-72

Computational Experience with a Group Theoretic Integer Programming Algorithm*

by

George Frank
G. Anthony Gorry and Jeremy F. Shapiro

May, 1972

WP 603-72

Computational Experience with a Group Theoretic Integer Programming Algorithm*

by

George Frank
G. Anthony Gorry and Jeremy F. Shapiro

May, 1972

WP 603-72

*This research was supported in part by the U. S. Army Research Office (Durham) under contract no. DAHC04-70-C-0058.

HD28
.M414
no. 603-72

Abstract

This paper gives specific computational details and experience with a group theoretic integer programming algorithm. Included among the subroutines are a matrix reduction scheme for obtaining group representations, network algorithms for solving group optimization problems, and a branch and bound search for finding optimal integer programming solutions. The innovative subroutines are shown to be efficient to compute and effective in finding good integer programming solutions and providing strong lower bounds for the branch and bound search.

Introduction

In this paper we report on computational experience with a new integer programming algorithm (IPA). Details of how group theory can be used to solve IP problems are discussed in Gorry and Shapiro [12]. Reference 12 also contains a brief discussion of some preliminary computational experience, but little detail was given there about computation and moreover, some of the subroutines in the preliminary version of the code were quite primitive. The current version of IPA contains subroutines which are fairly tight for code written in basic FORTRAN. Our primary purpose here is to try to establish that group theoretic IP methods as embodied by these subroutines are easy to construct and effective to use in solving IP problems. An important secondary purpose of this paper is to report on computational experience with the group theoretic constructs discussed abstractly in a number of papers (e.g., Glover [6, 7], Gomory [8, 9, 10], Hu [15, 16]).

We chose to implement IPA in FORTRAN because we desired to have the ability to transfer the system from one computer to another. We have found this ability particularly useful, and to date the system has been implemented on various versions of the IBM 360 System (Models 65, 67, 75 and 85) and on the Univac 1108. We have made the more obvious attempts to make efficient the encoding of this system in that we have been concerned with data structures and sequencing of operations within the system. On the other hand, we have not implemented any of this system in assembly language for a given machine. We do feel that such an implementation would significantly improve the computational efficiency of our system but the transferability of the FORTRAN had definite appeal for our purposes in the past.

Overview of IPA

We begin with a statement of the IP problem in its initial form. This problem is

$$\min z = \sum_{j=1}^{n+m} c_j w_j$$

$$\text{s.t.} \sum_{j=1}^{n+m} a_{ij} w_j = b_i \qquad i=1,\ldots,m \qquad (1)$$

$$w_j = 0,1,2,\ldots \qquad j \in U$$

$$w_j = 0 \text{ or } 1 \qquad j \in Z$$

where the coefficients c_j, a_{ij}, b_i are integers, U and Z are disjoint subsets of $\{1,2,\ldots,n+m\}$, and $U \cup Z = \{1,2,\ldots,n+m\}$. Let a_j denote a generic m vector with components a_{ij}, and let b be the m vector with components b_i. Problem (1) is solved first as a linear programming (LP) problem by a simplex algorithm. Without loss of generality, assume the optimal LP basic variables are $y_i = w_{n+i}$, $i=1,\ldots,m$, and the non-basic variables are $x_j = w_j$, $j=1,\ldots,n$. Let U_B and Z_B be the partition of the index set $\{1,\ldots,m\}$ into the index sets of unconstrained and zero-one basic variables; let U_N and Z_N be similarly defined for the non-basic index set $\{1,\ldots,n\}$.

The optimal LP basis B

$$B = \begin{pmatrix} a_{1,n+1} & \cdot\;\cdot\;\cdot\;\cdot\;\cdot & a_{i,n+m} \\ & & \\ a_{m,n+1} & \cdot\;\cdot\;\cdot\;\cdot\;\cdot & a_{m,n+m} \end{pmatrix} \qquad (2)$$

is used to transform (1) to the equivalent form

$$\min z = z_0 + \sum_{j=1}^{n} \bar{c}_j x_j \tag{3a}$$

$$\text{s.t. } y_i = \bar{b}_i - \sum_{j=1}^{n} \bar{a}_{ij} x_j \qquad i=1,\ldots,m \tag{3b}$$

$$y_i = 0,1,2,\ldots \qquad i \in U_B \tag{3c}$$

$$y_i = 0 \text{ or } 1 \qquad i \in Z_B \tag{3d}$$

$$x_j = 0,1,2,\ldots \qquad j \in U_N \tag{3e}$$

$$x_j = 0 \text{ or } 1 \qquad j \in Z_N \tag{3f}$$

where $\bar{c}_j \geq 0$, $j=1,\ldots,n$, $\bar{b}_i \geq 0$, $i=1,\ldots,m$, $\bar{b}_i \leq 1$, $i \in Z_B$.

The optimal LP solution is $x_j^* = 0$, $j=1,\ldots,n$, $y_i^* = \bar{b}_i$, $i=1,\ldots,m$. Clearly, the original IP problem (1) is solved if y_i^* is integer for all i. If one or more basic variables is not integer in the optimal LP solution, then some of the independent non-basic variables x_j must be set at positive integer values in order to make the dependent basic variables integer and non-negative. We call a non-negative integer vector $x = (x_1,\ldots,x_n)$ satisfying (3e) and (3f) a <u>correction</u> to the optimal LP solution. If the resulting y_i given by (3b) satisfy (3c) and (3d), then x is a feasible correction. The least cost feasible correction is an optimal or minimal correction.

