MANAGEMENT SCIENCE
Vol. 19, No. 4, December, Part I, 1972
Printed in U.S.A.

CHROMATIC SCHEDULING AND THE CHROMATIC NUMBER PROBLEM*

J. RANDALL BROWN

Kent State University

The chromatic scheduling problem may be defined as any problem in which the solution is a partition of a set of objects. Since the partitions may not be distinct, redundant solutions can be generated when partial enumeration techniques are applied to chromatic scheduling problems. The necessary theory is developed to prevent redundant solutions in the application of partial enumeration techniques to chromatic scheduling problems with indistinguishable partitions and distinct objects. The chromatic number problem, which is the problem of finding the chromatic number of any graph, is a particular case of the chromatic scheduling problem. Two algorithms, basic and look-ahead, are developed for the chromatic number problem. Computational experience is given for each algorithm.

The chromatic scheduling problem may be defined as any problem for which the solution is a partition of a set of objects. A number of problems, including the loading problem, finding the coefficient of external or internal stability of a graph, finding the kernel of a graph, and finding the chromatic number of a graph, are all similar in that the solution to each problem is a partition of a set of objects. This paper will develop the necessary theory to prevent redundant solutions in the application of partial enumeration techniques to chromatic scheduling problems.

The bulk of the paper is devoted to developing two algorithms to solve the chromatic number problem (finding the chromatic number of any graph) which is a classical problem in graph theory. Indeed, one of the most famous problems in mathematics is to prove that the chromatic number of any graph that can be drawn on a sphere cannot exceed four. This is the four-color map problem. The two algorithms developed here will make it possible for researchers to explore graphs for which it has not been proven that four colors suffice. Of course, the algorithms also work for graphs that cannot be drawn on a sphere.

Redundant Solutions

Let the set of n objects to be partitioned be represented by the set of n variables $X = \{x_1, x_2, \cdots, x_n\}$. A solution can be defined as a set of sets $\{C_1, C_2, C_3, \cdots, C_m\}$ such that

$$C_i \subseteq X \quad \text{for} \quad i = 1, 2, \cdots, m, \qquad C_i \cap C_j = \emptyset \quad \text{for} \quad i \neq j,$$

and

$$C_1 \cup C_2 \cup \cdots \cup C_m = X.$$

If c_i represents membership in set C_i, then a solution can be depicted as assignment of a particular c_i to every x_j. Let c_i be called an attribute and then define the attribute set as $A = \{c_1, c_2, \cdots, c_m\}$. In most chromatic scheduling problems it is unnecessary to produce a solution that can be obtained by simply permuting or interchanging some attributes of another solution. For example, consider the following chromatic scheduling problem with $A = \{c_1, c_2\}$, $X = \{x_1, x_2, x_3\}$. The eight possible solutions are:

* Received July 1971; revised January 1972.

Variables	Possible Solutions
x_1	$c_1\ c_1\ c_1\ c_1\ c_2\ c_2\ c_2\ c_2$
x_2	$c_1\ c_1\ c_2\ c_2\ c_1\ c_1\ c_2\ c_2$
x_3	$c_1\ c_2\ c_1\ c_2\ c_1\ c_2\ c_1\ c_2$

In most cases, c_1 and c_2 are merely a means of separating the variables into two groups and have no intrinsic or different meaning. In this case, four of the solutions are redundant because they can be obtained by interchanging c_1 and c_2 in the other four solutions. One out of each of the following solution pairs is redundant:

Variables	Possible Solutions
x_1	$c_1c_2\ c_1c_2\ c_1c_2\ c_1c_2$
x_2	$c_1c_2\ c_1c_2\ c_2c_1\ c_2c_1$
x_3	$c_1c_2\ c_2c_1\ c_1c_2\ c_2c_1$

In this case, c_1 and c_2 are called indistinguishable attributes.

