Journal of Computational and Applied Mathematics 11 (1984) 231–249
North-Holland

A branch and bound algorithm for solving the multiple-choice knapsack problem

M.E. DYER and N. KAYAL
Department of Mathematics and Statistics, Teesside Polytechnic, Middlesbrough, Cleveland, TS1 3BA, U.K.

J. WALKER
Department of Economics and Statistics, National University of Singapore, Kent Ridge, Singapore 0511

Received 25 January 1984
Revised 30 April 1984

Abstract: The multiple-choice knapsack problem is a binary knapsack problem with the addition of disjoint multiple-choice constraints. We describe a branch and bound algorithm based on embedding Glover and Klingman's method for the associated linear program within a depth-first search procedure. A heuristic is used to find a starting dual feasible solution to the associated linear program and a 'pegging' test is employed to reduce the size of the problem for the enumeration phase. Computational experience and comparisons with the code of Nauss and an algorithm of Armstrong et al. for the same problem are reported.

Keywords: Integer-programs, branch-and-bound, multiple-choice constraints, knapsack problem.

1. Introduction

1.1. Problem definition

This paper deals with the solution of the multiple-choice knapsack (MCK) problem which is of the form:

$$\text{maximise} \quad Z = \sum_{j \in N} c_j x_j, \tag{1}$$

$$\text{subject to} \quad \sum_{j \in N} a_j x_j + x_{n+1} = b, \tag{2}$$

$$\sum_{j \in N_k} x_j = 1, \quad k \in M, \tag{3}$$

$$x_j \geqslant 0, \qquad j \in N \cup \{n+1\}, \tag{4}$$

$$x_j \text{ integer}, \qquad j \in N, \tag{5}$$

where $N = \{1, 2,\dots,n\}$, $M = \{1, 2,\dots,m\}$, $N_p \cap N_q = \emptyset$, $p \neq q$, $p, q \in M$ and $\bigcup_{k \in M} N_k = N$.

Eq. (1) will be referred to as the objective function, the coefficients c_j as the objective function coefficients; (2) as the resource constraint, the coefficients a_j as the resource constraint coefficients; the index set N_k in (3) as the kth multiple-choice set; m the number of multiple-choice sets and n the total number of 0-1 variables.

The coefficients (c_j, a_j) $j \in N$ are assumed to be nonnegative reals with at least one $(c_j, a_j) = (0, 0), j \in N_k$, for all $k \in M$. If this is not the case, then an equivalent problem in which it does hold can be obtained as follows. Let $\underline{a}_k = \min_{j \in N_k} a_j$, $\underline{c}_k = \max_{j \in N_k}\{c_j : a_j = \underline{a}_k\}$ and multiplying the kth multiple-choice constraint in (3) by

(i) $\underline{c}_k$ and subtracting from the objective function; and by

(ii) $\underline{a}_k$ and subtracting from the resource constraint; for all $k \in M$ results in the desired property. (It is still possible that some $c_j < 0$, but the corresponding variables are clearly IP-dominated (see Section 2.1) and can be deleted from further consideration.)

It is also assumed, and again without loss of generality, that the b coefficient is such that

$$0 < b < \sum_{k \in M} \left(\max_{j \in N_k} a_j \right),$$

since if $b < 0$ the problem is infeasible and if $b = 0$ or $b \geqslant \Sigma_{k \in M}(\max_{j \in N_k} a_j)$ the optimal solution is obvious.

The associated linear programming (LP) multiple-choice knapsack ($\overline{\text{MCK}}$) problem is defined by omitting the requirements (5) from MCK, ie. relaxing the integrality constraints. Its optimal objective function value will be denoted by $\overline{Z}$.

1.2. *Previous research*

The MCK has a number of applications, see for example [5], [8], [15] and the references contained therein, and has attracted the attention of a number of researchers. Ibaraki et al. [12], Sinha and Zoltners [14], and Armstrong et al. [2] have developed branch and bound algorithms based on the LP relaxation $\overline{\text{MCK}}$. Ibaraki et al. use approximate solutions to the $\overline{\text{MCK}}$ to obtain lower bounds. Sinha and Zoltners, and Armstrong et al. use penalties and Beale and Tomlin [6] branching in the subproblems generated.