Problem (3) is transformed to another equivalent problem as follows. Let Z_{q_i} denote the cyclic group of order q_i. Each non-basic activity a_j is mapped into an r-tuple of integers $\alpha_j = (\alpha_{1j},\alpha_{2j},\ldots,\alpha_{rj})$ where $\alpha_{ij} \in Z_{q_i}$ and the integer q_i satisfy $q_i \geq 2$, $q_i | q_{i+1}$, $\prod_{i=1}^{r} q_i = D = |\det B|$. Similarly, the right hand side vector b is mapped into the r-tuple of integers

$\beta = (\beta_1,\ldots,\beta_r)$ where $\beta_i \in Z_{q_i}$. The collection of D r-tuples of the form $(\lambda_1,\ldots,\lambda_r)$, $\lambda_i \in Z_{q_i}$, under addition modulo $(q_1,\ldots,q_r)$ form an abelian group and hence these methods are called group theoretic. The equivalent formulation to (1) and (3) is

$$\min z = z_0 + \sum_{j=1}^{n} \bar{c}_j x_j \qquad (4a)$$

$$\text{s.t. } y_i = \bar{b}_i - \sum_{j=1}^{n} \bar{a}_{ij} x_j \qquad i=1,\ldots,m \qquad (4b)$$

$$\sum_{j=1}^{n} \alpha_{ij} x_j \equiv \beta_i \pmod{q_i} \qquad i=1,\ldots,r \qquad (4c)$$

$$y_i \geq 0, \qquad i \in U_B \qquad (4d)$$

$$y_i \leq 1 \qquad i \in Z_B \qquad (4e)$$

$$x_j = 0,1,2,\ldots \qquad j \in U_N \qquad (4f)$$

$$x_j = 0 \text{ or } 1 \qquad j \in Z_N \qquad (4g)$$

Problem (4) is the starting point for the group theoretic approach. It is difficult to explicitly consider all the constraints in (4), and IPA progresses by ignoring certain constraints and solving the related problems which result. The optimal solutions to these problems are then used in a manner to be described later to find an optimal solution to (4). We simply mention here that this method of ignoring some constraints thereby creating easier optimization problems is called <u>relaxation</u> (see [5], [12], [13]). Specifically, if X is the set of feasible corrections in (4), then the algorithm will usually deal with sets X' satisfying $X \subset X'$. Then, if a

correction x* which is optimal in $\sum_{j=1}^{n} \bar{c}_j x_j$ subject to $x \in X'$ is feasible in (4), (i.e., $x^* \in X$), then x* is optimal in (4). Moreover, $\sum_{j=1}^{n} \bar{c}_j x_j^* \leq \sum_{j=1}^{n} \bar{c}_j x_j$ for all feasible corrections $x \in X$. These properties of relaxation are the basis for the branch and bound search of IPA, and also for the data manipulation schemes.

Table 1 gives timings of IPA of some real life IP problems. We will refer to these problems at various points during the paper. The 313 row airline crew scheduling problem was solved to approximately the same degree of optimality by the Ophelie Mixte code in 64.15 seconds on the CDC 6600. The IBM test problem was solved by Geoffrion's improved implicit enumeration code in 114 seconds on the IBM 7044.

A simplified flow chart of the IPA algorithm is presented in Figure 1. The major components of the system are indicated by boxes labelled with the names of the principal system subroutines (e.g. LP for linear programming, etc.). In what follows, we will discuss each component briefly and present some computational experience with that portion of the system.

Table 1

EXAMPLES OF RESULTS WITH IPA

OBLEM TYPE	No. of Rows	No. of Variables	LP Time	Group Rep. Time	Group Soln. Time	Search Time	Total Time [1]	Machine
DIA SELECTION	12	116	1.34	0.37	0.19	0.14	14.0	UNIVAC 1108
XED CHARGE	14	32	0.07	0.03	1.70	0.27	2.36	360/85
PITAL INVESTMENT	36	72	5.56	0.91	1.16	103.2	112.1	360/67
RSONNEL SCHEDULING	57	132	6.86	-	-	2.85	33.0	UNIVAC 1108
AND ALLOCATION	86	195	12.82	1.52	0.95	2.59	29.0	UNIVAC 1108
RLINE CREW SCHEDULING[2]	313	482	35.14*	5.10	0.05	150.0	192.5	360/85
RLINE CREW SCHEDULING	176	2385	49.26*	17.84	0.01	-0-	67.11	360/85
PER ROLL SLITTING	5	54	0.35	0.06	0.01	0.0	0.42	360/85
REEN LOOM CUTTING PATTERN SELECTION	27	641	4.38*	C.72	0.03	0.13	5.26	360/35
,1) APPL. UNKNOWN	26	383	10.08*	0.56	2.03	123.07	135.74	360/85
M TEST PROBLEM	50	65	0.39*	0.06	0.03	3.36	3.84	360/85

otal Time is in seconds.

easible integer solution with a cost within 3.5 percent of the minimal LP cost found y search in time indicated. Optimality was not proven, run was terminated with a aximum time cutoff.

P time for IBM 360 MPS.

