$ that minimizes a certain function of the coefficients of $x_j$. They differ in the function $f$ used to evaluate the variables. If $k_j$ denotes the number of positive coefficients of $x_j$ in those rows of the current constraint set not yet covered, the general form of the evaluation function is $f(c_j, k_j)$.

Since it is computationally cheaper to consider only a subset of variables at a time and since every row must be covered anyhow, we restrict the choice at each step to those variables having a positive coefficient in some specified row $i \in M$, where $M$ indexes the rows. Denoting by $R$ the set of rows not yet covered and by $S$ the support of the cover to be constructed, the basic procedure that we use can be stated as follows.
Step 0. Set $R \cdot M$, $S \cdot 0$, $t \cdot 1$, and go to 1.

Step 1. If $R = 0$, go to 2. Otherwise let $k_j = \min_i |M_{ij}|$, choose $i^* \in R$, and choose $j(t)$ such that
\[
\min_{j \in I^*} \sum_j f(c_{kj}(t))
\]
Set $R \leftarrow R \setminus M_{ij}$, $S \leftarrow S \cup \{j(t)\}$, $t \leftarrow t + 1$, and go to 1.

Step 2. Consider the elements $i \in S$ in order, and if $S \setminus \{i\}$ is the support of a cover, set $S \leftarrow S \setminus \{i\}$. When all $i \in S$ have been considered, $S$ defines a prime cover.

As to the choice of $i^*$ in Step 1, we order the rows of the initial coefficient matrix once and for all according to decreasing $N_\cdot$, and then always choose $i^*$ as the last element of the ordered set $R$. Since the cuts generated in the procedure also tend to have a decreasing number of 1's, i.e. later cuts tend to have fewer positive coefficients than earlier cuts, this rule approximates the criterion of always choosing a row with a minimum number of positive coefficients.

If the set $N_\cdot$ in step 1 is replaced by $N$ and step 2 is removed, i.e. if the choice of columns is not restricted every time to a particular row and the procedure is allowed to stop whenever a cover is obtained, whether prime or not, then the above procedure is the greedy heuristic shown by Chvatal [4] to have the following property: if $z_\text{neu}$ is the value of (SC) and $z_\text{opt}$ the value of the solution found by the heuristic, then
\[
\frac{z_\text{neu}}{z_\text{opt}} < \frac{\max_{j \in N} |H_j|}{d^*}
\]
where
\[
d^* = \max_{j \in N} |H_j|.
\]
and this bound is best possible. From a practical standpoint, this bound is of course very poor and it was shown by Ho [6] that there is no better bound for any function \( f \) used in the above procedure. Ho's proof of this result relies on the construction of examples for which the worst case bound is attained, and different families of functions \( f \) require different examples. This suggests as a practical remedy against the poor worst case performance of the heuristic, the intermittent use of several functions \( f \) rather than a single one. This idea was implemented and tested with reasonably good results. The following five functions were considered: (i) \( c_j \); (ii) \( c_j/k_j \); (iii) \( c_j/\log_2 k_j \); (iv) \( c_j/k_j \log_2 k_j \); (v) \( c_j/k_j 2^{n_k} \). In cases (iii) and (iv), \( \log_2 k_j \) is to be replaced by 1 when \( k_j = 1 \); and in case (v), \( 2^{n_k} k_j \) is to be replaced by 1 when \( k_j = 1 \) or 2.

The five functions were tested on a set of randomly generated problems, with the result that mixing them intermittently rather than using any one of them by itself improves the quality of the solution considerably.

A different primal heuristic, that we use every time the subgradient method is applied to obtain an improved lower bound, is based on the reduced costs

\[
s_j = c_j - u_a_j
\]

produced by the subgradient method. This procedure sets \( x_j = 1 \) if \( s_j = 0 \), \( x_j = 0 \) otherwise. The resulting vector \( x \) either is a cover, or else if row \( i \) is uncovered, then \( s_j > 0 \) for all \( j \in \mathbb{N}_i \), and \( u_i \) can be increased to

\[
u_i + \min_{j \in \mathbb{N}_i} s_j.
\]

This creates at least one new reduced cost \( s_k \) equal to 0, and for each such \( k \) we set \( x_k = 1 \). We proceed this way until every row is covered, after which we apply step 2 of the first heuristic to make the cover prime. This second heuristic, though considerably more expensive than the first one (because of the computational effort involved in the subgradient method), consistently outperformed the first heuristic.
Dual heuristics and subgradient optimization. The purpose of these procedures is to find, at a low computational cost, "good" feasible solutions to (D), hence "good" lower bounds are the value of (SC). The heuristics used are again of the greedy type, in that they construct a feasible solution to (D) by a sequence of steps, each of which consists of selecting a row $i^*$ with a small number of positive coefficients, and assigning to $u_i$ the maximum value that can be assigned without violating the constraints or changing some earlier value assignment. In choosing $i^*$, priority is given to $i \in T(x) = \{i \in M | a_i x > 1\}$, where $x$ is the current cover. This is done in order to obtain a reduced cost vector $s = c - uA$ that satisfies condition (1) for $S = S(x)$, since it is known (see [2]) that this is the case if $u$ satisfies $u(Ax-e) = 0$.