Both [12] and [14] appear to use very rudimentary reoptimisation methods in the LP-subproblems. The present paper uses a dual-simplex scheme, due to Glover and Klingman [10], which permits rapid reoptimisation of the LP-subproblems. Emphasis is also given on the efficient implementation of the appropriate data structures.

Armstrong et al. [2] have developed the approach of [14] also giving emphasis to data structures and storage requirements. Their work bears some resemblance to that contained in this paper. However, the computational experience given is limited and we were unable to reproduce their results (see Section 5). Moreover their program is only available at a charge of \$2000, so we were unable to conduct comparisons with their program. Comparisons with their reported algorithm (obtained by modifying our own program) however, produced results markedly inferior to those given by our own program.

Nauss [13] developed a branch and bound algorithm based on a different relaxation of MCK (Nauss relaxes the multiple-choice constraints (3) and solves the resulting zero-one knapsack problem). In section 5 a report is given of a comparison of a program based on the algorithms developed in this paper with a program for his own method kindly supplied by Nauss.

1.3. Outline of paper

The remainder of this paper is sectioned as follows. Section 2 outlines the algorithms for solving and reoptimising $\overline{\text{MCK}}$. Section 3 outlines the branch and bound procedure and the strategies employed therein. Section 4 describes the data structures and their implementation in a computer program. Section 5 gives computational experience with this program (written in FORTRAN. 77) as implemented on a PRIME 750 computer. (A listing of the program may be obtained from the authors). The sixth and final section presents concluding comments.

2. Solving and reoptimising $\overline{\text{MCK}}$

Since the development of the branch and bound procedure of Section 3 relies heavily on Glover and Klingman's method for solving $\overline{\text{MCK}}$ it is helpful to summarise the basic ideas used. Detailed description of data structures, etc. is deferred to Section 4.

2.1. Convex screening, IP and LP dominance

Propositions 1 and 2 present two properties which will be called IP-dominance and LP-dominance (for proof of propositions see [12]).

Proposition 1 (IP-dominance). *If r, $s \in N_k$ with $a_r \geqslant a_s$ and $c_r \leqslant c_s$, then there are optimal solutions to* MCK *and* $\overline{\text{MCK}}$ *with $x_r = 0$.*

Proposition 2 (LP-dominance). *If r, s and $t \in N_k$ with $a_r < a_s < a_t$, $c_r < c_s < c_t$ and $(c_s - c_r)/(a_s - a_r) \leqslant (c_t - c_s)/(a_t - a_s)$ then there is an optimal solution to* $\overline{\text{MCK}}$ *with $x_s = 0$.*

The meaning of Propositions 1 and 2 is illustrated in Fig. 1. Fig. 1 sketches the 'upper convex boundary' of the coefficients (c_j, a_j), $j \in N_k$. For convenience this boundary is hereafter termed the 'convex hull of N_k'.

Variables whose coefficients (c_s, a_s), $s \in N_k$, fall in the shaded region are LP-dominated as a result of Proposition 2 and, therefore, inessential for $\overline{\text{MCK}}$. Variables whose coefficients (c_r, a_r), $r \in N_k$, fall below the shaded region are IP-dominated as a result of Proposition 1 and, therefore, inessential for MCK.

The term IP-linkage designates a doubly-linked list [11] ordering of the IP undominated variables associated with an index set N_k such that $a_r < a_s < a_t \ldots$ and $c_r < c_s < c_t \ldots$ for r, s and $t \in N_k$.

The term LP-linkage designates a double-linked list [11] ordering of a sublist of the IP-linkage such that $(c_s - c_r)/(a_s - a_r) > (c_t - c_s)/(a_t - a_s)$.

