Management Science

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Note—A Computational Survey of Methods for the Set Covering Problem

Nicos Christofides, S. Korman,

To cite this article:
Nicos Christofides, S. Korman, (1975) Note—A Computational Survey of Methods for the Set Covering Problem. Management Science 21(5):591-599. https://doi.org/10.1287/mnsc.21.5.591

MANAGEMENT SCIENCE
Vol. 21, No. 5, January, 1975
Printed in U.S.A.

Notes*

A COMPUTATIONAL SURVEY OF METHODS FOR THE SET COVERING PROBLEM*†

NICOS CHRISTOFIDES AND S. KORMAN

Imperial College of Science and Technology, England

This paper is a survey of available methods for the solution of the Set Covering Problem. The prime objective has been to establish the computational efficiency and relative merits of the various algorithms that have been proposed. To this end five methods have been programmed and tested on the same set of 33 test problems some of which can be found in the literature and the rest of which were randomly generated. Some of the methods examined have—as far as the authors are aware—never been previously tested, and in some cases some surprising results have been noted.

In addition, one type of tree search method has been studied in some greater detail and new techniques involving multiple dominance tests, and the calculation of a better lower bound have been suggested to limit the search. This resulting algorithm was found to be better than the original one by orders of magnitude in both computation times and numbers of tree-nodes generated, and has proved to be the most efficient of the methods tested

1. Introduction

This paper deals with the well-known "Set-Covering Problem", (SCP), which has wide applications in the fields of construction of optimal logical circuits [16], scheduling (and in particular aircrew scheduling) ([1], [15]) assembly line balancing [21], and information retrieval [6]. Additionally, certain other well-known problems can be formulated as SCP's, for example the graph-colouring problem which itself has a number of applications.

After formally stating the problem, three basic exact methods given in the literature for solving SCP's are mentioned: combinatorial tree search ([8], [12], [16], [17]) a specific type of (primal) integer programming ([4], [10]), and a variant of the classical Boolean algebra approach [11].

In §3, the tree search algorithm is developed further by deriving a new bound and by introducing elements of an incomplete dynamic programming procedure to limit the search. Finally in §4, five methods (four of which can be found in the literature and are variants of the three basic methods mentioned in §2, together with the improved tree search procedure discussed in §3), are coded and their computational performance compared on the same set of unicost test problems. Although each one of these five methods could be used to solve the general SCP, it was decided, in view of the great practical importance of the unicost SCP, and because of the lesser degree of problem-induced randomness, to concentrate on the latter type of SCP for the test examples.

* Processed by Professor Morton Klein, Departmental Editor for Network Flows; received June 1973, revised April 1974, May 1974. This paper has been with the authors 6 months for revision.
† Research supported by a grant from the Science Research Council, England.

1.1 *Problem formulation*

Given a set $X = \{x_1, \cdots, x_m\}$, and a family $S = \{S_1, \cdots, S_n\}$ of sets $S_j \subset X$, any subfamily $S' = \{S_{j_1}, S_{j_2}, \cdots, S_{j_k}\}$ of S such that

$$\bigcup_{i=1}^{k} S_{j_i} = X \tag{1}$$

is called a *set-covering* of X, and the S_{j_i} are called the *covering sets*. Also, if no $S'' \subset S'$ is a set-covering of X, then S' is called a *prime* set-covering. If, in addition to equation (1), S' also satisfies

$$S_{j_h} \cap S_{j_l} = \phi \ \forall\, h, l \in \{1, \cdots, k\}, \quad h \neq l \tag{2}$$

i.e. the $S_{j_i} (i = 1, \cdots, k)$ are pairwise disjoint, then S' is called a *set-partitioning* of X.

If, with each $S_j \in S$, there is associated a (positive) cost c_j, we wish to find that set-covering of X which has minimum cost, the cost of $S' = \{S_{j_1}, \cdots, S_{j_k}\}$ being $\sum_{i=1}^{k} c_{j_i}$. The above minimization problem is called the Set Covering Problem (SCP), the Set Partitioning Problem (SPP) being correspondingly defined.

