A Lagrangean Algorithm for the Multiple Choice Integer Program

Technical Report 82-14

James C. Bean

Department of Industrial and Operations Engineering
The University of Michigan
Ann Arbor, Michigan 48109

September, 1982,

Revised: November, 1982

Revised: May, 1983


THE UNIVERSITY OF MICHIGAN
LIBRARIES
THE UNIVERSITY OF MICHIGAN
1817

A Lagrangean Algorithm for the Multiple Choice Integer Program

James C. Bean

Department of Industrial and Operations Engineering
The University of Michigan
Ann Arbor, Michigan 48109

ABSTRACT

The Multiple Choice Integer program is a binary integer linear program in which the variables have been partitioned into generalized upper bounding sets (or special ordered sets). This problem is solved using a branch-and-bound scheme designed to take advantage of this structure. The basic procedure is enhanced by solving a Lagrangean problem as an approximation. Further, a variable reduction technique is used to eliminate many of the variables prior to problem solution. Computational experience is promising for the randomly generated problems, scheduling problems and budgeting problems tested.

This work was supported in part by ONR contract N00014-76-C0418 and NSF grant MCS76-81259.

A Lagrangean Algorithm for the Multiple Choice Integer Program

1. Introduction

The Multiple Choice Integer Program (MCIP) can be thought of as a generalization of the classical assignment problem. In the assignment problem we have m workers and m tasks. Each task must be assigned to a different worker to minimize the cost of completing all tasks.

This problem was relaxed by Ross and Soland (1973) to the generalized assignment problem. In this case some workers may be able to complete more than one task, up to some capacity. For example, one job that takes five hours and one that takes three hours can both be accomplished by the same worker in one day.

In the MCIP we still require that each task be assigned to some worker, but now allow any other constraints that can be expressed as linear inequalities or equalities. This makes the MCIP a generalization of the generalized assignment problem.

Mathematically, the MCIP is a type of binary integer linear program in which the set of variables has been partitioned. For any feasible solution, precisely one variable from each partitioning set must have the value 1 and all other variables must have the the value 0. Any additional constraints that can be expressed as linear inequalities (or equalities) are also acceptable.

A formal statement of the problem being considered is:

$$\text{Minimize} \quad \sum_{i=1}^{m} c_{ij} x_{ij}$$

$$\text{Subject to:} \quad b + Ax = 0 \qquad \text{(MCIP)}$$

$$\sum_{j=1}^{n_i} x_{ij} = 1, \quad i = 1, 2, \ldots, m$$

$$x_{ij} \in \{0,1\}$$

where m is the number of partitioning sets (referred to henceforth as generalized upper bounding sets or GUB-sets), n_i is the number of variables in the i^{th} GUB-set, c_{ij} is the cost associated with selecting variable x_{ij}, b is a constant vector in R^K, K is the number of general constraints, A is a K by (Σn_i) coefficient matrix and $\{x_{ij}\}$ are the binary variables. Interpreting this in the earlier problem context, each GUB-set is the set of workers to which a particular task could possibly be assigned.

Problems that can be formulated in this manner include many in the fields of scheduling, facility location, assembly line balancing, project selection, menu planning, catalogue space planning, school time tabling, budgeting and portfolio analysis.

Early attacks on problems similar to the MCIP came from Healy (1964), Beale and Tomlin (1970) and Ross and Soland (1973). More recently, direct atacks on this problem have begun: Mevert and Suhl (1977), Bean (1982), Martin (1980), Sweeney and Murphy (1981) and Martin, Sweeney and Doherty (1981). Most of these have been branch-and-bound algorithms and, henceforth, the

terminology of branch-and-bound algorithms will be assumed. Any reader not familiar with the terminology is referred to Geoffrion and Marsten (1972).

Following this introduction, the paper is divided into four sections. In Section 2 a branching strategy special to this problem is developed. The resultant branch-and-bound algorithm is described with its theoretical computational characteristics. Section 3 describes a Lagrangean alteration of the problem that greatly enhances computation. This is followed by a discussion of a variable reduction technique that eliminates most variables, a priori, from consideration in the branch-and-bound procedure. Finally, section five presents computational results on two classes of real problems and a set of randomly generated problems.

2. Branch-and Bound Procedure

We will begin by describing a branch-and-bound methodology, referred to as the basic procedure, that strongly exploits the structure of this problem.