2.2. Dual feasible solutions

The conditions under which basic solutions (with x_{n+1} nonbasic) to $\overline{\text{MCK}}$ are also dual-feasible will now be illustrated. Let NB denote the index set of the nonbasic variables in $\overline{\text{MCK}}$, and $\text{NB}_k = \text{NB} \cap N_k$ for all $k \in M$. Since $\overline{\text{MCK}}$ has $(m + 1)$ linearly independent constraints any basic solution has $(m+1)$ basic variables. It follows that when x_{n+1} is nonbasic all multiple-choice sets n_k, except one, have exactly one basic variable, denoted by x_{k^*}. The exceptional multiple-choice set, N_q say, contains two basic variables and is termed the fractional multiple-choice set. The two fractional basic variables in N_q are denoted by $x_{q'}$ and x_{q^*} where $a_{q'} < a_{q^*}$. With this notation an equivalent form of $\overline{\text{MCK}}$ in which the basic variables are expressed in terms of the nonbasic variables can be written

$$Z + \sum_{k\in M}\sum_{j\in \mathrm{NB}_k}\{(c_{k^*}-a_{k^*}\theta)-(c_j-a_j\theta)\}x_j + \theta x_{n+1} = \delta + \beta\theta, \tag{6}$$

$$x_{q'} + \sum_{k\in M}\sum_{j\in \mathrm{NB}_k}\left\{\frac{(a_j-a_{k^*})}{\alpha}\right\}x_j + \left\{\frac{1}{\alpha}\right\}x_{n+1} = \frac{\beta}{\alpha}, \tag{7}$$

$$x_{q^*} + \sum_{\substack{k\in M\\ k\neq q}}\sum_{j\in \mathrm{NB}_k}\left\{\frac{-(a_j-a_{k^*})}{\alpha}\right\}x_j + \sum_{j\in \mathrm{NB}_q}\left\{1-\frac{(a_j-a_{q^*})}{\alpha}\right\}x_j + \left\{\frac{1}{\alpha}\right\}x_{n+1} = 1-\frac{\beta}{\alpha}, \tag{8}$$

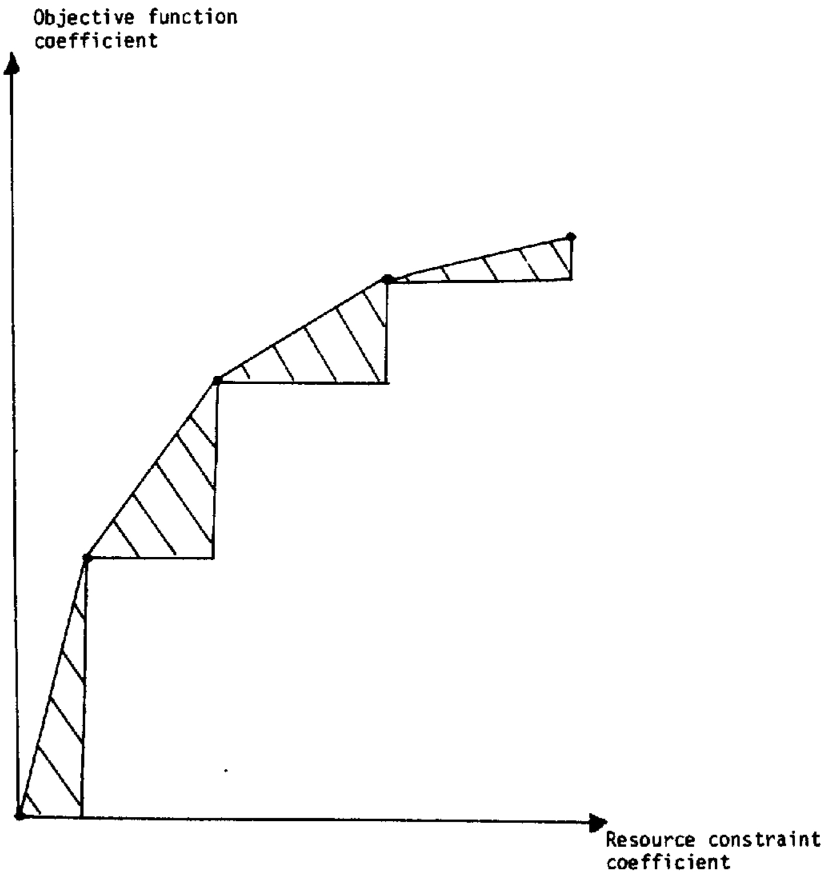
Objective function coefficient
Resource constraint coefficient