In general, the costs c_j are positive integers, but the case often occurs, in practice, where the costs c_j are all equal (and thus may be taken as unity); this situation giving rise to the *unicost* SCP/SPP. This paper deals with the general SCP, but because of the importance of the unicost SCP, we shall often make specific reference to this special case.

1.2 *Problem reduction*

Due to the nature of the SCP it is often possible to make certain well-known *a priori* deductions and reductions (e.g. [8], [9], [18]). Henceforth we shall assume that these reductions, if applicable, have been applied and the original SCP reformulated in its irreducible form.

2. Methods of Solution

In this section we classify and summarize various methods proposed in the literature for solving the SCP. Further details of these methods can be found in [5] and [9]. Improvements and computational comparisons are given in later sections.

A. Tree search methods ([8], [9], [12], [16], [17])

As noted earlier, the SPP is closely related to the SCP, being essentially an SCP with an additional (no-overcovering) restriction. This (and in general any) restriction is advantageous when we attempt to solve the problem by an implicit enumeration (tree search) method, since it may allow the early abandonment of potential branches of the tree. Therefore, we shall first discuss a tree search algorithm for solving the SPP, and then show how it can be extended to solve the SCP.

I. *Solving the SPP.* In [8] a tree search method for solving the SPP was proposed by Garfinkel and Nemhauser, and is essentially as follows:

At the start of the process, we make up m 'blocks' of columns, one block for each element of X. Each block consists of those sets of S (represented by columns) which contain that element corresponding to the block, but which contain no lower-numbered elements. Thus block k will comprise of exactly those sets which contain x_k but do not contain any of $x_1, \cdots, x_{k-1}$. Each set, therefore, appears in exactly one block, and the totality of blocks can in general be arranged in tableau form as shown in Figure 1, although some block(s) may be nonexistent in a particular problem.

During the course of the algorithm, blocks are searched sequentially, with block k not being searched unless every element x_i, $1 \leqq i \leqq k - 1$, has already been covered

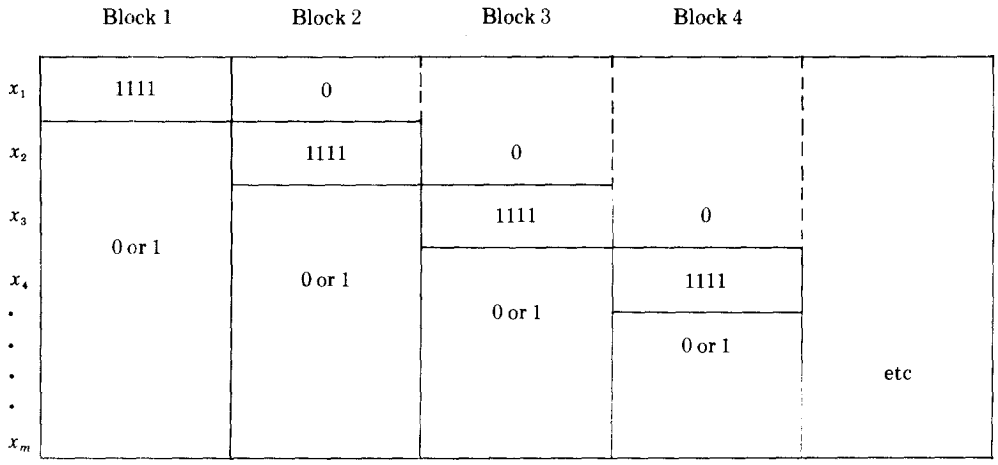

FIGURE 1. Initial Tableau

in the partial solution. Thus, if any set in block k were to contain elements indexed less than k, the set would have to be discarded (at this stage) due to the no-overcovering requirement.

The sets within each block are arranged (heuristically) in ascending order of their cost and the sets are now renumbered so that S_j will from now on be used to mean the set corresponding to the jth column of the tableau.

The tree search proceeds in the standard implicit-enumeration fashion [9] with the important advantage that the no-overcovering restriction forms a very powerful exclusion test.