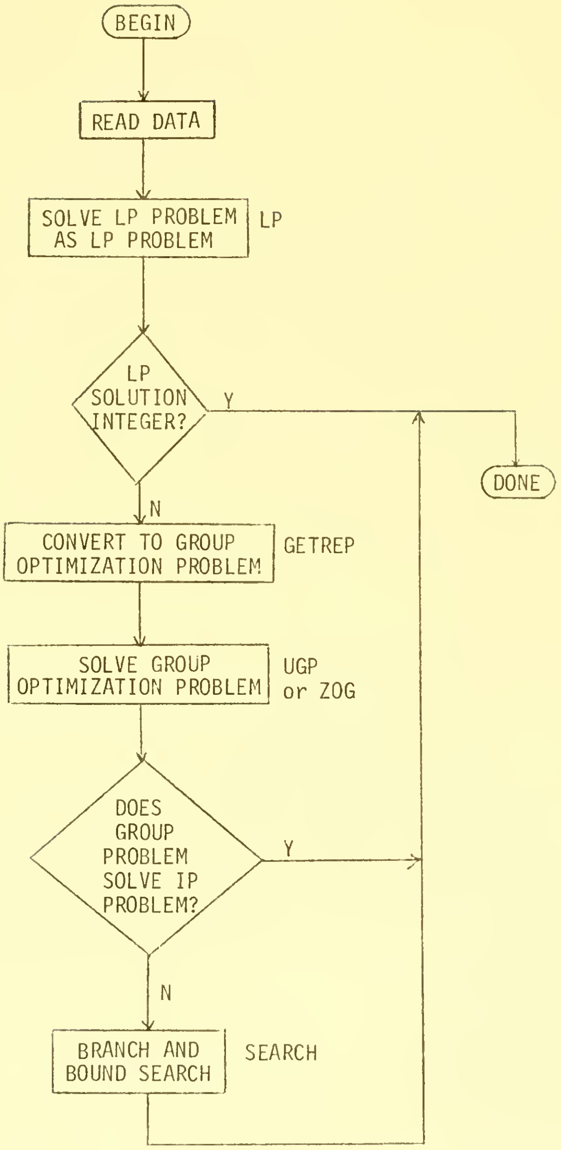

BEGIN
READ DATA
SOLVE LP PROBLEM AS LP PROBLEM
LP
LP SOLUTION INTEGER?
Y
DONE
N
CONVERT TO GROUP OPTIMIZATION PROBLEM
GETREP
SOLVE GROUP OPTIMIZATION PROBLEM
UGP or ZOG
DOES GROUP PROBLEM SOLVE IP PROBLEM?
Y
N
BRANCH AND BOUND SEARCH
SEARCH

Figure 1

IPA Linear Programming

IPA contains its own LP routine which can solve most LP problems with 200 rows or less, and a reasonable number of variables. We put less time and effort into this LP routine relative to that devoted to the more novel routines discussed below. The LP routine is an upper bounding variant of the simplex method written in FORTRAN IV. It uses single precision floating point arithmetic and therefore round-off problems can develop, especially if the determinant of the LP basis becomes large. Since large basis determinants make the group theoretic methods difficult to use, and moreover, since measures are used to try to prevent large determinants from occurring, the round-off problem is avoided and controlled to a certain extent. The procedures for controlling the determinants are discussed briefly below and in more detail in [13].

Nevertheless, round-off has sometimes been a problem for the LP routine and in these cases it was necessary to read in an optimal LP basis obtained from another system, e.g. the IBM 360 MPS linear programming code. We remark that it is not possible to use the final (optimal) LP tableau found by MPS because MPS does not give the determinant of the optimal LP basis from which the tableau is derived. Without knowledge of the determinant, the transformation to group optimization problems cannot be made.

The Group Representational Algorithm

The group representational algorithm (GETREP) is based on a method due to J. H. S. Smith for characterizing integer solutions to a system of linear equations by diagonalizing integer basis matrices. The original references are [20] and [21]. Wolsey [24] shows how Smith's method is used in IP; other discussions can be found in the paper by D. A. Smith [19] and the book by T. C. Hu [16]. We remark that Cabay in reference [1] uses similar congruential methods to find exact continuous solutions to systems of linear equations.

Conceptually, the procedure beings with the system of congruences

$$\sum_{j=1}^{n} \bar{a}_{ij}x_j \equiv \bar{b}_i (\text{mod } 1) \qquad i=1,\ldots,m, \qquad (5)$$

which is equivalent to the requirement that the independent non-basic variables x_j be chosen in such a way that the dependent basic variables y_i are integer. The subroutine GETREP transforms (5) to the system of congruences (4c) which we rewrite here

$$\sum_{j=1}^{n} \alpha_{ij}x_j \equiv \beta_i (\text{mod } q_i) \qquad i=1,\ldots,r.$$

The reduction is necessary for efficient computation as r is usually on the order of 1 to 5 regardless of the value of m.

The reduction of the system of congruences is achieved by reduction of the m x m matrix $F = f_{ki} = D\{\bar{a}_{k,n+i} - [\bar{a}_{k,n+i}]\}$, $k=1,\ldots,m$, $i=1,\ldots,n$. Through a series of passes through F, the matrix is diagonalized. Each of

these passes consists of finding the minimum non-zero element in F and using the row and column in which that element exists for elementary row and column operations on F. These row (column) operations are like the typical Gauss-Jordan reduction steps except that the constant multiple of a row (column) which is to be subtracted from another row (column) is determined in integer arithmetic and the reductions are made modulo D. In the event that a pass through the matrix leaves a given row and column with the only non-zero element at their intersection, the row and column are in general removed from any further consideration in the diagonalization procedure. The only exception to this is when the non-zero intersection element does not divide the determinant or does not divide the similar preceding element. In such a case the representational algorithm "backs up" and continues the process. We will omit here details of this last maneuver. Basically, the purpose for it is to insure that the minimal group representation is obtained by the algorithm. The diagonalization procedure is complete when the product of the non-zero intersection element equals the determinant of the basis matrix.

If the diagonalization procedure has revealed r subgroups (that is, if r elements were required to form a product equal to the determinant), then the α's are determined by the first r inner products of the rows of a matrix C generated from the elementary column operations of diagonalization procedure and the non-basic columns taken modulo the corresponding subgroup orders.