Fig. 1. IP and LP dominance.

$$x_{k^*} + \sum_{j \in \mathrm{NB}_k} x_j = 1, \quad \text{for all } k \in M - \{q\}, \tag{9}$$

where

$$\alpha = (a_{q'} - a_{q^*}), \qquad \beta = b - \sum_{k \in M} a_{k^*}, \qquad \delta = \sum_{k \in M} c_{k^*} \quad \text{and} \quad \theta = (c_{q'} - c_{q^*})/\alpha. \tag{10}$$

Thus for this basic solution to be dual-feasible; the coefficients of the x's, i.e., the 'shadow costs', in (6) should be nonnegative. That is $\theta \geqslant 0$ and $(c_{k^*} - a_{k^*}\theta) - (c_j - a_j\theta) \geqslant 0, j \in N_k$ for all $k \in M$. The meaning of this condition is illustrated in Figs. 2 and 3.

The observations above permit a simple method of obtaining or verifying basic dual-feasible solutions to $\overline{\text{MCK}}$. Thus choose any set to be N_q and any two adjacent points on the convex hull of N_q to correspond to the two fractional variables. Let the slope of the edge joining these two points be θ. Choose k^* in each other set N_k, $k \in M - \{q\}$ so that the slope of the convex hull of N_k to the left of (c_{k^*}, a_{k^*}) is greater than or equal to θ, and the slope of the convex hull of N_k to the right of (c_{k^*}, a_{k^*}) is less than or equal to θ.

Such a dual-feasible solution is also primal-feasible (and hence optimal for $\overline{\text{MCK}}$) if the right-hand sides of (7) and (8), calculated according to the formulae (10), are both non-negative.

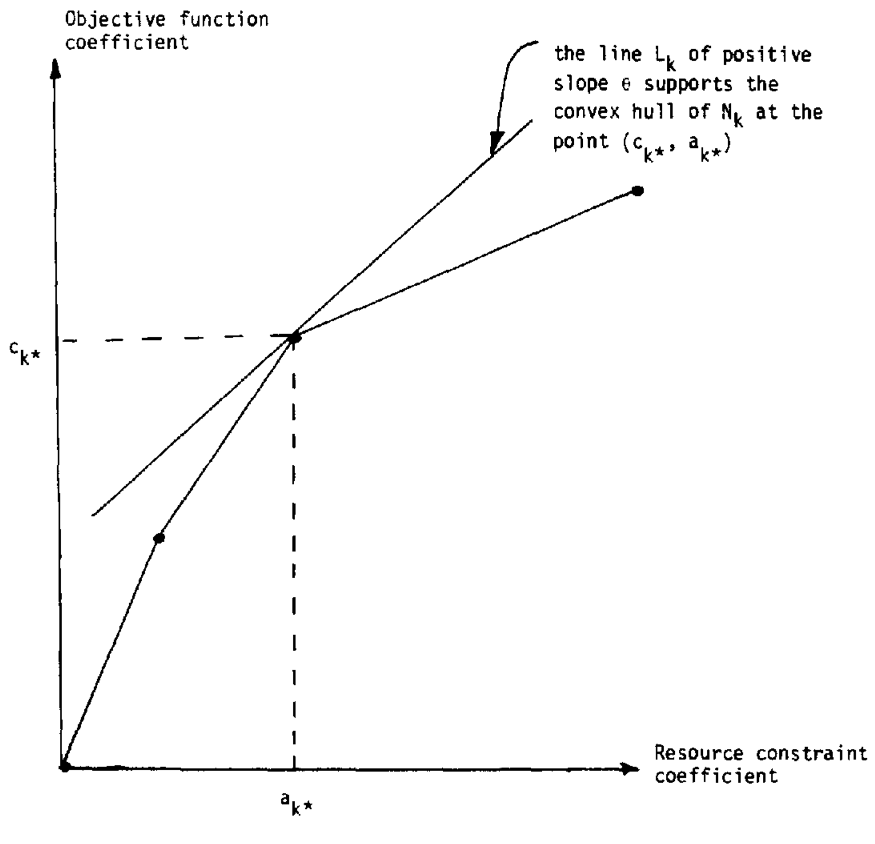
Objective function coefficient
the line L_k of positive slope θ supports the convex hull of N_k at the point (c_{k^*}, a_{k^*})
c_{k^*}
a_{k^*}
Resource constraint coefficient