A redundant solution may now be defined as any solution that can be derived from another solution by interchanging two or more indistinguishable attributes. In partial enumeration, a partial solution p is defined as an assignment of attributes to a subset of variables $V_p \subseteq X$. Similarly, a redundant partial solution is defined as any partial solution that can be derived from another partial solution by interchanging two or more indistinguishable attributes. Enumeration proceeds by choosing a variable that does not have an attribute assigned to it. If there are m attributes, m new partial solutions are produced from p by assigning each of the m attributes to the chosen variable. The following theorem gives a technique that prevents the production of redundant solutions during partial enumeration. Consider an attribute set A where a subset $A' \subseteq A$ of attributes are indistinguishable among themselves but can be distinguished from the attributes not in A'.

THEOREM. *If A' is a subset of indistinguishable attributes of A, redundant solutions may be prevented by dividing A' into two sets, A'_U and A'_N, when a partial solution p is being used to produce new partial solutions. A'_U are those attributes of A' that are used in p and, conversely, A'_N are those attributes of A' not used in p. Those attributes $A'_E \subseteq A'$ that can be used with p to form the new partial solutions are $A'_E = A'_U \cup a'_N$ where a'_N is any attribute from A'_N.*

PROOF. The theorem shall be proved by induction. Assume that the set of partial solutions and solutions found thus far during enumeration contains no redundant partial solutions or redundant solutions. Choose a partial solution p to produce new partial solutions. A new partial solution cannot be a redundant form of any partial solution produced up to this point because the chosen p is not redundant. Therefore, only the partial solutions produced from p need to be checked for redundancy. Any of the partial solutions produced from attributes $a'_U \subseteq A'_U$ are not redundant because those attributes have been used in p. If more than one $a'_N \in A'_N$ is used, the partial solutions produced would be redundant with respect to each other, but not with respect to the partial solutions produced by A'_U. Thus the theorem is proved.

The theorem is easily extended to cases where the attributes are not all mutually indistinguishable by simply applying the theorem separately to each subset of mutually indistinguishable attributes. There are many chromatic scheduling problems in which all the attributes are not indistinguishable. Consider the problem of scheduling a set of jobs that require one operation which can be performed by any one of a number of different types of machines. Another example is a loading problem [9] where the boxes do not all have the same capacity.

The above results will be used in developing an algorithm for the chromatic number problem.

Chromatic Number Problems

Consider a chromatic scheduling problem where each constraint acts on only two variables and restricts those two variables from being assigned to the same set C_i. These constraints can be represented by an undirected graph $G = (X, \Gamma)$ with the vertices X representing the variables and the edges or arcs $(x, \Gamma x)$ representing the constraints. The notation for graphs used in this paper corresponds to that used by Berge [5]. A partition of the variable set X subject to the graph $G = (X, \Gamma)$ into the smallest number q of sets C_i is termed a minimal chromatic decomposition, while q is said to be the chromatic number of G.

Two algorithms to find the chromatic number q of an arbitrary graph will be developed. If s represents a solution, the problem may be reformulated. For variable set $X = \{x_1, x_2, \cdots, x_n\}$ and attribute set $A = \{c_1, c_2, \cdots, c_m\}$, minimize $f(s)$, the number of attributes used in solutions, subject to the undirected graph $G = (X, \Gamma)$ such that x and Γx cannot be assigned the same attribute c_i. All the attributes are indistinguishable. In partial enumeration, large blocks of possible solutions are eliminated by screening techniques which restrict the attributes that can be used to produce new partial solutions from partial solution p. Initially, all attributes may be assigned to the chosen variable x_k. Let U_k represent the attributes that can be assigned to x_k after the attributes have been screened.

The screening process consists of three independent parts. The first part of the screening process is due to all the attributes being indistinguishable and is given by the theorem in the last section. The second part is due to the graph and prohibits the chosen variable from being assigned any attribute that has been assigned to any variable in the partial solution that is connected to the chosen variable. The third part is due to the objective function, so that when a solution is found using j attributes, no more than $j - 1$ attributes may be used in the following enumeration. These three parts produce a set of attributes U_k that can be assigned to the chosen variable x_k.