Table 2a. Group Representational Algorithm: Computational Experience

(*denotes IBM 360/67; all others Univac 1108)

	Number of Rows m	Determinant D	Number of Subgroups r	Time in Seconds t	Number of Non Basics n
1	6	180	2	0.90	240
2	6	1080	3	0.92	240
3	6	10	1	1.13	300
*4	9	128	3	0.06	14
5	12	5832	6	0.35	104
6	12	14400	5	0.38	104
7	18	81600	3	1.14	240
8	18	55296	5	1.15	240
9	18	900	2	1.11	240
10	18	1158	1	1.01	240
11	18	960	2	1.04	240
12	18	23328	4	1.14	240
13	18	96	2	1.05	239
*14	36	384	4	0.91	36
*15	36	144	2	0.73	36
*16	46	420	1	3.41	115
*17	50	21744	1	2.19	115
*18	50	36288	1	2.90	115
19	86	280	2	1.46	109
20	86	140	1	1.40	109
21	86	1080	1	1.82	109
22	86	17100	2	1.53	109
23	86	540	2	1.37	109
24	86	380	1	1.60	109
25	91	2048	3	1.67	104
26	91	512	3	1.45	104

Table 2b. Group Representational Algorithm:
Computational Experience

(IBM 360/85)

	Number of Rows m	Number of Non basics n	Determinant D	Number of Subgroups r	Time in Seconds t
1	9	12	64	2	0.02
2	9	14	128	3	0.00+
3	11	11	384	2	0.03
4	14	18	5025	1	0.03
5	36	36	864	4	0.12
6	36	36	1152	5	0.14
7	36	36	5760	5	0.15
8	36	36	6912	5	0.14
9	40	58	2	1	0.07
10	50	15	36	1	0.06
11	313	169	48	2	5.10

In Tables 2a and 2b we have presented some representative times for the algorithm on a variety of integer programming problems. As can be seen from this figure, in spite of the fact that the algorithm is written in FORTRAN, the computational requirements to obtain group representations are relatively small. We have not performed any detailed study on the relationship between the computational time required for the representational algorithm and such variables as the number of rows in the problem or the number of non-basic columns. There are, however, certain general observations we can make based on the data presented. In general the computational requirement increased with the number of rows in the integer programming problem because the number of elements in the matrix F which must be considered is equal to the square of the number of rows. Thus, increasing the number of rows increases the amount of computation required to diagonalize the fractional matrix.

On the other hand, our experience to date has not shown the number of subgroups to be an important factor in determining the computational time. The reason for this is that a single pass through the fractional matrix will produce many zero elements in that matrix. Subsequent passes through the matrix will then generally find a number of constant multiples for rows and columns which are zero and hence effectively can be ignored. The number of non-basic columns in the problem is also an important determinant of the computational time. Once C has been obtained, the calculation of the group identities for the non-basics requires the multiplication of some number of rows of C be each of the non-basic columns in turn. As a result, if the number of non-basics is quite large, then the computational time for the representational algorithm will also be relatively large. Notice, however, that these times will show only a relatively large increase.

In other words, the absolute time required to obtain the representation is in general quite small and for the problems which we have dealt with where the number of rows and columns is less than say 300, this time is on the order of one to three seconds.

In spite of the promising computational experience with GETREP, further research is required into group representational theory and IP. If many group problems are formulated and used during the branch and bound search, then faster group representation times are desirable. Johnson [14] has had success with a fast method of approximating the groups as cyclic groups.

Research is also required to gain a better understanding of the relationship between a group structure and the IP problem from which it is derived. For example, changing one number in the basis B can radically change the group structure. Useful structural relationships may also be identifiable between subgroups in the representation and subsystems of the IP problems. Tompkins and Unger [23] have done this for the fixed charge problem. Finally, it is interesting to note that Gomory originally expected the groups to be cyclic in most cases ([8; p. 279]), but computation thus far is counter to that expectation.

When GETREP is finished, it has produced the group representations of the non-basics (the α_j's) and that for $\bar{b}$ (denoted by β). IPA can now solve the implied shortest route problem as discussed in the next section.

Group Optimization Problems

After GETREP has obtained the group representation (4c), IPA solves one of two group optimization problems. These problems are constructed by ignoring the non-negativity constraints on the basics and possibly some upper bound constraints. As mentioned in the overview, this approach will provide either an optimal correction in (4), or the necessary information about lower bounds and trial corrections from which to do a search for an optimal correction.

One group optimization problem is the unconstrained group problem (UGP): For k = 0,1,...,D-1, find

$$\begin{aligned} G(\lambda_k) &= \min \sum_{j=1}^{n} \bar{c}_j x_j \\ \text{s.t.} &\sum_{j=1}^{n} \alpha_{ij} x_j \equiv \lambda_{ki} (\text{mod } q_i),\ i = 1,\ldots,r \\ x_j &= 0,1,2,\ldots;\quad j = 1,\ldots,n \end{aligned} \tag{6}$$

This problem is also referred to as the asymptotic problem by Gomory [9].

Problem (6) is a shortest route problem (see [12],[17]) in a network with D nodes, one for each r-triple λ_k, directed arcs $(\lambda_k - \alpha_j, \lambda_k)$, $j = 1,\ldots,n$, drawn to every node λ_k except the node $\lambda_0 = 0$. The arc length or cost associated with an arc type j is $\bar{c}_j$.

The unconstrained group problem solves (6) for all right hand sides λ_k rather than the specific right hand side β in problem (4) because the additional paths are important if a search is required. An optimal solution $x(\lambda_k)$ to (6) with right hand side λ_k is extracted from the shortest route

path by letting $x_j(\lambda_k)$ equal the number of times an arc j is used in the optimal path.

The other group optimization problem which IPA sometimes uses is problem (6) with additional upper bound constraints (often upper bounds of one) on some of the non-basic variables. This problem is called ZOG. The algorithm for solving this problem is given in reference [12]. Consistent computational times for it are not available since it has undergone several changes. Indications are, however, that it is only slightly slower than UGP for problems with basis determinants of less than 1000 although it does require more core storage than UGP.

Tables 3a and 3b give times for the solution of (6) using the unconstrained group problem algorithm of [12]; this algorithm was originally proposed in [17]. The algorithm finds in a few seconds the entire shortest route tree for IP problems with basis determinants of up to 5000. The determinant is the primary factor in determining the computation time of UGP but the number of non-basics n and the difficulty of the problem also are important factors. Notice also that the time appears to increase linearly in these factors.