A similar algorithm, incorporating a simple lower bound on the cost of the remaining subproblem, was also proposed by Pierce [16]; and in [17], Pierce and Lasky give some modifications to the basic algorithm, including the use of a better lower bound, and subsidiary use of an LP. A somewhat different algorithm for the SPP which also makes use of the block structure is that of Marsten [13], which is based on the fact that if g: $X \rightarrow K$ is any function from the set of elements X to the set of blocks K then $\bigcup_{k \in K} g^{-1}(k)$ induces a partition of X. It is shown in [13] that the set of feasible solutions to the SPP corresponds to a subclass of the class of all such functions g. Since g can be built up stage by stage in terms of the elements not yet assigned to blocks, the algorithm proceeds in the normal tree search manner, but with the additional facility that at each stage the corresponding LP is solved to serve both as a bound and as a means of directing the search.

In [14] Michaud describes another implicit enumeration algorithm which is based on an LP problem corresponding to the SPP, with the block structure given earlier being used in a secondary role; while other algorithms involving simplex-type iterations have been proposed, both primal ([2], [3]) and dual [20].

II. *Extension to the SCP.* An SCP can be easily shown, [16], to be transformable into an equivalent SPP (of larger dimension) by including S_j in each block corresponding to an element of S_j, so that S_j would now appear in $|S_j|$ blocks. Moreover, since an element x_{j_p} of S_j would, by the sequential nature of the search, be covered before branching on the sets of block q ($q > j_p$) is considered, it is now possible to remove all the elements x_{j_p} from a set S_j when entering S_j into any block $q > j_p$ without affecting the solution to the problem in any way [5].

In [12], Lemke, Salkin and Spielberg give another implicit enumeration algorithm in which a linear program is solved at any node of the search tree. The solution of this

LP is used as a lower bound during the search and is also used in determining the next forward tree branching from the current node. Apart from the use of the LP to direct and limit the tree search, the algorithm proceeds in much the same way as that of §I except for the fact that the order in which the rows are examined is not fixed, but depends on the last column (j_0) chosen for the forward branching. This has advantages because it allows one to branch on nodes from which only a small number of forward branchings emanate and hence, hopefully, reduce the number of future nodes to be searched. However, an important disadvantage results from this additional flexibility in that the block structure of the tableau used in the tree search of §I is not maintained, with the result that the processing time per node searched is increased.

B. Cutting plane approaches [4], [10]

In [10], House, Nelson and Rado describe a general method for the solution of the SCP which involves 'solving' a sequence of problems. Thus, given an SCP with 0-1 matrix A^P, one first finds any prime set-covering solution vector, $\tilde{y}^P$, of cost $\tilde{z}^P$. Then, if one can find a new (tth) constraint (or cut) such that

(i) All solutions of cost less than $\tilde{z}^P$ satisfy it and

(ii) $\tilde{y}^P$ does not, then matrix A^{P+1} is formed by adjoining the tth constraint to the rows of the matrix A^P. If at any stage p^*, no such constraint can be generated, the procedure terminates with an optimal solution $\bar{y}$ of cost $\bar{z}$, where $\bar{z} = \min_{0 \leqq k \leqq p^*} [\tilde{z}^k]$. A conceptually similar method is given by Bellmore and Ratliff, who describe in [4] a method for generating cutting constraints which is often computationally superior to that of [10].

C. Lawler's 'Duality' Method [11]

In [11], Lawler describes a method for solving the unicost SCP and generalizes it for the SCP. Basically, the idea is that given a unicost SCP whose optimal cover is of cost $\tilde{z}$ (as yet undetermined), one generates from this a further unicost SCP whose optimal cost is $\tilde{z} - 1$. Thus, by starting the process with the given SCP and terminating when a SCP is reached whose optimal cover is of cost 0 (i.e. the empty SCP), it is seen that $\bar{z}$, the optimal cost of the given unicost SCP, is equal to the number of intermediate problems generated, while the optimal cover itself can be established by means of a backtracking method.