An n variable binary program in general has an enumeration tree containing 2^n branches. If the variables are separated into GUB-sets (or special ordered sets) such that there are m sets, the i^{th} containing n_i variables, we have $n = \Sigma n_i$ variables. Of the branches mentioned above, only Πn_i are feasible in the multiple choice constraints defined by this partitioning. If the GUB-feasible nodes of the general enumeration tree are connected, we get a much smaller enumeration tree.

The algorithm presented here enumerates the smaller tree

leading to savings from two sources. The primary savings result from the fact that there are many fewer branches to enumerate. Beyond this, there are now fewer constraints to consider since the GUB constraints are handled implicitly. No potential solution need be tested for GUB-feasibility since only GUB-feasible solutions are considered. When a partial solution is augmented in the algorithm, it is augmented by a GUB-set of variables. That is, some unfixed GUB-set is chosen and all the variables in that GUB-set are fixed; one at 1 and the others at 0. Progress through this more complicated tree will be recorded by storing m^2 integer values.

Before introducing these values some preliminary notation is necessary. Since we know that in any feasible solution there is a single 1 in each GUB-set, it is never necessary to record any feasible solution in its entirety. It is sufficient to simply record which element of each GUB-set has the value 1 in that solution. This requires storing m rather than $\sum n_i$ integer values. This can be accomplished by storing the index, in each GUB-set, of the variable set to 1. Such a vector of indices will be referred to as a position.

Progress through the tree for this algorithm can be recorded by storing m solutions in this manner. These positions will serve as addresses to which the algorithm will backtrack or jumptrack. These stored positions will be referred to as bases and will be indexed 1, 2,...,m. The notation b^j and term j-base will be used to indicate specific bases. The bases record the nodes of the tree passed through while branching to the current

solution.

2.1 Branching

The branching process of this algorithm will be referred to as stepping. A step involves changing the index of the 1 in some GUB-set. If a step is carried out on a particular GUB-set that currently has a 1 for its third variable, the new position will have a 1 for the fourth variable of that GUB-set. The values in all other GUB-sets will remain the same. This implies that branching rules are imbedded in the indexing of the variables in each GUB-set (as well as rules for augmentation of partial solutions). Steps are indexed by j = 1, 2,...,m. A j-step is a step carried out when the current partial solution contains j GUB-sets of variables.

When augmenting a partial solution, the free GUB-set of variables to be fixed is chosen as follows. Let j be the number of GUB-sets of variables currently fixed in the partial solution. Recall that the j-base is a GUB-feasible binary vector corresponding to some node in the enumeration tree. It is the completion of the partial solution at that node, which the algorithm investigated last. The binary vector represented in the j-base has precisely m 1's, one in each GUB-set. Consider the objective coefficients for each of these 1's. Label the GUB-sets according to these coefficients, lowest first. It can be proven that the j lowest of these correspond to those GUB-sets already in the current partial solution (see Bean (1982)). To augment the partial solution, choose the GUB-set associated with the next smallest objective value ($(j+1)^{st}$ smallest overall) as the next GUB-set to be fixed. The variable indexed by the j-base

for this GUBset is fixed at 1 and all other variables at 0. All ties are broken by the least index rule.

Having discussed the primary attributes of this algorithm we will now state it formally.

2.2. The Algorithm.

Step 0: (Initialize) Order the variables in each GUB-set by cost, lowest first. Set all m bases at the solution consisting of the lowest cost element in each GUB-set (set each base to a vector of ones). Set the incumbent at ϕ and the incumbent objective value at $+\infty$ (assuming minimation). Set j = 0, indicating that the partial solution is empty. Go to Step 2.

Step 1: (Cost fathom) Is the total cost associated with the j-base greater than or equal to the incumbent value? If so, set j = j - 1 and go to Step 4. If not, go to Step 2.

Step 2: (Feasibility fathom) Is there an easily determined optimal and feasible completion to the partial solution with better objective value than the incumbent? If so, go to Step 3. If not, go to step 4. If there is not enough information to decide, go to Step 5.

Step 3. (Record New Incumbent) Record this new best solution as the incumbent. Set j = j - 1 and go to Step 4.