Basic Algorithm

The enumeration technique that will be used is backtrack programming. Before the algorithm is applied, the variables are ordered such that variable x_k is connected to more of the variables $x_1, x_2, \cdots, x_{k-1}$ than any of the variables $x_{k+1}, x_{k+2}, \cdots, x_n$. Ties are broken by choosing the variable with the most edges. This preordering specifies which variable is to be chosen to augment the partial solution.

Figure 1 is a flow diagram of the basic algorithm. The index k refers to the variable being considered, and U_k is the set of unused attributes for variable x_k. The number of attributes used in the best solution found so far is q, and l is the number of attributes used in the current partial solution. L_k is l for every variable x_k.

Look-Ahead Algorithm

A natural question to ask at this juncture is whether the basic algorithm can be improved. One strategy that might yield improvement is to try and eliminate more

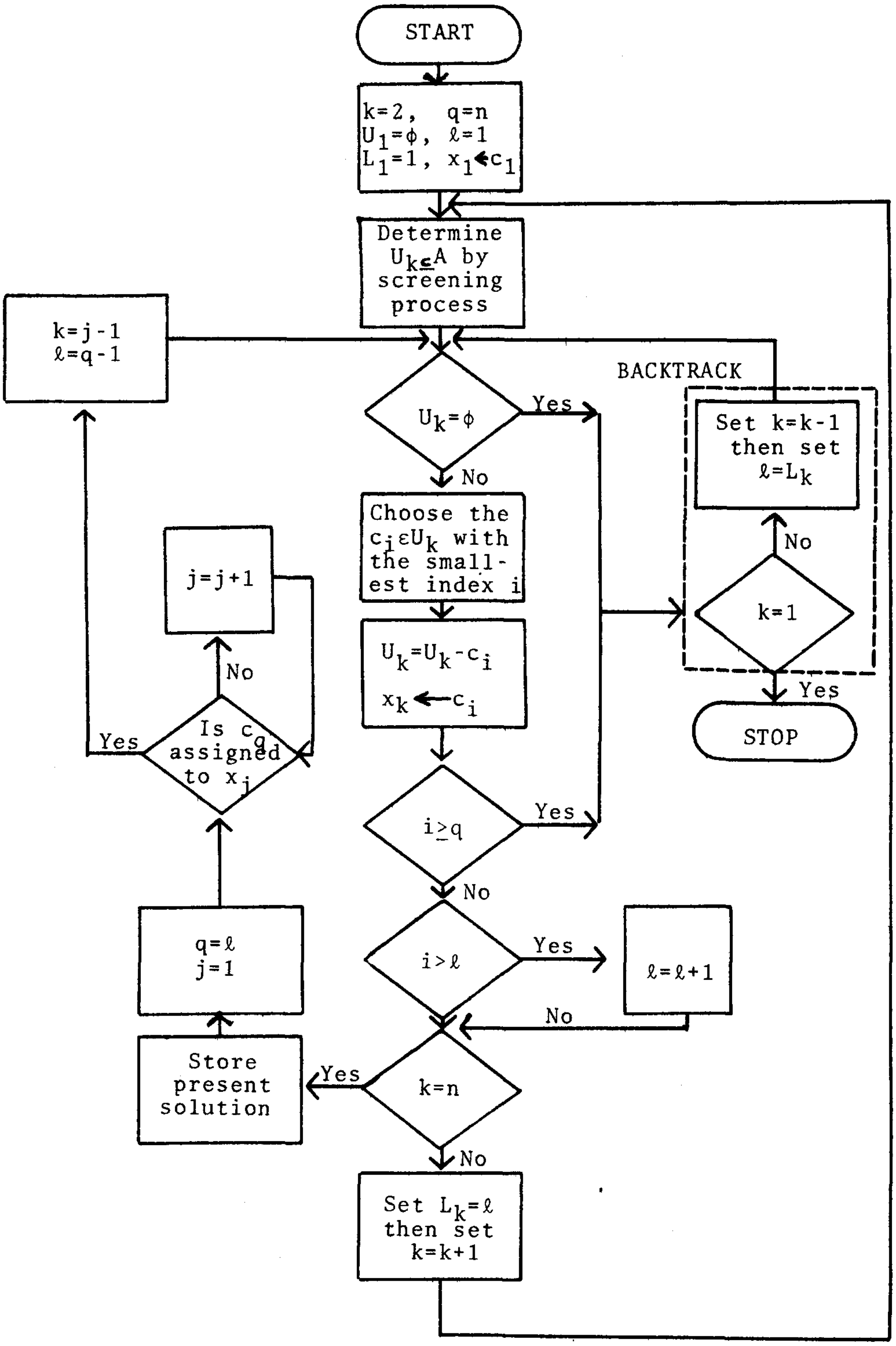
START
k=2, q=n
U1=φ, ℓ=1
L1=1, x1←c1
Determine Uk⊆A by screening process
k=j-1
ℓ=q-1
BACKTRACK
Uk=φ
Yes
No
Set k=k-1 then set ℓ=Lk
No
k=1
Yes
STOP
Choose the ci∈Uk with the smallest index i
j=j+1
No
Is cq assigned to xj
Yes
Uk=Uk-ci
xk←ci
i≥q
Yes
No
i>ℓ
Yes
ℓ=ℓ+1
No
q=ℓ
j=1
Store present solution
Yes
k=n
No
Set Lk=ℓ then set k=k+1

