A Class of Hard Small 0–1 Programs *

Gérard Cornuéjols[1] and Milind Dawande[2]**

[1] Graduate School of Industrial Administration
Carnegie Mellon University, Pittsburgh, PA 15213, USA
[2] IBM, T. J. Watson Research Center
Yorktown Heights, NY 10598, USA

Abstract. In this paper, we consider a class of 0–1 programs which, although innocent looking, is a challenge for existing solution methods. Solving even small instances from this class is extremely difficult for conventional branch-and-bound or branch-and-cut algorithms. We also experimented with basis reduction algorithms and with dynamic programming without much success. The paper then examines the performance of two other methods: a group relaxation for 0,1 programs, and a sorting-based procedure following an idea of Wolsey. Although the results with these two methods are somewhat better than with the other four when it comes to checking feasibility, we offer this class of small 0,1 programs as a challenge to the research community. As of yet, instances from this class with as few as seven constraints and sixty 0–1 variables are unsolved.

1 Introduction

Goal programming [2] is a useful model when a decision maker wants to come "as close as possible" to satisfying a number of incompatible goals. It is frequently cited in introductory textbooks in management science and operations research. This model usually assumes that the variables are continuous but, of course, it can also arise when the decision variables must be 0,1 valued. As an example, consider the following market-sharing problem proposed by Williams [18]: A large company has two divisions D_1 and D_2. The company supplies retailers with several products. The goal is to allocate each retailer to either division D_1 or division D_2 so that D_1 controls 40% of the company's market for each product and D_2 the remaining 60% or, if such a perfect 40/60 split is not possible for all the products, to minimize the sum of percentage deviations from the 40/60 split. This problem can be modeled as the following integer program (IP):

* This work was supported in part by NSF grant DMI-9424348.
** Part of the work was done while this author was affiliated with Carnegie Mellon University.

R. E. Bixby, E. A. Boyd, and R. Z. Ríos-Mercado (Eds.): IPCO VI
LNCS 1412, pp. 284–293, 1998.

$$\begin{array}{lll} \text{Min} & \sum_{i=1}^{m} |s_i| & \\ \text{s.t.} & \sum_{j=1}^{n} a_{ij}x_j + s_i = b_i & i = 1, ..., m \\ & x_j & \in \{0, 1\} \text{ for } j = 1, ..., n \\ & s_i & \textit{free} \text{ for } i = 1, ..., m. \end{array}$$

where n is the number of retailers, m is the number of products, a_{ij} is the demand of retailer i for product j, and the right hand side vector b_i is determined from the desired market split among the two divisions D_1 and D_2. Note that the objective function of IP is not linear but it is straightforward to linearize it. This integer program also models the following basic *Feasibility Problem* (FP) in geometry:

Feasibility Problem: Given m hyperplanes in $\Re^n$, does there exist a point $x \in \{0, 1\}^n$ which lies on the intersection of these m hyperplanes?

If the optimum solution to IP is 0, the answer to FP is "yes", else the answer to FP is "no". Clearly, FP is NP-complete since for $m = 1$, FP is the subset-sum problem which is known to be NP-complete [6]. Problems of this form can be very difficult for existing IP solvers even for a relatively small number n of retailers and number m of products (e.g. $n = 50$, $m = 6$, and uniform integer demand between 0 and 99 for each product and retailer). More generally, with this choice of a_{ij}, asking for a 50/50 split and setting $n = 10(m - 1)$ produces a class of hard instances of 0–1 programs for existing IP solvers.

In this paper, we consider instances from the above class generated as follows: a_{ij} uniform integer between 0 and 99 $(= D - 1)$, $n = 10(m - 1)$ and $b_i = \lfloor \frac{1}{2} \sum_{j=1}^{n} a_{ij} \rfloor$ or, more generally, in the range $\lfloor \frac{1}{2}(-D + \sum_{j=1}^{n} a_{ij}) \rfloor$ to $\lfloor \frac{1}{2}(-D + \sum_{j=1}^{n} a_{ij}) \rfloor + D - 1$.

2 Available Approaches for Solving IP

In this section, we report on our computational experience with four different IP solvers available in the literature.