Step 4: (Jumptracking) If $j \leq 0$ stop, the incumbent is optimal. Otherwise order the costs associated with the j-base and label the GUB-sets 1 through m according to this ordering (lowest = 1, highest = m, break ties by

the least index rule). If the position of the 1 in the GUB-set labeled j is at the end ($=n_j$), set j = j - 1 and go to Step 4. Otherwise, increment the position of the 1 in the GUB-set labeled j. Set the j-base and all bases of order higher than j to this new position. Relabel the GUB-sets according to the costs associated with the new base. Go to Step 1.

Step 5: (Augmentation) Set j = j + 1. Go to Step 2.

It is important to note here that at any point in the algorithm the partial solution is determined as the variables in those GUB-sets labeled one through j in the current labeling scheme. The partial solution consists of a zero for all variables in these GUB-sets except those indexed by the current j-base.

Proof that this basic procedure finds an optimal solution to the MCIP in finite time, as well as an illustrative example, can be found in Bean (1982).

2.3 Worst Case Performance.

We will now look at how the tree enumerated here grows in number of variables and configuration. This algorithm is complex enough that it is very difficult to establish bounds on performance tighter than the greatest number of subproblems that could be evaluated. In fact, if Step 2 is restricted to its simplest version, the algorithm can be shown to investigate every possible subproblem for some examples.

In the case of a general binary problem with n variables, an enumerative algorithm must consider, implicitly or explicitly, 2^n

possible solutions. In this general problem the variables could be partitioned into m sets, containing n_i variables for i = 1, 2,..., m. Then $n = \sum_{i=1}^{m} n_i$ and $2^n = \prod_{i=1}^{m} 2^{n_i}$. In the MCIP, where such a partitioning is natural, the number of possible combinations is $\prod_{i=1}^{m} n_i$. Clearly, for any positive integers m and n_i, i = 1, 2,...,m, it is true that

$$\prod_{i=1}^{m} n_i \ll \prod_{i=1}^{m} 2^{n_i}.$$

In general binary trees the maximum number of solutions investigated is a function only of the number of variables. In the multiple choice case the number of branches is a function of both the number of variables and the configuration of the GUB partitioning. Following is a discussion of three configurations: the worst for computation, an average case, and the best for computation.

Given n variables, the worst possible configuration is n/2 GUB-sets each with 2 elements. In this case the upper bound on investigated solutions is

$$\prod_{i=1}^{n/2} 2 = 2^{n/2} \tilde{=} 1.4^n.$$

Through this is clearly better than the 2^n upper bound on the general problem, it is exponential and greatly limits the size of problems that are solvable.

An average case configuration is "square," that is,

$$m = n_1 = n_2 = \ldots = n_m.$$

The classical assignment problem is an example of a "square" problem.

The upper bound on investigated solutions for this

configuration is

$$\prod_{i=1}^{m} m = m^m.$$

Since the number of variables is $n = m^2$, the upper bound on the number of partial solutions explicitly investigated is $\sqrt{n}^{\sqrt{n}}$. It can easily be shown that this is better than exponential, but not polynomial.

The last configuration to be analyzed is that having a fixed number of GUB-sets. Let m be held constant as the number of variables is increased. That is, all new variables are added to existing GUB-sets rather than introducing new GUB-sets. The number of elements in any particular GUB-set is not greater than the total number of variables so

$$\prod_{i=1}^{m} n_i \leq \prod_{i=1}^{m} n = n^m.$$

Hence, the upper bound for this configuration is polynomial with order, at most, equal to the number of GUB-sets.

3. Lagrangean Alteration

As noted, the basic version of the algorithm first selects variables with low costs. This strategy is very effective so long as the variables of low cost contain a feasible solution. If all feasible solutions involve higher cost variables, the basic algorithm uses many steps to reach them.

We would like to reorder in such a way that variables low in cost, but also contributing significantly to a feasible solution, are considered early in the process. This can be done effectively by a Lagrangean alteration of the costs. Let λ be

any non-negative vector of appropriate dimension.