FIGURE 1

attributes during the screening process and thus reduce the number ot backtracks required. Of course, this would increase the time required for each iteration. The unanswered question then is whether the time saved by decreasing the number of backtracks will be significantly greater than the time spent per iteration.

The following procedure is an attempt to reduce the number of backtracks required by looking ahead and identifying the branches of the tree which cannot possibly yield a solution and, alternatively, identifying those branches which appear most promising. Thus, this procedure is called the look-ahead algorithm. During the screening process when the subset $U_k \subseteq A$ of attributes that may be assigned to x_k is being determined, the algorithm looks ahead and determines whether a candidate c_i for U_k would, at some later time, cause an increase in the number of attributes needed. The information gained by looking ahead is also used to determine which attribute $c_i \in U_k$ to assign to x_k.

More specifically, a record is kept for each x_k indicating the number of times that a variable x_i, where $i < k$ and x_i is connected to x_k, is assigned a particular attribute c_j. This record tells what attributes are available for U_k when the process iterates to x_k. For each of these candidate attributes c_j, the process is to find what effect assigning c_j to x_k would have on each x_l, where $l > k$ and x_l is connected to x_k. Two statistics are gathered for each c_j: the number of x_l's which in the future could possibly be assigned c_j unless c_j is assigned to x_k (hereafter called the number of preventions), and whether this assignment would increase the current upper bound in the chromatic number (hereafter called an increase of the chromatic number bound). If the assignment of an attribute c_j would cause an increase of the chromatic number bound to equal the current upper limit on the chromatic number (the upper limit is equal to the chromatic number of the last solution found), this attribute c_j is not included in U_k. The attributes $c_j \in U_k$ are ordered first, by whether they would increase the chromatic number bound, and, secondly, by the number of preventions. The first attribute c_j in this ordering is then assigned to x_k.