3. Some Improvements to the Tree Search

We will now discuss some modifications to the simple tree-search algorithm of §§ 2I and 2II, which are shown to be of great computational significance in the next section.

3.1 *General Dominance Tests*

At any stage of the algorithm, its state can be defined by recording

R: the collection of sets S_j which are in the current partial solution,

z: the sum of their costs, with $\bar{z}$ being the cost of the best solution found so far, and

T: the set of elements x_i covered by the sets in R.

Let us now assume that we have saved some list $L(z_i)$ of previous sets T that have been attained during the course of the algorithm at a summed-cost level z_i. Suppose a stage is reached where $T = T'$, $R = R'$, $z = z'$, and we are about to examine block k (i.e. $k = \min \{i \mid x_i \notin T'\}$), and to consider set S_j^k of cost c_j^k for the next branching. If $z' + c_j^k < \bar{z}$, the original algorithm (of §2A) would have branched forward from

this stage, and updated $T = T' \cup S_j^k$, $R = R \cup \{S_j^k\}$, $z = z + c_j^k$, regardless of any other considerations.

However, one could now also ensure that

$$T' \cup S_j^k \nsubseteq T_l \,, \forall\, T_l \in L(z_i) \qquad \text{and for all } z_i \,,\ z' < z_i \leqq z' + c_j^k$$

before branching forward. If the set S_j^k fails the above test, then it is rejected and instead the next set, S_{j+1}^k, of block k is considered, etc. If S_j^k passes the test, one would branch forward on S_j^k, and continue as previously after updating $L(z)$ by introducing the set $T' \cup S_j^k$ into $L(z' + c_j^k)$.

Since it is generally impractical to save all the maximal sets T at any cost level (this being similar to a Dynamic Programming approach, or alternatively, a full breadth-first tree search to the problem as a whole), some heuristic criterion must be used to determine the size of the lists $L(z)$ and how they should be updated during the search.

3.2 *A New Lower-Bound*

At some stage of the search, given by R', T', z', and where block k is the next block to be considered, a bound, h, can be calculated as follows.

Consider an uncovered element $x_i \in X - T'$ which does not appear in any of the sets of those blocks k, $k+1, \cdots, i-1$, corresponding to elements not yet covered by the partial solution. The element x_i cannot then be covered unless some set S_j^i of block i is chosen to add to R' at a future stage. Thus, for each such element x_i, construct a row for a matrix $M = [m_{rs}]$ and a row for a second matrix $M' = [m'_{rs}]$, where m_{rs} is the number of elements in set S_s^r and m'_{rs} is its cost.

In addition, append an extra row, θ say, to M and M' with $m_{\theta s} = s$ for all $s = 0, 1, \cdots, m - |T'|$ and $m'_{\theta s} = s.\min [c_j^i / |S_j^i|]$, the minimum being taken over all sets S_j^i with $x_i \notin T'$. Note that although the number of elements in a row r, of M (or M') may be different from that of another row r_2, we assume here that O's and ∞'s are entered at the end of some rows of M and M' respectively, so that all rows have L (say) elements and the matrices become rectangular.

The following observations can now be made: since the optimal solution to the current subproblem must cover $m - |T'|$ elements, a choice of one entry from each row i of M, which satisfies $(m_{1s_1} + m_{2s_2} + \cdots + m_{\theta s_\theta}) \geqq m - |T'|$ and which minimises the corresponding cost, $v = (m'_{1s_1} + \cdots + m'_{\theta s_\theta})$ gives v as a lower bound to the optimal cost of the (covering) subproblem. Note that the assumption is made that the sets corresponding to the allocations in the rows of M are disjoint, which is obviously the best possible situation. The last row θ simply ensures that, if $\sum_{i=1}^{\theta-1} m_{is_i} < m - |T'|$, the remaining elements are covered in the best possible way, i.e. at minimal cost-per-additional-element-covered.