2.1 Branch and Bound

We found that even small instances of IP are *extremely* hard to solve using the conventional branch-and-bound approach. We offer the following explanation. For instances chosen as described above, there is often no 0–1 point in the intersection of the hyperplanes, that is the optimum solution to IP is strictly greater than 0, whereas the solution to the linear programming relaxation is 0, even after fixing numerous variables to 0 or to 1. Because the lower bound stays at 0, nodes of the branch-and-bound tree are not pruned by the lower bound until very deep into the enumeration tree. We illustrate this point in Table 1. We generated 5 instances of IP (each having 30 variables and 4 constraints) using the setup described above. We indicate the number of nodes enumerated to solve IP using CPLEX 4.0.3. For each of the 5 instances, the number of nodes enumerated by the branch-and-bound tree is greater than 2^{20}. Note that, in each

Table 1. Size of the branch-and-bound tree (5 instances).

Problem size	Number of nodes enumerated	Optimal solution
4×30	1224450	1.00
4×30	1364680	2.00
4×30	2223845	3.00
4×30	1922263	1.00
4×30	2415059	2.00

case, the solution to IP is strictly greater than 0. Instances with 40-50 variables take several weeks before running to completion.

Note: The problem class IP is related to the knapsack problems considered by Chvátal [3]. It is shown in [3] that these knapsack problems are hard to solve using branch-and-bound algorithms. However, the coefficients of the knapsack constraint are required to be very large ($U[1, 10^{\frac{n}{2}}]$). For the instances in the class IP that we consider, the coefficients a_{ij} are relatively small ($U[0, 99]$). By combining the constraints of IP with appropriate multipliers (e.g. multiplying constraint i by $(nD)^{i-1}$) and obtaining a *surrogate* constraint, we get an equivalent problem by choosing D large enough, say $D = 100$ in our case. The resulting class of knapsack instances is similar to that considered in [3].

2.2 Branch and Cut

The idea of branch-and-cut is to enhance the basic branch-and-bound approach by adding cuts in an attempt to improve the bounds. Here, we used MIPO, a branch-and-cut algorithm which uses lift-and-project cuts [1]. The computational results were even worse than with the basic branch-and-bound approach (see Table 2). This is not surprising since, in this case, the linear programming relaxation has a huge number of basic solutions with value 0. Each cutting plane cuts off the current basic solution but tends to leave many others with value 0. As we add more cuts, the linear programs become harder to solve and, overall, time is wasted in computing bounds that remain at value 0 in much of the enumeration tree.

2.3 Dynamic Programming

Using the surrogate constraint approach described above, we get a 0–1 knapsack problem which is equivalent to the original problem. Clearly, this technique is suitable for problems with only a few constraints. Dynamic programming algorithms can be used to solve this knapsack problem. Here, we use an implementation due to Martello and Toth [13] pages 108–109. The complexity of the algorithm is $O(\min(2^{n+1}, nc))$ where c is the right-hand-side of the knapsack

constraint. For the instances of size 3×20 and 4×30, we used the multiplier $(nD)^{i-1}$ with $D = 100$ for constraint i to obtain the surrogate constraint. For the larger instances, we faced space problems: We tried using smaller multipliers (e.g. $(nD^{'})^{i-1}$ with $D^{'} = 20$) but then the one-to-one correspondence between solutions of the original problem and that of the surrogate constraint is lost. So one has to investigate *all* the solutions to the surrogate constraint. Unfortunately, this requires that we store all the states of the dynamic program (the worst case bound for the number of states is 2^n). Hence, even when we decreased the multipliers, we faced space problems. See Table 2. We note that there are other more sophisticated dynamic programming based procedures [14] which could be used here. Hybrid techniques which use, for example, dynamic programming within branch-and-bound are also available [13].

2.4 Basis Reduction

For the basis reduction approach, we report results obtained using an implementation of the generalized basis reduction by Xin Wang [16][17]. It uses the ideas from Lenstra [11], Lovász and Scarf [12] and Cook, Rutherford, Scarf and Shallcross [4]. We consider the feasibility question FP here. Given the polytope $P = \{0 \leq x \leq 1 : Ax = b\}$, the basis reduction algorithm either finds a 0,1 point in P or generates a direction d in which P is "flat". That is, $\max\{dx - dy : x, y \in P\}$ is small. Without loss of generality, assume d has integer coordinates. For each integer t such that $\lceil \min\{dx : x \in P\} \rceil \leq t \leq \lfloor \max\{dx : x \in P\} \rfloor$ the feasibility question is recursively asked for $P \cap \{x : dx = t\}$. The dimension of each of the polytopes $P \cap \{x : dx = t\}$ is less than the dimension of P. Thus, applying the procedure to each polytope, a search tree is built which is at most n deep. A direction d in which the polytope is "flat" is found using a generalized basis reduction procedure [12][4].