Consider the problems

$$\begin{aligned} &\min \quad cx - \lambda(b + Ax) \\ &\text{Subject to:} \quad b + Ax \geq 0 \\ (P_\lambda) \qquad &\sum_{j=1}^{n_i} x_{ij} = 1, \quad i = 1, 2, \ldots, m \\ &x_{ij} \in \{0,1\} \end{aligned}$$

$$\begin{aligned} &\min \quad cx - \lambda(b + Ax) \\ &\text{Subject to:} \quad \sum_{j=1}^{n_i} x_{ij} = 1, \quad i = 1, 2, \ldots, m \\ (PR_\lambda) \qquad &x_{ij} \in \{0,1\} \end{aligned}$$

and

$$\begin{aligned} &\min \quad cx - \lambda(b + Ax) \\ (\overline{PR}_\lambda) \quad &\text{Subject to:} \quad \sum_{j=1}^{n_i} x_{ij} = 1 \\ &0 \leq x_{ij} \leq 1 \end{aligned}$$

Let $v(\cdot)$ be the optimal objective value to problem $(\cdot)$. It should be clear that;

$$v(\overline{PR}_\lambda) \leq v(PR_\lambda) \leq v(P_\lambda) \leq v(P).$$

Further, the relaxation PR_λ has the integrality property (see

Geoffrion (1974) and Fisher (1981)) so that $v(\overline{PR}_\lambda) = v(PR_\lambda)$.

By solving the problem P_λ we guarantee a feasible solution to P and get upper and lower bounds on v(P). If $x^*(\cdot)$ is an optimal solution to $(\cdot)$, then

$$v(P_\lambda) = cx^*(P_\lambda) - (b + Ax^*(P_\lambda)) \leq v(P) \leq cx^*(P_\lambda).$$

If $(b + Ax^*(P_\lambda)) = 0$, then we have solved P.

For an appropriate $\lambda \geq 0$, altering the costs in this manner reorders the costs just as we desired. The cost of a variable now reflects both its actual cost and contribution to feasibility in the general constraints. The basic procedure can now be used to solve P .

In cases of Lagrangean relaxation we choose λ to get as close as possible to problem P. Typically, we can find λ as the $\operatorname{argmax}_\lambda v(PR_\lambda)$. In a problem with the integrality property such as this, we know that this is the optimal dual vector from the L.P. Relaxation of P, $\overline{P}$ (see Fisher (1981)). We cannot use this approach when only altering costs as in P since $\max_\lambda v(P_\lambda) = v(P)$ with $\lambda \geq 0$. This solution is useless since it does not accomplish the reordering we seek. Hence, we will use an optimal dual vector from P here, i.e., from now on we will choose $\lambda = \bar{\lambda}$, an optimal dual vector from P.

By solving P_λ rather than P we are not assured of finding an optimal solution of P. If the Lagrangean term $\lambda(b + Ax) = 0$ at the optimal solution to P_λ then we have found the optimal to P also. However, when the constraints are inequalities, as is generally the case, this complementarity condition is not guaranteed to hold (see Fisher 1981)). In our experience, however, an optimal solution has been found in most problems.

The solution of P_λ provides an upper bound on error. If this error is not satisfactory, $cx^*(P_\lambda)$ can be used as a good initial incumbent objective value and the basic algorithm used to solve P.

If the general constraints are of the form $b + Ax = 0$, then the Lagrangean term $\lambda(b + Ax) = 0$ for any feasible solution. In this case, $x^*(P_\lambda)$ <u>is</u> guaranteed to be optimal for P. Such is the case in the scheduling problems tested below.

4. <u>Variable Reduction</u>

A method for reducing the number of variables that must be considered in solving the MCIP has recently been presented in Sweeney and Murphy (1981) and Martin, Sweeney and Doherty (1981). This method begins by ordering the costs $cx - \lambda(b + Ax)$ in each GUB-set just as is done in Step 0 os the basic algorithm when solving P_λ. Assume also that these costs have been modified by subtracting the smallest cost in each GUB-set from each cost in the GUB-set. Since some element of that GUB-set must be chosen, this does not alter the optimal solution. It simply offsets the optimal objective function value by a constant. (This is similar to "cost reduction" used in solving assignment problems). Now the lowest cost in each GUB-set is 0. These modified costs will be referred to as the <u>reduced costs</u>. Note also that the sum of the m costs subtracted from the costs in the respective GUB-sets is $v(\overline{PR_\lambda}) = v(PR_\lambda)$.