Computational Experience

The preceding algorithms were coded for a computer and 165 representative problems were run to estimate each algorithm's efficiency. The concept of the density d of an undirected graph will be needed. The total number of possible edges t in an undirected complete graph with n vertices is $t = (n^2 - n)/2$. If the number of edges in an undirected graph with n vertices is e, then the density d of the graph is defined as $d = e/t = 2e/(n^2 - n)$ and since $e \leqq t$, thus $0 \leqq d \leqq 1$. In terms of the efficiency of the algorithms, graphs of widely different densities will probably behave differently. Therefore, three sets of graphs with different densities were generated. Each set contains 55 graphs subdivided into 11 subsets. A graph with n vertices was produced by interrogating a uniform random number generator for each possible edge and including the edge if the generator said yes. The probability of yes was set at 0.25, 0.5, and 0.75 for each of the three sets so that the densities of the first set range from 0.20 to 0.30, the second set from 0.45 to 0.55, and the third set from 0.70 to 0.80. These three sets comprise a good representation of the entire scale of densities with the exception of those graphs with densities near zero or one. However, since a graph with a density of exactly zero or one can be solved for the chromatic number by inspection or with only one backtrack in the algorithms, graphs with densities near zero or one should not take much time to solve, and therefore have not been investigated in this paper.

The algorithms were coded in FORTRAN and run on a UNIVAC 1108. The results

TABLE 1

Look-Ahead versus Basic Algorithm, Graphs with Densities from 0.20 *to* 0.30

Number of Vertices	Number of Problems	Range in Time (in seconds)		Average Number of Backtracks		Average Time (in seconds)	
		BASIC	LOOK-AHEAD	BASIC	LOOK-AHEAD	BASIC	LOOK-AHEAD
20	5	0.003–0.007	0.007–0.016	2.4	1.8	0.0046	0.0102
22	5	0.004–0.008	0.009–0.013	2.4	1.2	0.0062	0.0104
24	5	0.004–0.010	0.013–0.023	3.2	2.8	0.0072	0.0160
26	5	0.005–0.025	0.013–0.021	5.2	1.6	0.0100	0.0150
28	5	0.006–0.038	0.018–0.057	10.6	4.4	0.0170	0.0298
30	5	0.011–0.102	0.020–0.116	35.2	13.2	0.0420	0.0566
32	5	0.011–0.065	0.021–0.108	12.0	7.6	0.0240	0.0480
34	5	0.010–0.062	0.026–0.080	21.2	9.6	0.0338	0.0568
36	5	0.069–0.678	0.030–0.216	272.4	30.0	0.2470	0.1404
38	5	0.019–0.369	0.027–0.147	89.2	14.0	0.1105	0.0802
40	5	0.025–0.154	0.050–0.400	70.6	33.2	0.0932	0.1470

TABLE 2

Look-Ahead versus Basic Algorithm, Graphs with Densities from 0.45 *to* 0.55

Number of Vertices	Number of Problems	Range in Time (in seconds)		Average Number of Backtracks		Average Time (in seconds)	
		BASIC	LOOK-AHEAD	BASIC	LOOK-AHEAD	BASIC	LOOK-AHEAD
20	5	0.004–0.011	0.008–0.018	5.4	3.0	0.0078	0.0124
22	5	0.005–0.078	0.010–0.099	20.8	9.4	0.0230	0.0312
24	5	0.006–0.072	0.011–0.065	28.4	9.4	0.0314	0.0320
26	5	0.006–0.104	0.013–0.072	32.6	9.8	0.0376	0.0424
28	5	0.010–0.325	0.037–0.190	124.8	32.4	0.1322	0.1132
30	5	0.023–0.133	0.018–0.087	66.4	10.4	0.0754	0.0478
32	5	0.037–0.502	0.024–0.244	163.0	30.2	0.2060	0.1248
34	5	0.041–1.781	0.056–1.843	533.8	134.2	0.6224	0.5246
36	5	0.801–6.427	0.185–3.061	2300.6	373.4	2.7104	1.4402
38	5	0.165–1.251	0.184–1.299	543.0	153.0	0.6850	0.6650
40	5	0.176–10.388	0.329–6.403	4293.4	835.8	5.3166	3.4934

are contained in Tables 1, 2, and 3. The look-ahead algorithm is clearly superior to the basic algorithm for problems with densities from 0.70 to 0.80 (Table 3). For problems with densities from 0.20 to 0.30 and from 0.45 to 0.55, the basic algorithm is superior to the look-ahead algorithm for 30 or less vertices. For problems with more than 30 vertices, the look-ahead algorithm appears to be better and should be much better for larger number of vertices because of its smaller growth rate.