2.5 Computational Experience

Table 2 contains our computational experience with the different approaches given in Section 2. We choose 4 settings for the problem size: $m \times n = 3 \times 20$, 4×30, 5×40 and 6×50. For each of these settings, we generated 5 instances as follows: a_{ij} random integers chosen uniformly in the range [0,99] and $b_i = \lfloor (\sum_{j=1} a_{ij})/2 \rfloor$. For branch-and-bound, we use the CPLEX 4.0.3 optimizer. For branch-and-cut, we use MIPO [1]. For dynamic programming, we use DPS [13]. For basis reduction, we use Wang's implementation [16]. Times reported refer to seconds on an HP720 Apollo desktop workstation with 64 megabytes of memory. None of the problems with 40 or 50 variables could be solved within a time limit of 15 hours.

Table 2. Computing times for various solution procedures.

Problem size ($m \times n$)	B & B CPLEX 4.0.3	B & C MIPO	DP DPS	Basis Red. Wang
3×20	11.76	213.62	0.93	300.42
3×20	13.04	125.47	1.28	209.79
3×20	13.16	208.76	0.94	212.03
3×20	13.64	154.71	1.31	277.41
3×20	11.21	190.81	1.11	197.37
4×30	1542.76	***	20.32	***
4×30	1706.84	***	20.37	***
4×30	2722.52	***	20.31	***
4×30	2408.84	***	18.43	***
4×30	2977.28	***	18.94	***
5×40	***	***	+++	***
5×40	***	***	+++	***
5×40	***	***	+++	***
5×40	***	***	+++	***
5×40	***	***	+++	***
6×50	***	***	+++	***
6×50	***	***	+++	***
6×50	***	***	+++	***
6×50	***	***	+++	***
6×50	***	***	+++	***

** Time limit (54000 seconds) exceeded.
+++ Space limit exceeded.

3 Two Other Approaches

In this section, we introduce two other approaches to FP.

3.1 The Group Relaxation

The feasible set of FP is

$$S = \{x \in \{0,1\}^n : \ Ax = b\}$$

where (A, b) is an integral $m \times (n+1)$ matrix. We relax S to the following *group problem* (GP).

$$S_\delta = \{x \in \{0,1\}^n : \ Ax \equiv b \ (\text{mod } \delta)\}$$

where $\delta \in Z_+^m$. GP is interesting because (i) In general, GP is easier to solve than IP [15] and (ii) Every solution of FP satisfies GP. The feasible solutions to GP can be represented as s-t paths in a directed acyclic layered network G. The digraph G has a layer corresponding to each variable $x_j, j \in N$, a source node s

and a sink node t. The layer corresponding to variable x_j has $\delta_1 \times \delta_2 ... \times \delta_m$ nodes. Node j^k where $k = (k_1, ..., k_m)$, $k_i = 0, \ldots, \delta_i - 1$ for $i = 1, \ldots, m$, can be reached from the source node s if variables $x_1, x_2, ..., x_j$ can be assigned values 0 or 1 such that $\sum_{\ell=1}^{j} a_{i\ell} x_\ell \equiv k_i \pmod{\delta_i}$, $i = 1, 2, ..., m$. When this is the case, node j^k has two outgoing arcs $(j^k, (j+1)^k)$ and $(j^k, (j+1)^{k'})$, where $k'_i \equiv k_i + a_{i,j+1} (\mathrm{mod}\ \delta_i)$, corresponding to setting variable x_{j+1} to 0 or to 1. The only arc to t is from node $n^{b \pmod{\delta}}$. So, the digraph G has $N = 2 + n \times \delta_1 ... \times \delta_m$ nodes and at most twice as many arcs.