The procedure continues by accepting a user specified value $\delta > 0$. The problem is reduced by eliminating all variables with reduced costs greater than δ. Let D_i be the set of all variables

in GUB-set i with cost not greater than δ and x_δ be the new vector of variables. Then we have a new problem.

$$\min \quad cx_\delta - \lambda(b + Ax_\delta)$$

$$\text{Subject to} \quad b + Ax_\delta \geq 0$$

$$(MP_{\delta\lambda}) \qquad \sum_{j \in D_i} x_{ij} = 1, \quad i = 1, 2, \ldots, m$$

$$x_{ij} \in \{0,1\}$$

We will also be interested in the problem

$$\min \quad cx_\delta$$

$$\text{Subject to:} \quad b + Ax_\delta \geq 0$$

$$(MP_\delta) \qquad \sum_{j \in D_i} x_{ij} = 1, \quad i = 1, 2, \ldots, m$$

$$x_{ij} \in \{0, 1\}$$

Sweeney and Murphy (1981) prove the following theorem.

Theorem 1: If $v(MP_\delta) \leq v(PR_\lambda) + \delta$, then

$v(MP_\delta) = v(P)$.

They go on to prove

Theorem 2: If θ is is the value of a feasible solution to MP_δ and $LB(MP_\delta)$ is a lower bound on $v(MP_\delta)$, then

$$\theta - v(P) \leq \theta - \min\,(v(PR_\lambda) + \delta,\ LB(MP_\delta)).$$

We have $cx^*(MP_{\delta\lambda})$ as θ and $v(MP_{\delta\lambda})$ as $LB(MP_\delta)$ so

$$cx^*(MP_{\delta\lambda}) - v(P) \leq cx^*(MP_{\delta\lambda}) - \min\,[v(PR_\lambda) + \delta,\ v(MP_{\delta\lambda})].$$

The efficiency of this technique depends greatly on the value of δ chosen. Too small a δ causes the optimal solution to be discarded, too large a δ includes so many variables that the problem is no easier to solve. We can establish an upper bound for δ. Recall that we are dealing with reduced costs. Denote these costs $\overline{c_{ij}}$. Let UB(P) be any upper bound on v(P).

Corollary 3: If $\overline{c_{ij}} < UB(P) - v(PR_{\lambda}^{-})$, then $x_{ij} = 0$ in any optimal solution.

Proof: Similar to Theorem 1.

As a result, if we set $\delta = v(P) - v(PR_{\lambda}^{-})$, the duality gap in this problem, we know that no variables in any optimal solution will be eliminated. Before choosing δ, however, we do not know v(P). If by a heuristic or sampling procedure we can get a reasonable upper bound on v(P) it can be used to choose a risk-free δ. One use of this result is that at each new incumbent the improved objective value gives a better upper bound. This leads to the opportunity to fix all variables with $c_{ij} > UB - v(PR_{\lambda}^{-})$ at 0 for the remainder of the algorithm.

General Procedure

We will now state the general procedure:

1) Solve $\bar{P}$ to find $\bar{\lambda}$. (if the optimal solution is integral, stop).
2) Using some δ, ($\delta \leq UB(P) - v(P)$ if possible), reduce the number of variables in P.
3) Solve $MP_{\delta\lambda}$ by the basic procedure.
4) If $cx^* - \min(v(PR_{\lambda}^{-}) + \delta, v(MP_{\delta\lambda}))$ is sufficiently small, stop. Otherwise, use $\delta = v(MP_{\delta\lambda}) - v(PR_{\lambda}^{-})$ and

solve MP_δ by the basic procedure to get the optimal solution.

This algorithm can be run as a heuristic or as an optimal algorithm. If "sufficiently small" in Step 4 is zero, then the algorithm is optimal. Otherwise, the difference expression in Step 4 is an upper bound on error.

5. Computational Results

This algorithm has been tested on three classes of problems: retail budgeting, league scheduling and randomly generated problems. The budgeting and scheduling problems were derived from real data while the randomly generated problems allow comparison with other solution techniques. In each case the general procedure was used. In Step 4, any error was considered sufficiently small. When the basic procedure is used to solve $MP_{\delta\lambda}$ step two is evaluated using an internal LP routine. Except where otherwise noted, problems were run on the University of Michigan Amdahl V8 with MTS operating system.

5.1 Retail Budgeting.

These problems involve determining budgets for departments of a large department store. Each line in Table 1 represents the average of five real problems of the size indicated. The last column indicates the average percent of the best known value for these maximization problems. Details on these problems can be found in Haessler, Sweeney and Talbot (1982).