There are three other algorithms in the literature for solving group optimization problems. The first was the algorithm proposed by Gorony in [9] in which the group optimization problem (6) was originally formulated. Published computational experience with this algorithm does not appear to exist.

A second algorithm for solving group optimization problems is given by Glover in [6] which includes some computation times for randomly

generated group optimization problems. It is difficult to make comparisons between Glover's results and our own because the characteristics he gives for randomly generated group optimization problems are different from the ones from actual IP problems.
Specifically, for group problems derived from real IP problems the α_j in (6) do not appear to be randomly generated and the range of arc costs is definitely greater than the range given by Glover. These characteristics of group optimization problems derived from real IP problems should make them easier to solve than randomly generated ones because a relatively small number of non-basic activities with low costs dominate the shortest route solutions.

Finally, there is the algorithm of Hu [15] which is an adaptation of the shortest route algorithm of Dijkstra [2]. The essential difference between Hu's algorithm and UGP is the manner of selecting a path in the network to extend. Hu's algorithm searches through all the paths for which extension is indicated to find the minimal cost one, whereas UGP selects the first such path it can find. We have experimented briefly with the Dijkstra-Hu approach by making the appropriate change in UGP; it appears that this change causes the time to increase by as much as a factor of five to ten (Hu does not give any computational experience in [15]). The explanation of why the Dijkstra method might be ineffective in this case is because the group shortest route network has special structure; namely, the same types of arcs with the same arc costs are drawn to every node. Thus, low cost dominating arcs are discovered quickly and can be used repeatedly.

Table 3a. Solution of the Unconstrained Group Optimization Problem: Computational Experience

(*denotes IBM 360/67; all others Univac 1108)

	Number of Non Basics n	Determinant D	Number of Subgroups r	Time in Seconds t
1	300	10	1	0.06
2	104	12	1	0.04
3	104	24	2	0.07
4	104	24	2	0.18
5	240	24	2	0.20
6	104	44	1	0.19
7	104	48	3	0.14
8	239	96	2	0.96
9	240	120	3	1.11
*10	36	144	2	0.30
11	109	140	1	0.58
*12	14	128	3	0.27
13	104	144	2	0.76
14	109	168	2	0.89
15	240	180	2	0.68
16	104	192	2	0.64
17	104	192	2	0.65
18	109	200	2	0.95
19	240	240	3	1.22
20	109	280	2	1.93
21	104	288	4	1.00
22	109	380	1	1.18
23	109	380	1	1.40
*24	36	384	4	1.16
25	104	512	3	5.86
26	109	540	2	2.53
27	240	900	2	12.34
28	240	960	2	3.83
29	104	972	3	1.83
30	109	1080	1	4.20
31	240	1080	3	7.89
32	240	1158	1	3.98
33	109	1260	2	12.50
34	240	1440	3	8.21
35	104	2048	3	12.88
36	103	3780	2	7.62
37	104	4752	2	13.27

Table 3b. Solution of the Unconstrained Group Optimization Problem: Computational Experience

(IBM 360/85)

	Number of Non Basics n	Determinant D	Number of Subgroups r	Time in Seconds t
1	58	2	1	0.00+
2	195	48	2	0.05
3	12	64	2	0.01
4	14	128	3	0.03
5	11	384	2	0.05
6	36	864	4	0.21
7	36	1152	5	0.29
8	18	5025	1	1.70
9	36	5760	5	3.74
10	36	6912	5	1.55

Johnson [14] has developed a special algorithm to solve cyclic group problems quickly where the cyclic groups are approximations to subgroups of the true group implied by the matrix B.

We now turn to the question of when the group problem or asymptotic problem (6) does in fact solve the IP problem (4) from which it was derived. In reference [16] on page 347, T. C. Hu states, "Although we have a sufficient condition that tells when the asymptotic algorithm works, actual computation will reveal that the algorithm works most of the time even if the sufficient condition is not satisfied." Computation has in fact revealed that solving the asymptotic problem does not solve the original problem for most real life IP problems of any size, say problems with more than 40 rows. The positive search times on all but two of the problems in Table 1 indicates that the solutions to the asymptotic problems (6) were infeasible in the original problems. The difficulty is compounded by the fact that problem (6) almost always has many alternative optimal solutions, only one of which may be feasible and therefore optimal in (4). This was the case, for example, for the 57 row clerk scheduling problem. The optimal IP correction was optimal in (6) but it had to be found during search. The asymptotic problem does provide good lower bounds in objective function costs and good candidate solutions for the search which is discussed in the next section.

Search Procedures in IPA

The branch and bound search performed by IPA implicitly considers all non-negative integer corrections x in (4). At an intermediate point of computation, the minimal cost solution found thus far is called the incumbent and denoted by $\hat{x}$ with incumbent cost $\hat{z} = z_0 + \bar{c}\hat{x}$. The algorithm generates IP subproblems of the following form from explicitly enumerated non-negative integer vectors $\tilde{x}$

$$z(\tilde{x}) = z_0 + c\tilde{x} + \min \sum_{j=1}^{j(\tilde{x})} \bar{c}_j x_j \qquad (7a)$$

$$\text{s.t.} \sum_{j=1}^{j(\tilde{x})} \bar{a}_{ij} x_j \leq b - \sum_{j=j(\tilde{x})}^{n} \bar{a}_{ij} x_j \quad i = 1,\ldots,m \qquad (7b)$$

$$\sum_{j=1}^{j(\tilde{x})} \alpha_{ij} x_j \equiv \beta_i - \sum_{j=j(\tilde{x})}^{n} \alpha_{ij} \tilde{x}_j \pmod{q_i} \quad i=1,\ldots,r \qquad (7c)$$

$$y_i \geq 0 \qquad i \varepsilon U_B \qquad (7d)$$

$$y_i \leq 1 \qquad i \varepsilon Z_B \qquad (7e)$$

$$y_j = 0,1,2\ldots \qquad j \varepsilon U_N \cap \{1,2,\ldots,j(\tilde{x})\} \qquad (7f)$$

$$x_j = 0 \text{ or } 1 \qquad j \varepsilon Z_N \cap \{1,2,\ldots,j(\tilde{x})\} \qquad (7g)$$

where $j(\tilde{x})$ is either the smallest index j such that $\tilde{x}_j>0$ if this index corresponds to an unbounded variable, or it is the smallest index minus 1 such that $\tilde{x}_j>0$ if this index corresponds to a zero-one variable. The summations in (7) are restricted to the range 1 to $j(\tilde{x})$ in order that the search be non-redundant.