Example 2: Consider the set

$$S = \{x \in \{0,1\}^4 : 3x_1 + 2x_2 + 3x_3 + 4x_4 = 5$$
$$6x_1 + 7x_2 + 3x_3 + 3x_4 = 10\}$$

Corresponding to the choice $\delta_i = 2$, $i = 1, 2$, we have the following group relaxation

$$S_{22} = \{x \in \{0,1\}^4 : 1x_1 + 0x_2 + 1x_3 + 0x_4 \equiv 1 \pmod 2$$
$$0x_1 + 1x_2 + 1x_3 + 1x_4 \equiv 0 \pmod 2)\}$$

Figure 1 gives the layered digraph G for this relaxation.

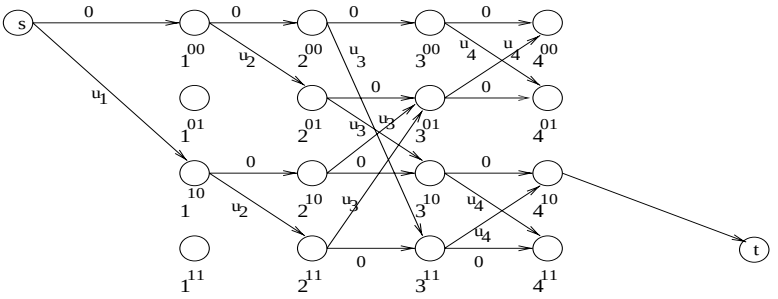

Fig. 1. The layered digraph, G, for Example 2.

Every s-t path in G represents a feasible 0–1 solution to S_{22}. What is more important is that every feasible solution to FP is also a s-t path in G. But this relationship is not reversible. That is, an s-t path in G may not be feasible for IP. Also, G may contain several arcs which do not belong to any s-t path. Such arcs can be easily identified and discarded as follows: Among the outgoing arcs from nodes in layer $n-1$, we only keep those arcs which reach the node $n^{b \pmod{\delta}}$ and delete all other arcs. This may introduce paths which terminate at layer $n-1$. Hence, going through the nodes in layer $n-2$, we discard all outgoing arcs on paths which terminate at layer $n-1$, and so on. It can be easily seen that performing a "backward sweep" in G in this manner, in time $\mathrm{O}(N)$, we get a new graph G' which consists of only the arcs in solutions to the group relaxation. For the graph G in Figure 1, the graph G' is shown in Figure 2.

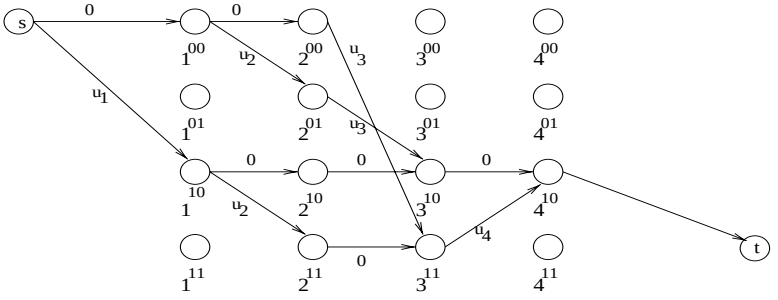

Fig. 2. The solution digraph, $G^{'}$, for Example 2.

For each s-t path in G', we check whether it corresponds to a feasible solution of FP. Thus, a depth-first-search backtracking algorithm solves FP in time $O(pmn)$ where p is the number of s-t paths in G'. Two issues are crucial with regard to the complexity of this approach: the size of p and the size of the digraph $G^{'}$. A simple estimate shows that the expected value of p is $\frac{2^n}{\delta_1 \times \ldots \times \delta_m}$ when the data a_{ij} are uniformly and independently distributed between 0 and $D-1$, the δ_i's divide D and the b_i's are uniformly and independently distributed in the range $\lfloor \frac{1}{2}(-D + \sum_{j=1}^{n} a_{ij}) \rfloor$ to $\lfloor \frac{1}{2}(-D + \sum_{j=1}^{n} a_{ij}) \rfloor + D - 1$. On the other hand, the size of $G^{'}$ is of the order $n \times \delta_1 ... \times \delta_m$. The two issues, namely the size of the digraph $G^{'}$ and the number of solutions to the group relaxation, are complementary to each other. As the size of $G^{'}$ increases, the number of solutions to the group relaxation decreases. The best choice is when these two sizes are equal, that is $\delta_1 \times \delta_2 \times \ldots \delta_m \approx \frac{2^{\frac{n}{2}}}{\sqrt{n}}$. Then, under the above probabilistic assumptions, the expected complexity of the group relaxation approach is $O(\sqrt{n} 2^{\frac{n}{2}})$.