Fig. 2. Multiple-choice sets N_k, $k \in M - \{q\}$.

The slope θ corresponding to such a solution will be called the optimal slope, θ^*, say. For future reference note that the quantities in any primal-feasible solution to $\overline{\mathrm{MCK}}$ with x_{n+1} nonbasic will satisfy the inequalities.

$$\sum_{\substack{k\in M\\ k\neq q}} a_{k^*} + a_{q'} \leqslant b < \sum_{\substack{k\in M\\ k\neq q}} a_{k^*} + a_{q^*}. \tag{11}$$

In particular an optimal solution to $\overline{\mathrm{MCK}}$ with x_{n+1} nonbasic satisfies (11).

2.3. A starting heuristic IBASIS

A heuristic to find a starting dual-feasible solution is outlined. The aim of this heuristic is to find a dual-feasible solution in which the slope θ is 'reasonably close' in value to the optimal slope θ^* for $\overline{\mathrm{MCK}}$. Armstrong et al. [2] describe a somewhat similar heuristic, but provide no justification that it results in a dual-feasible solution, or in what sense it is likely to be close to the optimal.

From Section 2.1 it is clear that only variables in the LP-linkage need to be considered and this

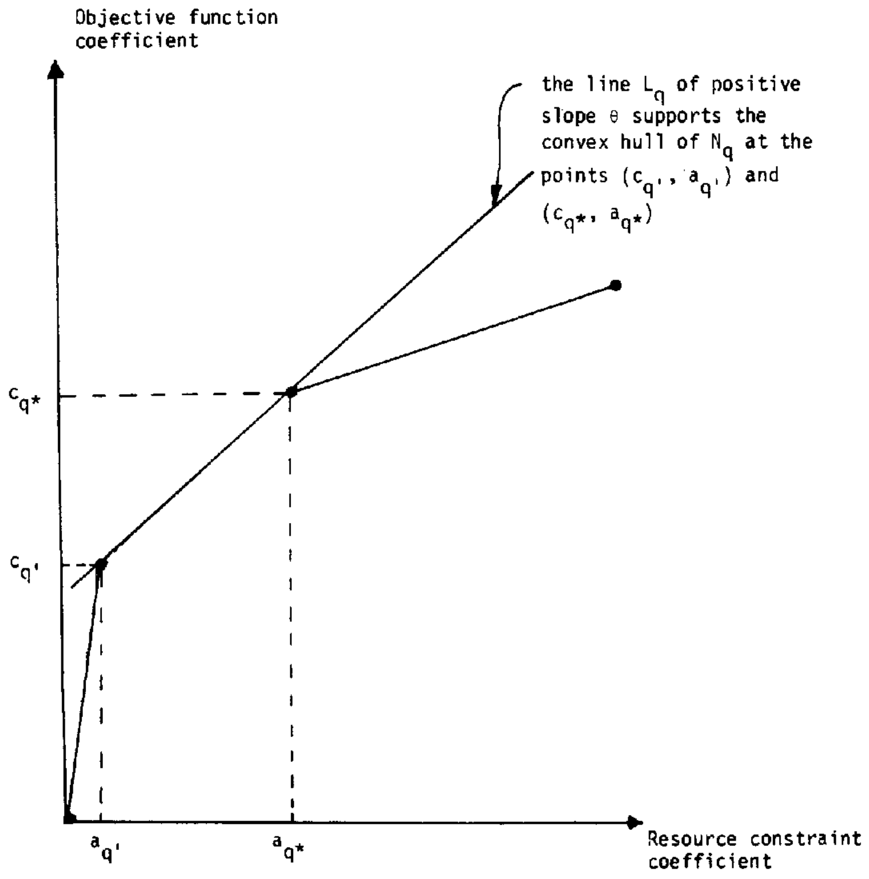
Objective function coefficient
the line L_q of positive slope θ supports the convex hull of N_q at the points $(c_{q'}, a_{q'})$ and (c_{q^*}, a_{q^*})
c_{q^*}
$c_{q'}$
$a_{q'}$
a_{q^*}
Resource constraint coefficient