If we can find an optimal solution to (7), then we have effectively tested all non-negative integer corrections $x \geq \tilde{x}$, $x_j = \tilde{x}_j$, $j = j(\tilde{x}) + 1,\ldots,n$,

and they do not have to be explicitly enumerated. The same conclusion is true if we can ascertain that $z(\tilde{x}) \geq \hat{z}$ without discovering the precise value of $z(\tilde{x})$. If either of these two cases obtain, then we say that $\tilde{x}$ has been fathomed. If $\tilde{x}$ is not fathomed, then we abandon (7) and create new sub-problems of the same form as (7) from the solutions $\tilde{x} + e_j$, $j = 1,\ldots,j(\tilde{x})$, where e_j is the j^{th} unit vector.

We try to fathom (7) in the same manner that we tried to solve (4); namely, by ignoring some of the constraints. If problem (6) was the group optimization problem selected before search began, then we use the group optimization problem

$$G(\beta - \sum_{j=1}^{n} \alpha_j \tilde{x}_j) = \min \sum_{j=1}^{n} \bar{c}_j x_j$$

$$\text{s.t.} \quad \sum_{j=1}^{n} \alpha_{ij} x_j \equiv \beta_i - \sum_{j=j(\tilde{x})}^{n} \alpha_{ij} \tilde{x}_j \pmod{q_i} \quad i = 1,\ldots,r \qquad (8)$$

$$x_j = 0,1,2,\ldots \qquad j = 1,\ldots,n$$

The use of (8) in trying to fathom (7) is given in Figure 2. Here we will comment briefly on the various components of the search.

Our discussion of the search will assume the use of UGP; at the user's choice, ZOG can be used.

In a problem with zero-one variables, the latter will yield better bounds for the fathoming test. The basic generation of the search, however, would be the same as with UGP.

The solution to the group problem chosen is maintained in core in a list-structured form which permits the rapid retrieval of backtracked solutions to

(8). Basically, for each node of the group problem, the system stores the cost of the shortest-route path to the node, the identity of the arc into the node, and a pointer which can be used to reconstruct the whole path back to zero in the group network.

The remaining free core memory is used by the search routine for the storage of subproblems. The space is used as a buffer area in which blocks of subproblems for consideration are stored. As the search generates more subproblems, blocks of them are moved to disk storage, generally to be returned to core later when the space becomes available. Through the use of secondary storage, the system is able to maintain a very large number of active subproblems while it searches for an incumbent.

Step 1: Here a subproblem is chosen for the fathoming tests. In the current system, subproblems are considered in the order of their generation. For each subproblem, the system stores: 1) the bound obtained when the subproblem was generated; 2) the index of the node $\beta(\tilde{x}) = \beta - \sum_j^n \alpha_j \tilde{x}_j$ for the particular $\tilde{x}$ in question; 3) $\tilde{x}$ itself (with only the non-zero elements stored as pairs $(j_i, \tilde{x}_{j_i})$); and 4) the shortest route path from $\beta(\tilde{x})$ to 0.

Step 2: If the bound $z_0 + G(\beta - \sum_{j=1}^n \alpha_j \tilde{x}_j)$ is greater than or equal to the current incumbent cost $\hat{z}$ then the subproblem is deleted, because it is fathomed by bound. As will be seen below, this is the second time this subproblem has been tested for bound. The first time was when it was created. Because the subproblem is still active, we know the first check failed to fathom it. In general, a very large number of other subproblems have been considered

since then, and a new incumbent with lower cost may have been found. Hence this second test may fathom the correction.

If the subproblem is not fathomed by bound, it is continued. Basically, for each $\ell \leq j(\tilde{x})$, a new correction of the form $\tilde{x} + e_\ell$ is generated. This correction is tested for bound by comparing

$$z_0 + c_j + G(\beta - \sum_{j=1}^{n} \alpha_j \tilde{x}_j - \alpha_\ell)$$

with the incumbent cost. If the bound for this new correction is less than $\hat{z}$, the group network is used to backtrack this correction. If $\tilde{x} + e_j + x^t$ is feasible, then a new incumbent has been found, and it is recorded along with its costs. (Parenthetically, we note that by maintaining $\bar{b}$ and $\bar{a}_j$ cleared by the determinant, the system can use integer arithmetic to test feasibility in (4b). Integrality in (4b) is guaranteed through the use of the group problem.)

If the new continuation is not feasible, then it is saved on the subproblem list for later consideration.

In fact the search routine does not generate all continuations $\tilde{x} + e_j$, $j \geq j(\tilde{x})$ for a given $\tilde{x}$. Recall that the backtracked solution from the group network, x^t, is stored with the subproblem. It can be shown that any correction $\tilde{x} + u \leq \tilde{x} + x^t$ will be infeasible if $\tilde{x} + x^t$ is infeasible, and moreover the former will be backtracked to $\tilde{x} + x^t$. Hence only descendants $\tilde{x} + u \not\leq \tilde{x} + x^t$ are created and tested.