3.2 An $O(n2^{(n/2)})$ Sorting-Based Procedure

Laurence Wolsey [21] suggested a solution approach based on sorting. We first describe this procedure for the subset sum feasibility problem (SSFP).

SSFP: Given an integer n, an integral n-vector $a = (a_1, ..., a_n)$ and an integer b, is $\{x \in \{0,1\}^n : \sum_{j=1}^{n} a_j x_j = b\} \neq \phi$?

The procedure SSP(n,a,b) described below decomposes the problem into two subproblems each of size $\frac{n}{2}$. It then sorts the $2^{\frac{n}{2}}$ subset sums for both the subproblems and traverses the two lists containing these subset sums in opposite directions to find two values that add up to b. WLOG, we assume n is even.

SSP(n,a,b)

1. Let $p = \frac{n}{2}, v^1 = \{a_1, ..., a_p\}$ and $v^2 = \{a_{p+1}, ..., a_n\}$. Compute SS^1 and SS^2, the arrays of subset sums of the power sets of v^1 and v^2 respectively.
2. Sort SS^1 and SS^2 in ascending order.

3. Let $k = 2^p$, $i = 1$ and $j = k$.
   do while $((i \leq k)$ OR $(j \geq 1))$ {
   If $(SS^1[i] + SS^2[j] = b)$ then **quit**. (Answer to SSFP is "yes")
   else if $(SS^1[i] + SS^2[j] < b)$ then set $i = i + 1$
   else set $j = j - 1$ }.
4. Answer to SSFP is "No".

The complexity of this procedure is dominated by the sorting step and therefore it is $O(n2^{\frac{n}{2}})$. This procedure can be extended to answer FP as follows: For $i = 1, ..., m$, multiply the i^{th} constraint by $(nD)^{i-1}$ and add all the constraints to obtain a single *surrogate* constraint

$$\sum_{j=1}^{n}\sum_{i=1}^{m}(nD)^{i-1}a_{ij}x_j = \sum_{i=1}^{m}(nD)^{i-1}b_i$$

Let $a = (\sum_{i=1}^{m}(nD)^{i-1}a_{i1}, ..., \sum_{i=1}^{m}(nD)^{i-1}a_{in})$, $b = \sum_{i=1}^{m}(nD)^{i-1}b_i$. Call SSP($n$,$a$,$b$).

Note: As for dynamic programming, this technique is suitable only for problems with a few constraints. If $(nD)^{m-1}$ is too large, a smaller number can be used but then the one-to-one correspondence between the solutions of FP and the solutions of the surrogate constraint is lost. In this case note that, if for some i and j we have $SS^1[i] + SS^2[j] = b$, the corresponding 0–1 solution may not satisfy FP. So, in order to solve FP, we need to find all the pairs i, j such that $SS^1[i] + SS^2[j] = b$.

3.3 Computational Experience

See Table 3. For the group relaxation, we use $\delta_i = 8$, for all $i = 1, ..., m$, for the 5 instances of size 3×20 and $\delta_i = 16$, for all $i = 1, ..., m$, for the remaining instances. Times reported refer to seconds on an HP720 Apollo desktop workstation with 64 megabytes of memory. The sorting-based procedure dominates the group relaxation for instances up to 40 variables. The group relaxation could solve all the instances but is very expensive for larger problems. Within each problem setting, there is very little difference in the amount of time taken by the group relaxation. This is not surprising since, within each problem setting, the number of solutions to the group relaxation is about the same. A similar observation holds for the subset sum approach as well as dynamic programming.

4 Conclusions

In this paper, we consider a class of 0,1 programs with n variables, where n is a multiple of 10. As noted in the previous section, although the group relaxation is able to solve problem instances with up to 50 variables, its running time increases rapidly. Its space complexity and expected time complexity can be estimated to be $O(\sqrt{n}2^{\frac{n}{2}})$. It is an open question to find an algorithm with expected time

Table 3. Computing time comparisons.