TABLE 1: Retail Budgeting Problems

No. Dept. (m)	No. Gen. Const.(K)	No. Variables	CPU (Sec.)	% of Best known
5	1	15	.016	97.5%
10	1	30	.051	99.9%
20	1	60	.019	100.0%

5.2 League Scheduling

This problem deals with moving teams from city to city to play games in an athletic league. The objective is to schedule games to minimize travel. Details of these formulations can be found in Bean (1982). They were suggested by the problem in Bean and Birge (1980).

Three problems were solved containing, 14, 16 and 18 teams. Table 2 shows the results. All found optimal solutions as was guaranteed by the fact that the general constraints were equalities. The parenthetical values for dual pivots and CPU time are performance to the best integer solution found. The second values given in each cell are to the end of the algorithm.

TABLE 2: League Scheduling Results

No. Teams (m)	No. Gen Const. (K)	No. variables	Pivots	CPU (Min)
14	14	196	(47) 55	(.004) .005
16	16	256	(46) 54	(.01) .01
18	18	324	(351) 410	(.04) .05

5.3 Random Problems.

The randomly generated problems tested were developed by Martin, Sweeney and Doherty (1982). Details on problem generation can be found there. Only a subset of the problems generated were tested due to budget limits. These consisted of the eight problems with 300 to 400 variables. The problems were tested on the IBM code MPSX-MIP and their own RCBB by Martin et. al. on an Amdahl V/6-II. The problems were tested on this code (MCIPC) on an Amdahl V/8 with the MTS operating system. The ancedotal belief is that the V8 is slightly faster, though the systems are roughly comparable.

Table 3 shows the problem characteristics while tables 4 and 5 show comparisons of the three codes on solution accuracy and CPU usage. In all cases timings did not include data input or solution of the first LP to find $\bar{\lambda}$ for variable reduction. It should be noted that MPSX-MIP and RCBB are general mixed integer codes, though both contain special considerations for handling multiple choice constraints. They both find optimal solutions (if any) if run to termination. The code MCIPC is more limited in that it solves only pure integer problems with exhaustive multiple choice constraints. Further, MCIPC as run here is a heuristic due to the Lagrangean objective. Both RCBB and (clearly) MPSX-MIP require an installation to support MPSX-MIP while MCIPC has a self-contained non-proprietary LP routine. The problem numbers refer to the indexing in Martin, Sweeney and Doherty (1981).

TABLE 3: Problem Characteristics for Random Problems

Problem	No. Var.	No.Gen. Const.(K)	No.GUB-sets(m)	Density†	$\frac{v(P) - v(P)}{v(p)}$
1	352	10	16	28.2	.13
2	315	10	21	21.7	.10
3	345	8	15	26.1	.17
5	323	8	19	22.9	.06
6	380	12	20	24.5	.09
7	323	12	19	16.3	.04
13	384	11	24	15.3	*
14	345	11	15	26.4	.17

*No solution known.

†Fraction of non-zero coefficients for all constraints.

TABLE 4: Best Objective Found for Random Problems

	MPSX/MIP		RCBB		MCIPC	
Problem	Value	Error	Value	Error	Value	Error
1	767	0	767	0	812	5.5%
2	2657	0	2657	0	2838	6.4%
3	1069	0	1069	0	1069	0
5	1815	0	1815	0	1815	0
6	1920	0	1920	0	2103	8.7%
7	804	0	804	0	804	0
13	-	-	-	-	-	-
14	3004	36.8%	2180	1.9%	-	-

- no integer solution found.

TABLE 5: CPU Times in Minutes and Dual Pivots for Random Problems

	MPSX/MIP		RCBB		MCIPC	
Problem	Pivots	CPU	Pivots	CPU	Pivots	CPU
1	(3268)	(.86)	(1070)	(.19)	(3202)	(.24)
	5806	1.66	1774	.35	3440	.26
2	(4906)	(1.33)	(1809)	(.34)	(6668)	(.77)
	7823	2.24	4771	.98	6668	.83
3	(1740)	(.42)	(1125)	(.18)	(111)	(.03)
	3270	.91	2509	.48	111	.04
5	(1986)	(.60)	(481)	(.09)	(258)	(.02)
	2179	.67	1678	.33	258	.02
6	(5934)	(2.07)	(480)	(.09)	(226)	(.02)
	7299	2.61	3498	.75	753	.07
7	(288)	(.08)	(165)	(.04)	(214)	(.01)
	302	.08	165	.04	217	.01
13	-	-	-	-	-	-
14	(8238)	(2.15)	(4407)	(.77)	-	-
	-	-	-	-	-	-

6. Conclusions.

In both the retail budgeting and league scheduling problems MCIPC found solutions in very low computation times. Despite the fact that optimality is not guaranteed in the budgeting problems, in most cases an optimal solution was found.