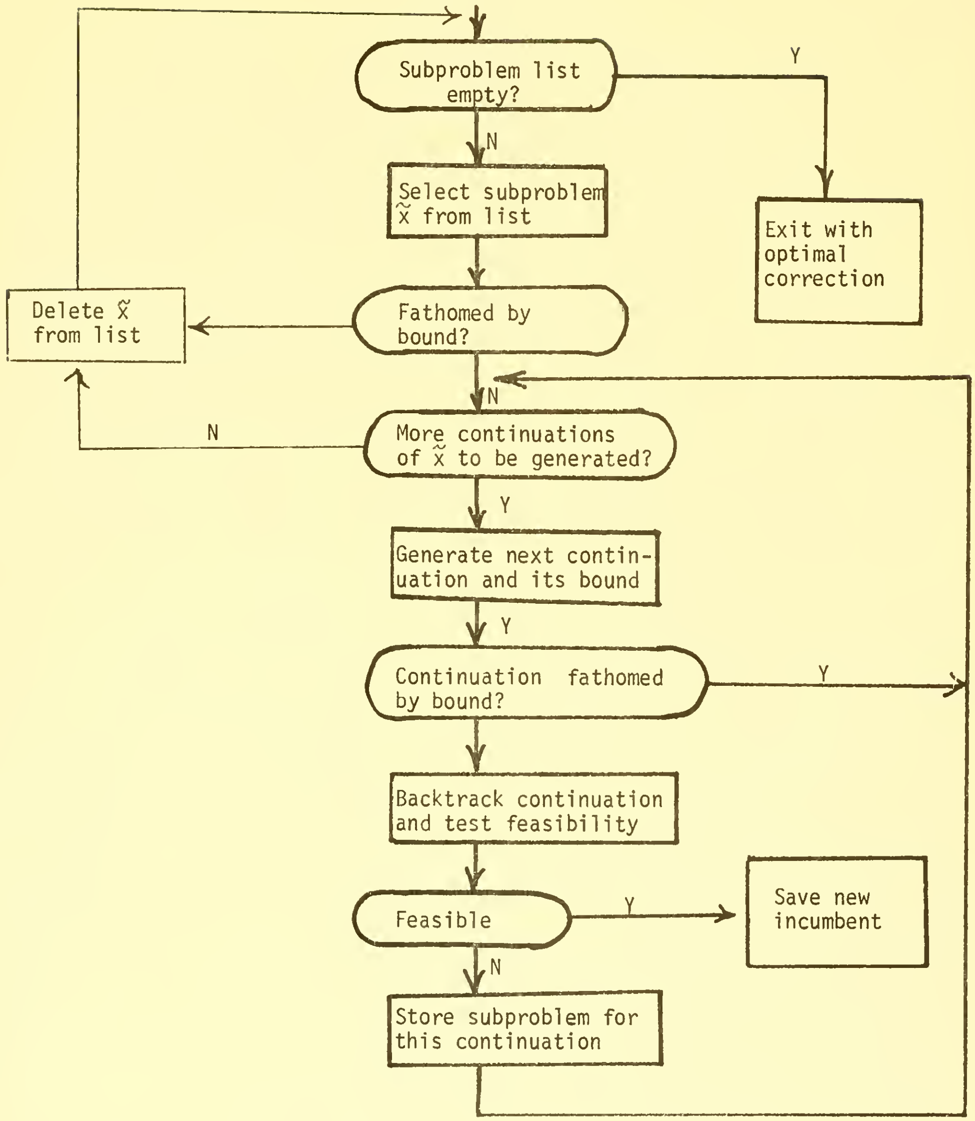
When all these descendants of $\tilde{x}$ have been created, tested, and saved if necessary, the subproblem $\tilde{x}$ is deleted. This is possible, of course, because

fathoming the descendants of $\tilde{x}$ is equivalent to fathoming $\tilde{x}$.

The particular search routine in our system is of the most straightforward kind. As can be seen from Table 4, it can generate and test solutions very rapidly. On the other hand, the real issue is how rapidly a problem can be solved. Finding a good incumbent early can have a dramatic effect on the efficiency of the search.[1] The current algorithm does not give sufficient emphasis to this goal. The theory for an improved search is presented in (4) and we hope to implement it along with some heuristics for managing the search.

[1]This is the reason we backtrack solutions as they are generated in the hopes of finding an incumbent early in the search.

FIGURE 2



Subproblem list empty?
Y
Exit with optimal correction
N
Select subproblem $\tilde{x}$ from list
Fathomed by bound?
Delete $\tilde{x}$ from list
N
More continuations of $\tilde{x}$ to be generated?
N
Y
Generate next continuation and its bound
Y
Continuation fathomed by bound?
Y
Backtrack continuation and test feasibility
Feasible
Y
Save new incumbent
N
Store subproblem for this continuation

Table 4. Search Times

	Rows	Non-basics	Number of Corrections	Time (sec)	Machine
1	12	104	99	0.77	1108
2	9	12	6769	7.14	360/85
3	36	36	7979	103.20	360/67
4	9	14	8463	19.63	360/67
5	12	104	11570	67.68	1108
6	9	14	12814	7.51	360/85
7	14	13	20692	10.51	360/85
8	12	104	169902	253.00	1108
9	313	170	668420	361.85	360/85

Concluding Remarks

We have tried to demonstrate in this paper that group theoretic methods are easy to derive and effective in solving IP problems. It is important to reemphasize that the algorithm IPA which uses these methods is perhaps the simplest one possible. After finding an optimal LP basis, IPA derives one group problem, either UGP or ZOG, solves it, and then if necessary, implicitly searches the set of all non-negative corrections where trial corrections and the extent of the search are derived from the previously solved group problem. In spite of its simplicity, IPA has performed quite well on a variety of real life IP problems as indicated in the figures in Table 1. We believe that future algorithms will be even more effective when we implement the dual methods of [4] for improving the bounds and generating strong cuts, and when we make dynamic use of the group optimization problems as discussed in [12]. A third important area of improvement and innovation is the use of data manipulation for controlling basis determinants by the methods discussed in [13]. We will conclude this paper with a discussion of the importance of the basis determinants in IP computation.