The minimal value of $\sum_{i=1}^{\theta} m'_{is_i}$ subject to $\sum_{i=1}^{\theta} m_{is_i} \geqq m - |T'|$ can easily be derived by a dynamic programming algorithm as follows:

Let $g_\rho(v)$ be the maximum number of elements that can be covered using only the first ρ rows of M (i.e. only ρ of the blocks in the subproblem), and whose total cost does not exceed v. $g_\rho(v)$ can then be calculated iteratively as:

$$g_\rho(v) = \max_{s=1, \cdots, L} [m_{\rho s} + g_{\rho-1}(v - m'_{\rho s})],$$

where $g_0(v)$ is initialized to 0 for all v.

Hence, the lowest value, v^*, of v, for which $g_\theta(v) \geqq m - |T'|$ is then the required lower bound, h, and can be easily obtained from the dynamic programming tableau derived from the above iterative equation. One should note that only that range of v

where $0 \leqq v < \bar{z} - z'$ need be considered, since if $h \geqq \bar{z} - z'$ (i.e. $g_\theta(v) < m - t$ for $v \geqq \bar{z} - z'$), a backtracking step may be taken.

4. Computational Results

Most of the methods described in the previous sections were coded in Fortran for the CDC 6600 computer, and this section presents computational comparisons of these programs on a large number of unicost set covering test problems. All programs used Boolean operations for set manipulation and although it may be possible to appreciably improve on any one program, approximately the same effort was devoted to coding each one so that comparisons of the computational times presented should be quite meaningful. The first group of 25 test problems (Tables 1 and 2) was obtained by generating random graphs, and using the set-covering methods to solve the graph-colouring problem associated with each graph. The elements (rows) for the SCP are the vertices of the graph, and the sets (columns) are the maximal internally stable sets of the graph, which were generated by a separate program.

The second group of 8 test problems (Table 3) was sent to the authors by H. M. Salkin, and is a subgroup of some of the problems previously used in the literature ([4], [12], [20]). The distinctive feature of these problems is that, for the most part, any column contains exactly two 1's, and no column reductions are applicable.

TABLE 1

Computational Times (CDC 6600 sec) Recorded for Various Methods

Problem	Number of Rows	Number of Columns	Density	Methods*				
				A	B	C	D	E
1	15	20	0.22	0.03	0.16	0.20	0.10	0.05
2	15	25	0.28	0.04	0.22	0.25	0.08	0.04
3	15	25	0.30	0.05	0.20	0.25	0.10	0.06
4	20	60	0.18	0.50	1.75	1.75	0.25	0.25
5	20	50	0.20	0.10	0.20	1.00	0.12	0.11
6	20	75	0.23	0.40	7.50	1.90	0.16	0.35
7	25	110	0.17	2.50	300.+	21.10	300.+	0.60
8	25	120	0.19	4.75	45.00	15.15	20.00	1.30
9	25	170	0.22	7.50	81.75	17.25	300.+	1.50
10	30	80	0.12	13.50	0.60	125.+	20.00	1.00
11	30	180	0.15	300.+	0.50	14.20	300.+	10.50
12	30	250	0.17	58.50	11.75	40.00	300.+	6.00
13	30	340	0.21	280.00	3.25	32.§	300.+	9.20
14	30	475	0.20	—	300.+	—	—	11.00
15	30	500	0.23	—	17.25	—	—	22.00
16	30	700	0.23	—	300.+	—	—	105.50
17	30	725	0.25	—	—	—	—	32.50
18	30	875	0.26	—	—	—	—	57.20
19	30	1000	0.27	—	—	—	—	72.80
20	35	160	0.09	—	300.+	—	—	18.50
21	35	290	0.13	—	—	—	—	28.00
22	35	460	0.16	—	—	—	—	75.50
23	35	585	0.18	—	—	—	—	129.50
24	35	780	0.19	—	—	—	—	141.20
25	35	1075	0.20	—	—	—	—	110.00

* A. PIERCE/GARFINKEL-NEMHAUSER. B. HOUSE-NELSON-RADO. C. BELLMORE-RATLIFF. D. LAWLER. E. CHRISTOFIDES-KORMAN

§ Problem run out of core storage after generating 30 cuts.