Problem size ($m \times n$)	B & B	B & C	DP	Basis	Group	Subset Sort
3×20	11.76	213.62	0.93	300.42	0.13	0.01
3×20	13.04	125.47	1.28	209.79	0.11	0.01
3×20	13.16	208.76	0.94	212.03	0.11	0.01
3×20	13.64	154.71	1.31	277.41	0.12	0.03
3×20	11.21	190.81	1.11	197.37	0.12	0.03
4×30	1542.76	***	20.32	***	18.06	0.99
4×30	1706.84	***	20.37	***	17.83	1.03
4×30	2722.52	***	20.31	***	17.92	0.98
4×30	2408.84	***	18.43	***	17.93	1.00
4×30	2977.28	***	18.94	***	18.04	1.01
5×40	***	***	+++	***	1556.43	46.10
5×40	***	***	+++	***	1562.66	46.18
5×40	***	***	+++	***	1604.02	45.61
5×40	***	***	+++	***	1548.55	46.20
5×40	***	***	+++	***	1606.24	45.51
6×50	***	***	+++	***	26425.01	+++
6×50	***	***	+++	***	26591.28	+++
6×50	***	***	+++	***	26454.30	+++
6×50	***	***	+++	***	27379.04	+++
6×50	***	***	+++	***	27316.98	+++

** Time limit (54000 seconds) exceeded.
+++ Space limit exceeded.

complexity better than $O(2^{\frac{n}{2}})$. Our computational experience indicates that the standard approaches to integer programming are not well suited for this class of problems. As such, we would like to present these small-sized 0–1 integer programs as a challenge for the research community and we hope that they may lead to new algorithmic ideas.

References

1. E. Balas, S. Ceria, and G. Cornuéjols. Mixed 0–1 programming by lift-and-project in a branch-and-cut framework. *Management Science*, 42:1229–1246, 1996.
2. A. Charnes and W. W. Cooper. *Management Models and Industrial Applications of Linear Programming*. Wiley, New York, 1961.
3. V. Chvátal. Hard knapsack problems. *Operations Research*, 28:1402–1411, 1980.
4. W. Cook, T. Rutherford, H. E. Scarf, and D. Shallcross. An implementation of the generalized basis reduction algorithm for integer programming. *ORSA Journal of Computing*, 3:206–212, 1993.
5. H. P. Crowder and E. L. Johnson. Use of cyclic group methods in branch-and-bound. In T. C. Hu and S. M. Robinson, editors, *Mathematical Programming*, pages 213–216. Academic Press, 1973.

6. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.
7. R. E. Gomory. On the relation between integer and non-integer solutions to linear programs. *Proceedings of the National Academy of Sciences, Vol. 53*, pages 260–265, 1965.
8. G. A. Gorry, W. D. Northup, and J. F. Shapiro. Computational experience with a group theoretic integer programming algorithm. *Mathematical Programming*, 4:171–192, 1973.
9. G. A. Gorry and J. F. Shapiro. An adaptive group theoretic algorithm for integer programming problems. *Management Science*, 7:285–306, 1971.
10. G. A. Gorry, J. F. Shapiro, and L. A. Wolsey. Relaxation methods for pure and mixed integer programming problems. *Management Science*, 18:229–239, 1972.
11. H. W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–547, 1983.
12. L. Lovász and H. Scarf. The generalized basis reduction algorithm. *Mathematics of Operations Research*, 17:751–763, 1992.
13. S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, Chichester, U.K, 1990.
14. S. Martello and P. Toth. A mixture of dynamic programming and branch-and-bound for the subset sum problem. *Management Science* 30:765–771, 1984.
15. M. Minoux. *Mathematical Programming: Theory and Algorithms*. Wiley, New York, 1986.
16. X. Wang. *A New Implementation of the Generalized Basis Reduction Algorithm for Convex Integer Programming*, PhD thesis, Yale University, New Haven, CT, 1997. In preparation.
17. X. Wang. Private communication, 1997.
18. H. P. Williams. *Model Building in Mathematical Programming*. Wiley, 1978.
19. L. A. Wolsey. Group-theoretic results in mixed integer programming. *Operations Research*, 19:1691–1697, 1971.
20. L. A. Wolsey. Extensions of the group theoretic approach in integer programming. *Management Science*, 18:74–83, 1971.
21. L. A. Wolsey. Private communication, 1997.