A large real life problem was obtained from the people responsible for scheduling final examinations at the Massachusetts Institute of Technology. Each vertex of the graph represents an exam and each arc represents one or more students taking both the exams to which it is connected. One important question is what is the minimum number of time periods which must be utilized so that there are no conflicts when each exam is given only once. In other words, what is the chromatic number of the associated graph? Since the chromatic number was undoubtedly greater than eight, vertices with less than eight arcs were eliminated until all the vertices of the resultant

TABLE 3

Look-Ahead versus Basic Algorithm, Graphs with Densities from 0.70 *to* 0.80

Number of Vertices	Number of Problems	Range in Time (in seconds)		Average Number of Backtracks		Average Time (in seconds)	
		BASIC	LOOK-AHEAD	BASIC	LOOK-AHEAD	BASIC	LOOK-AHEAD
20	5	0.005–0.047	0.008–0.029	12.6	5.4	0.0174	0.0156
22	5	0.006–0.013	0.011–0.013	3.2	1.6	0.0088	0.0116
24	5	0.008–0.047	0.011–0.021	16.0	2.8	0.0220	0.0156
26	5	0.021–0.134	0.018–0.108	34.8	14.4	0.0516	0.0464
28	5	0.023–1.575	0.021–0.449	376.8	61.4	0.4666	0.1730
30	5	0.010–0.640	0.020–0.399	153.6	44.0	0.2306	0.1586
32	5	0.130–1.034	0.051–0.260	296.6	44.4	0.4100	0.1708
34	5	0.021–2.106	0.023–0.834	484.0	84.0	0.6406	0.3102
36	5	0.162–3.947	0.091–1.049	861.0	108.8	1.3614	0.4216
38	5	0.831–4.416	0.313–2.168	346.4	216.4	1.9794	0.9678
40	5	0.189–10.989	0.096–1.782	3095.6	174.2	4.8956	0.9308

subgraph had at least eight arcs. Of course, if the chromatic number of the subgraph exceeds seven, then the chromatic number of the graph is equal to the chromatic number of the subgraph. The basic algorithm was applied to the subgraph of 112 vertices and the chromatic number was found to be 15 with a solution time of 0.4 seconds with 162 backtracks.

Alternative Formulations

Two distinct integer programming formulations have been identified, but both require the solving of more than one integer program because each integer program will only test if the graph can be colored with a predetermined number of colors. One integer programming formulation can be found in Berge [5] and the other in Dantzig [8]. For both formulations, each integer program has more than one variable and one constraint for each edge of the graph. For a small problem with 20 vertices and a density of 25%, this would produce an integer program with more than 48 variables and 48 constraints or a zero-one linear programming problem of 80 variables and 210 constraints if the chromatic number is no greater than four.

To test the efficiency of these integer programming techniques, the chromatic number of a graph with six nodes was verified using the formulation by Berge [5]. The resulting zero-one linear programming problem had 18 variables and 42 constraints. This problem was solved using implicit enumeration [10] in 164.6 seconds on a Burroughs 5500. The basic algorithm presented in this paper solved the same problem in 0.02 seconds on a Burroughs 5500. Given the growth rate in the number of variables and constraints in the integer programming formulations, these techniques are undoubtedly very inefficient as compared with the two algorithms presented in this paper.

Hammer and Rudeanu [14] formulate a boolean method to find the chromatic number of any arbitrary graph. The method is composed of two distinct steps: first, all the maximal internally stable sets of the graph are found and, second, these maximal internally stable sets are used to form a nonlinear pseudo-boolean expression whose minimum is the chromatic number of the graph.