Consider the 313 row airline crew scheduling problem in Table 1. When the optimal LP basis obtained by MPS was pivoted into an identity matrix in preparation for solving the problem as an IP problem, the determinant equalled +1 or -1 for 311 pivots and then went to 24 on pivot 312 and 48 on pivot 313. Thus, the LP basis was almost unimodular and this made the problem easy to solve in spite of its large size. IPA found good solutions to a more difficult airline crew scheduling problem that had 259 rows and about 750 variables; it is not listed in Table 1 because we were not able to time its performance except to estimate the total solution time at about 15 minutes on an IBM 360/65. This problem had a determinant of 176 and

this solution was very fractional. It is interesting to note that we found alternative optimal bases for this problem with determinants of 8 and 103,000, so the selection of an optimal basis is important. Thiriez [22] also reports on the successful use of group theoretic methods in airline crew scheduling.

By contrast to the 313 row problem, consider the difficult 26 row zero-one problem (application unknown). Other codes had been tried on this problem with little success. The coefficient matrix of this problem consisted of 0's, 1's and 2's, and the determinant of the optimal LP basis was on the order of 5000. Another problem of this class with an additional less than or equal to constraint consisting of small positive integers with a right hand side of 235 had an optimal basis determinant of 155,000. We relaxed the constraint by the methods of [13] and hoped to satisfy it a fortiori during search. During one run we generated 37 solutions which were feasible on all of the constraints except the indicated one, but violated the relaxed one by approximately 10 to 20 units out of 235.

References

[1] Cabay, Stanley, "Exact Solution of Linear Equations", Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation, published by Association for Computing Machinery, Los Angeles, March 1971, 392-398.

[2] Dijkstra, E. W., "A Note on Two Problems in Connexion with Graphs", Numerische Mathematik, 1 (1959), 269-271.

[3] Dreyfus, S. E., "An Appraisal of Some Shortest-Path Algorithms", Operations Research, 17 (1969), 395-412.

[4] Fisher, M. L. and J. Shapiro, "Constructive Duality in Integer Programming", Operations Research Center Working Paper No. OR 008-72, M.I.T., April 1972.

[5] Geoffrion, A. M. and R. E. Marsten, "Integer Programming Algorithms: A Framework and State-of-the-Art Survey", Management Science, 18 (1972), 465-491.

[6] Glover, F., "Integer Programming Over a Finite Additive Group", SIAM J. on Control, 1 (1969), 213-231.

[7] Glover, F., "Faces of the Gomory Polyhedron", Chapter 17 in Integer and Nonlinear Programming, J. Abadie, editor, North-Holland Publishing Co., Amsterdam, 1970.

[8] Gomory, R. E., "An Algorithm for Integer Solutions to Linear Programs", Chapter 34 in Recent Advances in Mathematical Programming, Graves and Wolfe, editors, McGraw-Hill Book Co., Inc., New York, 1963, 269-302.

[9] Gomory, R. E., "On the Relation between Integer and Non-Integer Solutions to Linear Programs", Proc. Nat. Acad. Sci., 53 (1965), 260-265.

[10] Gomory, R. E., "Some Polyhedra Related to Combinatorial Problems", Journal of Linear Algebra and Applications, 2 (1969), 451-558.

[11] Gomory, R. E. and E. L. Johnson, "Some Continuous Functions Related to Corner Polyhedra", RC-3311 (February 1971), IBM, Yorktown Heights, N.Y.

[12] Gorry, G. A. and J. F. Shapiro, "An Adaptive Group Theoretic Algorithm for Integer Programming Problems", Management Science, 17 (1971), 285-306.

[13] Gorry, G. A., J. F. Shapiro and L. A. Wolsey, "Relaxation Methods for Pure and Mixed Integer Programming Problems", Management Science, 18 (1972), 229-239.

[14] Johnson, E. L., personal communication.

[15] Hu, T. C., "On the Asymptotic Integer Algorithm", Linear Algebra and Its Applications", 3 (1970), 279-294.

[16] Hu, T. C., Integer Programming and Network Flows, Addison-Wesley, Reading, 1969.

[17] Shapiro, J. F., "Dynamic Programming Algorithms for the Integer Programming Problem--I: The Integer Programming Problem Viewed as a Knapsack Type Problem", Operations Research, 16 (1968), 103-121.

[18] Shapiro, J. F., "Generalized Lagrange Multipliers in Integer Programming", Operations Research, 19 (1971), 68-76.

[19] Smith, D. A., "A Basis Algorithm for Finitely Generated Abelian Groups", Mathematical Algorithms, 1 (1969), 13-26.

[20] Smith, J. H. S., "On Systems of Linear Indeterminate Equations and Congruences", Philosophical Transactions, 151 (1861), 293-326.

[21] Smith, J. H. S., Collected Mathematical Papers, Vol. I, Clarendon Press, Oxford, 1894.

[22] Thiriez, H., "The Set Covering Problem: A Group Theoretic Approach", Revue Francaise d'Informatique et de Recherche Operationelle, V-3 (1971), 83-103.

[23] Tompkins, C. J. and V. E. Unger, "Group Theoretic Structures in the Fixed Charge Transportation Problem", paper presented at the Forty-first National Meeting of ORSA, New Orleans, April, 1972.

[24] Wolsey, L. A.,"Mixed Integer Programming: Discretization and the Group Theoretic Approach", Ph.D. thesis published as Technical Report No. 42 (June 1969), Operations Research Center, M.I.T.

[25] Wolsey, L. A., "Group-Theoretic Results in Mixed Integer Programming", Operations Research, 19 (1971), 1691-1697.

[26] Wolsey, L. A., "Extensions of the Group Theoretic Approach in Integer Programming", Management Science, 18 (1971), 74-83.