While this heuristic (used with minor variations depending on the situation) provides reasonably good solutions to (D) at a very low computational cost, a sharper lower bound could of course be obtained by solving (D) to optimality. After sufficient cuts have been added, the value $z_{\text{L}}$ of (D) may exceed $z_{\text{T}}$, thus bringing the procedure to an end. However, the computational effort involved in repeatedly solving (D) by the simplex method is considerable, and increases about quadratically with the number of cuts added to the constraint set of (SC). On the other hand, one can use subgradient optimization to find a near-optimal solution to (D) at a computational cost that increases only linearly with the number of cuts added. This is what we are doing periodically in order to generate lower bounds stronger than those provided by the heuristic.

Our experience with the subgradient method has been that although it is more expensive than the dual heuristics often by 1 or 2 orders of magnitude, it nevertheless pays off if used sparingly, in combination with the heuristics.
On the one hand, it usually improves the lower bound; on the other, it produces a set of reduced costs that can be used to obtain improved covers, as explained in connection with the primal heuristics. At the same time, subgradient optimization cannot replace the dual heuristics, since it usually provides fractional solutions to (D) and such solutions tend to yield weaker cuts than the integer solutions obtained by the heuristic.

**Fixing variables and generating cuts.** Every time a new solution $u$ to (D) is obtained either by the heuristic or by subgradient optimization, the algorithm searches for variables $x_j$ such that $s_j > z_j - u_x$, and fixes them at 0. This feature comes into play from early on in the procedure, and in the randomly generated test problems that we solved, the number of variables left by the time the first branching occurred, was almost always close to the initial number $m$ of constraints.

To generate cuts, the algorithm uses the results stated at the beginning of this section. In order to obtain a cut (4) as strong as possible, i.e. with $|w|$ as small as possible, the construction of the sets $Q_i$ and the choice of the indices $h(i)eM$ is done sequentially, so that at each step the set $N_i ..AQ_i$ is minimized. The cut generating subroutine is as follows. Let $x$ be a cover with $S(x)$ and $T(x)$ defined as before, let $u$ be a feasible solution to (DX with $s * c-u_A$, and assume that $s$ satisfies (1) for $S > S(x)$.

**Step 0.** Set $W = 0$, $S = \{j|S(x)|s_j > 0\}$, $y = u_x$, $t = 1$, and go to 1.
Seep 1. Let

\[ v = \operatorname{tmin}(\max s_j, \min s_j \mid s_j \geq y - y) \}, \]

\[ J = \{ s_j \mid s_j \geq v \}, \quad Q = \{ \sum s_j \mid s_j \geq y \}, \quad M = U \{ M_j \}. \]

Choose \( i(t) \) such that

\[ \text{IN...} \setminus N (= \min \mid N \setminus Q) \]

and let \( \{ c_j(t) \} \) be \( J \setminus N(i(t)) \).

Then set \( W = 1 + J(N(i(t)) \setminus Q), \quad 7 = 7 + s_j(t) - 1 \) if \( 7 > -\delta \). Go to 2.

Otherwise set \( S = S \setminus C_j(t) \),

\[ s_j = \begin{cases} s_j - s_j(t), & j \in N(i(t)) \\ s_j, & \text{otherwise} \end{cases} \]

\[ t = t + 1, \quad \text{and go to 1.} \]

Step 2. Add to (SC) the inequality

\[ Z \cdot x_i > 1. \]

This procedure terminates after a number of iterations equal to the number of \( j \in S(x) \) such that \( s_j > 0 \), with an inequality satisfied by every cover better than the one associated with \( z_n \), and violated by the cover \( x \).