TABLE 2

Number of Cuts/Nodes/Iterations for Various Methods

Problem	Cuts/Nodes (in 1000's)/Iterations				
	A*	B†	C†	D††	E*
1	0.055	2	0	6	0.020
2	0.030	4	0	5	0.010
3	0.035	3	0	4	0.020
4	2.000	3	0	6	0.070
5	0.100	2	0	5	0.020
6	0.750	24	0	5	0.050
7	7.000	120^+	2	4^+	0.060
8	13.200	51	29	6	0.160
9	16.900	71	4	4^+	0.110
10	40.700	1	70^+	9	0.200
11	—	0	8	4^+	1.600
12	114.000	12	4	3^+	0.520
13	546.700	2	30§	3^+	0.300
14	—	70^+	—	—	0.090
15	—	8	—	—	0.400
16	—	65^+	—	—	3.020
17	—	—	—	—	0.120
18	—	—	—	—	0.180
19	—	—	—	—	0.190
20	—	95^+	—	—	2.180
21	—	—	—	—	2.030
22	—	—	—	—	2.950
23	—	—	—	—	3.960
24	—	—	—	—	5.030
25	—	—	—	—	0.550

* Number of nodes (in thousands) n Search Tree

† Number of Cuts needed.

†† Number of Subproblems Generated (see §2 c).

§ Problem run out of core storage.

TABLE 3

Computational Performance Recorded for Various Methods

Problem	Number of Rows	Number of Columns	Density	Methods: Computational Times (sec.)				Methods: Number of Cuts/Nodes (1000's)/iterations				α	β
				B	C	D	E	B	C	D	E		
26	30	60	0.07	$300.^+$	3.2	$300.^+$	0.5	130^+	0	11^+	0.160	18.0	10.5
27	30	60	0.07	30.00	26.5	0.4	1.7	50	38	14	0.650		
28	30	70	0.07	$300.^+$	2.9	70.7	1.7	130^+	0	14	0.620	20.4	9.3
29	30	70	0.07	$75.^+$	3.1	$300.^+$	1.4	70^+	0	11^+	0.430		
30	30	80	0.07	$75.^+$	4.5	$300.^+$	5.2	70^+	0	11^+	1.830	31.2	11.7
31	30	80	0.07	$75.^+$	4.8	$200.^+$	0.3	70^+	0	10^+	0.100		
32	30	90	0.07	$75.^+$	5.7	$200.^+$	6.7	70^+	0	8^+	1.600	30.6	**
33	30	90	0.07	$75.^+$	$120.^+$	$200.^+$	7.1	70^+	70^+	8^+	2.590		

$^\alpha$ Results reported by Lemke, Salkin and Spielberg in [12]. Times are in IBM 360/50 seconds and represent the average over the two problems.

$^\beta$ Results reported by Bellmore and Ratliff in [4]. Times are in IBM 7094 seconds and represent an average over (about) 10 problems.

** There is a misprint in Table 1 of [4] for these problems.

The five methods given in Tables 1 to 3 coded basically as described earlier, with only slight modifications (except for Method D). Some features of the results shown in the tables are listed below.

(i) Algorithm A is uniformly inferior to algorithm E. Although algorithm A can be improved by the embedding of an LP [17], it was decided (for the purposes of comparison) not to do so, since any such addition could be used in a similar context in other tree search methods (e.g. Algorithm E) and this would serve only to distract from the various branching and bounding distinctions between these methods.

(ii) Algorithm B behaves quite erratically, solving some large problems very quickly, while failing on some smaller ones.

(iii) Algorithm C also exhibits some variation in performance, although not as marked as that exhibited by algorithm B. It is perhaps interesting to note that very often, especially for the smaller problems, the round-up of the optimal LP cost is the optimal cost to the SCP.

(iv) Algorithm D was coded with various additional tests being used for implicitly eliminating r-fold combinations from $\bar{A}^P$ (see [11]). The inclusion of such tests improved the performance of this method by orders of magnitude, and although the results still remain unsatisfactory, it is felt that the potential of this method is not yet realized let alone exhausted.