The first step is based on boolean algebra and would be very difficult to program for a computer. Also, the number of maximal internally stable sets produced for even a medium-size graph could be very large. Even if these difficulties were overcome, the second step would probably take much longer than the two algorithms presented in this paper. To test this assertion, the chromatic number was found by the boolean technique for the graph with six nodes used to test the integer programming formulations above. Six maximal internally stable sets were produced by hand in the first step and used to form a nonlinear pseudo-boolean expression that was converted to a zero-one linear programming problem by using a technique established by Watters [23]. The resulting problem had 11 variables and 10 constraints and was solved by implicit enumeration [10] in 1.23 seconds on a Burroughs 5500. Since the same problem was solved by the basic algorithm in 0.02 seconds on a Burroughs 5500, the boolean method for solving the chromatic number problem is undoubtedly very inefficient as compared with the two algorithms presented in this paper.

References

1. Agin, N., "Optimum Seeking with Branch and Bound," *Management Science*, Vol. 13 (1960), pp. B176–B185.
2. Balas, E., "An Additive Algorithm for Solving Linear Programs with Zero-One Variables," *Operations Research*, Vol. 12 (1965), pp. 517–546.
3. ——, "Discrete Programming by the Filter Method," *Operations Research*, Vol. 15 (1967), pp. 915–957.
4. Balinski, M. L., "Integer Programming: Methods, Uses, Computation," *Management Science*, Vol. 12 (1965), pp. 253–313.
5. Berge, C., *The Theory of Graphs and Its Applications*, John Wiley & Sons, 1962.
6. Brown, J. R., "Subductive Programming and Chromatic Scheduling," Unpublished Ph.D. Dissertation, Massachusetts Institute of Technology (1970).
7. Busacker, R. G. and Saaty, T. L., *Finite Graphs and Networks: An Introduction with Applications*, McGraw-Hill, 1965.
8. Dantzig, G. B., "On the Significance of Solving Linear Programming Problems with Some Integer Variables," *Econometrica*, Vol. 28 (1960), pp. 30–44.
9. Eilon, S. and Christofides, N., "The Loading Problem," *Management Science*, Vol. 17, No. 5 (January 1971), pp. 259–268.
10. Geoffrion, A. M., "Integer Programming by Implicit Enumeration and Balas' Method," RM-4783-PR RAND Corporation (1966).
11. ——, "Implicit Enumeration Using an Imbedded Linear Program," RM-5406-PR RAND Corporation (1967).
12. Glover, F., "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem," *Operations Research*, Vol. 13 (1965), pp. 879–919.
13. Golomb, S. W. and Baumert, L. D., "Backtrack Programming," *Journal of ACM*, Vol. 12 (1965), pp. 516–524.
14. Hammer (Ivanescu), P. L. and Rudeanu, S., *Boolean Methods in Operations Research*, Springer-Verlag, 1968.
15. Healy, W. C., "Multiple Choice Programming," *Operations Research*, Vol. 12 (1964), pp. 122–138.
16. Land, A. H. and Doig, A. G., "An Automatic Method of Solving Discrete Programming Problems," *Econometrica*, Vol. 28 (1960), pp. 497–520.
17. Lemke, C. E. and Spielberg, K., "Direct Search Algorithms for Zero-One and Mixed-Integer Programming," *Operations Research*, Vol. 25 (1967), pp. 892–914.
18. Little, J. D. C., Murty, K. G., Sweeney, D. W. and Karel, C., "An Algorithm for the Traveling Salesman Problem," *Operations Research*, Vol. 11 (1963), pp. 972–989.
19. Ore, O., *Theory of Graphs*, American Mathematical Society, 1962.
20. ——, *The Four-Color Problem*, Academic Press, 1967.
21. Peterson, E. E., "Computational Experience with Variants of the Balas Algorithm Applied to the Selection of R & D Projects," *Management Science*, Vol. 13 (1967), pp. 736–750.
22. Reiter, S. and Sherman, G., "Discrete Optimizing," *SIAM*, Vol. 13 (1965), pp. 864–889.
23. Watters, L. J., "Reduction of Integer Polynomial Problems to Zero-One Linear Programming Problems," *Operations Research*, Vol. 15 (1967), pp. 1171–1174.