Typically, the cuts tend to become successively stronger during the procedure, the last few cuts often having just one or two \( l \)'s. The total number of cuts required to solve an \( m \times n \) problem tends to increase with both \( m \) and \( n \). For the randomly generated sparse problems solved in our experiment, the number of cuts
needed was typically less than $3m$ or $n/3$. This of course refers to the number of cuts required when the cuts are used within the framework of an algorithm that also uses implicit enumeration. The cuts by themselves, without branching, were able to solve all 20 test problems from the literature that we could obtain, and all but one of 10 randomly generated test problems with $m = 100$ and $n = 100, 200, \ldots, 1000$. As to the larger problems, six of the ten $200 \times 1000$ problems and four of the ten $200 \times 2000$ problems that we generated, were solved by cutting planes only without branching.

**Branching and node selection.** We branch whenever the gap $\frac{U}{L} - z.$ decreases by less than $e - 0.5$ during a sequence of $a$ iterations, where $a$ is the frequency of applying the subgradient method (in number of iterations). The algorithm uses two branching rules intermittently. The first one is based on disjunction (3), the second one is a variant of the dichotomy proposed by Etcheberry [5]. Since our tests showed that none of the two rules dominates the other, we use both rules, with the following choice criterion: since rule 1 fixes more variables, but at the cost of creating more branches, we prefer rule 1 only if it fixes more variables than could be fixed by creating the same number of branches through binary (dichotomic) branching. More precisely, we choose rule 1 if, while creating $p$ branches, it fixes at least $p \log_2 p$ variables. As to node selection, we use the LIFO rule.

**Computational experience.** A detailed account of our computational experience is to be found in [3]. Here we reproduce only the results on the largest 10 test problems, a set of randomly generated problems with 200 constraints and 2,000 variables, with 8,000 non-zero entries in the coefficient matrix and with costs drawn from the interval $[1,100]$. The results are shown in Table 1.
Table 1. Results on 10 randomly generated problems.

<table>
<thead>
<tr>
<th>No.</th>
<th>$z_{opt}$</th>
<th>Before first branching</th>
<th>Nodes in search tree</th>
<th>Cuts</th>
<th>Time Dec 20/50 seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>253</td>
<td>256</td>
<td>250.6</td>
<td>30</td>
<td>473</td>
</tr>
<tr>
<td>2*</td>
<td>307**</td>
<td>315</td>
<td>299.3</td>
<td>&gt;51</td>
<td>&gt;625</td>
</tr>
<tr>
<td>3</td>
<td>226</td>
<td>226</td>
<td>226.0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>242</td>
<td>247</td>
<td>240.3</td>
<td>49</td>
<td>765</td>
</tr>
<tr>
<td>5</td>
<td>211</td>
<td>211</td>
<td>211.0</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>213</td>
<td>213</td>
<td>213.0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>293</td>
<td>296</td>
<td>291.0</td>
<td>173</td>
<td>298</td>
</tr>
<tr>
<td>8</td>
<td>288</td>
<td>288</td>
<td>286.1</td>
<td>125</td>
<td>413</td>
</tr>
<tr>
<td>9</td>
<td>279</td>
<td>281</td>
<td>276.2</td>
<td>181</td>
<td>118</td>
</tr>
<tr>
<td>10</td>
<td>265</td>
<td>265</td>
<td>265.0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

* Time limit of 10 minutes exceeded.
** Best solution found in 10 minutes.

Based on our computational experience, we can assert that the above described algorithm performs considerably better than earlier procedures proposed in the literature, and is in fact a reasonably reliable, efficient tool for solving large, sparse set covering problems, as well as for finding good approximate solutions to problems that are too hard to solve exactly.
References


Appendix I

Bibliography for Set Covering and Set Partitioning:

Theory and Algorithms

Andrew, G., Hoffmann, Th. and Krabek, Ch. [1968]: "On the Generalized Set Covering Problem." CDC, Data Centers Division, Minneapolis.


Balas, E. [1975a]: "Facets of the Knapsack Polytope." Mathematical Programming, 8, 146-164.


Balas, E. [1979]: "Set Covering with Cutting Planes from Conditional Bounds," in A. Prekopa (editor), Survey of Mathematical Programming, Hungarian Academy of Sciences, Budapest, 393-422.


1.2


Glover, F. and Klingman, D. [1973b]: "Improved Convexity Cuts for Lattice Point Problems." CS133, University of Texas, Austin, April.


II.1

Appendix II

Bibliography for Set Covering and Set Partitioning: Applications

(By area of application, chronologically within each area)

Crew Scheduling (Airline, Railroad, etc.)
II.2

13 Spitzer, M., "Solutions to the Crew Scheduling Problem", AGUORS Symposia, October (1971).

Airline Fleet Scheduling


Truck Delivers


Stock Cutting


Line and Capacity Balancing


Facility Location


II.3

Capital Investment

Switching Current Desira and Symbolic Logic


II.4

Information Retrieval

Marketing

Political Districting