(v) Algorithm E was coded and the list sizes were (arbitrarily) set at 60 for the first five levels of the search-tree and 30 thereafter. *A FIFO* (first in, first out) policy was used for updating the lists. As can be seen from the tables, this method was the most consistently effective of the methods coded, and compares favourably with the unicost results of other published algorithms.

References

1. ARABEYRE, J. P., FEARNLEY, J., STEIGER, F. C. AND TEATHER, W. "The Airline Crew Scheduling Problem, A Survey", *Transportation Science* 3 (1969) pp. 140–163.
2. BALAS, E. AND PADBERG, M. W., "On the Set Covering Problem", *Operation Research* 20 (1972) pp. 1152–1160.
3. BALAS, E. AND PADBERG, M. W., "On the Set Covering Problem. II. An Algorithm", *Management Sciences Research Report No. 295* May–Nov. 1972, Carnegie-Mellon University.
4. BELLMORE, M. AND RATLIFF, H. D., "Set Covering and Involutary Bases", *Management Science* 18 (1971) pp. 194–206.
5. CHRISTOFIDES, N., AND KORMAN, S. M., "Some Methods for the Set Covering Problem", Report. MP/74/1, Department of Management Science, Imperial College, London (1974).
6. DAY, R. H., "On Optimal Extracting from a Multiple File Data Storage System: An Application of Integer Programming", *Operations Research* 13 (1965) pp. 482–494.
7. GARFINKEL, R. S. "Optimal Political Districting", Ph.D. Thesis, The John Hopkins University, 1968.
8. —— AND NEMHAUSER, C. L., "The Set Partitioning Problem: Set Covering with Equality Constraints", *Operations Research* 17 (1969) pp. 848–856.
9. —— AND ——, *Integer Programming*, Wiley & Sons, 1972.
10. HOUSE, R., NELSON, L. AND RADO, T. "Computer Studies of a Certain Class of Linear Integer Programs", in *Recent Advances in Optimization Techniques*, (LEVI and VOGEL eds.) Wiley, N. Y., 1965.
11. LAWLER, E. L. "Covering Problems: Duality Relations and a New Method of Solution", *SIAM J. Appl. Maths.* 14 (1966) pp. 1115–1132.
12. LEMKE, C. E., SALKIN, H. M. AND SPIELBERG, K. "Set Covering by Single Branch Enumeration with Linear Programming Subproblems", *Operations Research* 19 (1971) pp. 998–1022.
13. MARSTEN, R. E., "An Implicit Enumeration Algorithm for the Set Partitioning Problem with Side Constraints", Ph.D. Dissertation, UCLA, 1971.
14. MICHAUD, P., "Exact Implicit Enumeration Method for Solving the Set Partitioning Problem", *IBM Journal. Res. & Dev.* 16 (1972) pp. 573–578.

15. PIERCE, J. F. "On the Truck Disgtching Problem—Part I", IBM Scientific Center Technical Report 320-2018, 1967.
16. ——, "Application of Combinatorial Programming to a Class of All-Zero-One Integer Programming Problems", *Management Science* 15 (1968) pp. 191–209.
17. —— AND LASKY, J. S. "Improved Combinational Programming Algorithms for a Class of All-Zero-One Integer Programming Problems", *Management Science* 19 (1973) pp. 528–543.
18. PYNE, J. B. AND MCCLUSKEY, E. J. JR., "An Essay on Prime Implicant Tables", *SIAM J. Appl. Maths.* 9 (1961) pp. 604–631.
19. ROY, B. "An Algorithm for a General Constrained Set Covering Problem", in *Computing and Graph Theory*, Academic Press, N. Y. 1972.
20. SALKIN, H. M. AND KONCAL, R., "Set Covering by an All Integer Algorithm: Computational Experience", *Jnl. of the Assoc. for Computing Machinery* 20 (1973) pp. 189–193.
21. SALVESON, M. E. "The Assembly Line Balancing Problem", *Journal of Industrial Engineering* 6 (1955) pp. 18–25.