The results of the random problems were favorable though somewhat mixed. On problems with duality gaps less than 15% of v(P), MCIPC performed significantly better than the other codes. On problems with very large duality gaps, where it is difficult for any of the codes to find even one feasible solution. RCBB appeared quicker at finding such a feasible solution.

In 50% of the random problems on which MCIPC stopped, it found an optimal solution. The average error was 3.44%. Since

none of the scheduling and few of the budgeting problems had solution error, this may be somewhat atributable to fact that these randomly generated constraints have positive slack with probability (nearly) 1 at all solutions. This leads to positive Lagrangean terms $\lambda(b + Ax)$. The real problems did not have this characteristic.

In conclusion, the code MCIP appears to be an efficient tool for solving problems of the structure discussed here. Further, it is self-contained, and written for widely supported FORTRAN-G.

Acknowledgement

I would like to thank Paul Sweeney for his work in coding and testing this algorithm, Frederick Hillier for supporting the initial research and also a referee for many valuable suggestions.

REFERENCES

Beale, E.M.L. and J.A. Tomlin (1970), "Special Facilities in a General Mathematical Programming System for Non-Convex Problems Using Ordered Sets of Variables," **Proceedings of the Fifth IFORS Conference**, pp. 447-454.

Bean, J.C. (1982), "A Branch-and-Bound Algorithm for the Multiple Choice Integer Program" Working Paper No. 82-1, The Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, Michigan 48109.

Bean, J.C. and J.R. Birge (1980), "Reducing Travelling Cost and Player Fatigue in the NBA," **Interface**, Vol. 10, pp. 98-102.

Fisher, M.L. (1981), "Lagrangean Relaxation Method for Solving Integer Programming Problems," **Management Science**, Vol. 27, pp. 1-18.

Geoffrion, A.M. (1974), "Lagrangean Relaxation for Integer Programming." **Mathematical Programming Study 2, North-Holland**, New York, pp. 82-114.

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

Haessler, R., P. Sweeney, F. Talbot (1982), "A Separable Linear Programming Model for Retail Planning," **Proceedings of the 14th Annual Meeting, American Institute for Decision Sciences**, San Francisco, pp. 271-273.

Healy, W.C. (1964), "Multiple Choice Programming," **Operations Research**, Vol. 12, pp. 122-138.

Johnson, M.A., A. Zoltners and P. Sinha (1979), "An Allocation Model for Catalogue Space Planning," **Management Science**, Vol. 25, pp. 117-129.

Martin, K., (1980), "Branching Strategies for Integer Programming with Special Ordered Sets," Presented at ORSA/TIMS National Meeting, Colorado Springs.

Martin, K., D. Sweeney and M. Doherty (1981), "The Reduced Cost Branch and Bound Algorithm for Mixed Integer Programming," Working Paper, Graduate School of Business, University of Chicago. (unnumbered)

Mevert, P. and U. Suhl (1977), "Implicit Enumeration with Generalized Upper Bounds," **Annals of Discrete Mathematics 1**, pp. 393-402.

Ross, G.T. and R. Soland (1973), "A Branch and Bound Algorithm

for the Generalized Assignment Problem" **Mathematical Programming**, pp. 91-103.

Sweeney, D.J., R.A. Murphy (1981), "Branch and Bound Methods for Multi-Item Scheduling," **Operations Research**, Vol. 29, pp. 853-864.

Zoltners, A.A. and P. Sinha (1980), "Models for Sales Resource Allocation," **Management Science**, Vol. 26, pp. 242-260.

Zoltners, A.A., P. Sinha and P. Change (1979), "Algorithm for Sales Representation Time Management," **Management Science**, Vol.25, pp. 1197-1207.

Bean/Integer