Fig. 3. Multiple-choice set N_q.

will be assumed in the remainder of Section 2. Now suppose the coefficients (c_j, a_j) are 'reasonably random', then the convex hulls of the sets N_k should be 'relatively similar', $k \in M$. Hence it may be expected that each set N_k would 'utilise' approximately the same proportion of the available right-hand side b of the resource constraint. If the proportions were exactly equal this utilisation would be b/m. Let $a_{k^0} = \max\{a_s\colon a_s \leqslant b/m,\ s \in N_k\}$ for all $k \in M$. Such indices always exist since if for any $k \in M$, $a_s > b/m$ for all $s \in N_k$ implies $b < 0$ and hence the problem is infeasible. Consider the lines, L_k, of slope θ_k, passing through the points (c_{k^0}, a_{k^0}), $(c_{s(k^0)}, a_{s(k^0)})$ where $s(k^0)$ is the next index in the LP-linkage (if k^0 is the last index in the LP-linkage define $\theta_k = 0$) for all $k \in M$. It is shown in Proposition 3 that the slopes of these lines bound the optimal slope, θ^*, for $\overline{\text{MCK}}$.

Proposition 3. *If the points (c_{k^0}, a_{k^0}) are selected such that $a_{k^0} \leqslant b/m < a_{s(k^0)}$ for all $k \in M$ then θ^* the optimal slope for $\overline{\text{MCK}}$ satisfies the inequalities:*

$$\min_{k \in M} \theta_k \leqslant \theta^* \leqslant \max_{k \in M} \theta_k. \tag{12}$$

Proof. Suppose that (12) does not hold, then one of the following two cases must hold:

Case 1. $\theta_k > \theta^*$ for all $k \in M$. This implies that k^*, the index of the optimal basic variable in N_k is located to the right of k^0 in the LP-linkage, i.e., $a_{k^0} < a_{s(k^0)} \leqslant a_{k^*}$, for all $k \in M$. (In the case of N_q, $a_{q^0} < a_{s(q^0)} \leqslant a_{q'} < a_{q^*}$.) Thus,

$$\sum_{k \in M} a_{k^0} \leqslant b < \sum_{k \in M,\, k \neq q} a_{k^*} + a_{q'} < \sum_{k \in M} a_{k^*}$$

which contradicts (11). Therefore, $\theta^* \geqslant \min_{k \in M} \theta_k$.

Case 2. $\theta_k < \theta^*$ for all $k \in M$. This implies that k^*, the index of the optimal basic variable in N_k is located on or to the left of k^0 in the LP-linkage, i.e., $a_{k^*} \leqslant a_{k^0} < a_{s(k^0)}$. Thus

$$\sum_{k \in M} a_{k^*} = \sum_{k \in M,\, k \neq q} a_{k^*} + a_{q^*} \leqslant \sum_{k \in M} a_{k^0} \leqslant b$$

which again contradicts (11). Therefore $\theta^* \leqslant \max_{k \in M} \theta_k$. This completes the proof.

Remark 1. Proposition 3 implies that the average of the θ_k's, $\bar{\theta}$ may be used as a 'good' estimate of θ^* since (12) is always satisfied if $\bar{\theta}$ replaces θ^*. Other options are available e.g. the median of the θ_k's is also a 'good' estimate of θ^* and in this case Step 4 in the following heuristic would be superfluous.

The heuristic, IBASIS, proceeds in four steps

IBASIS

Step 1. Determine the points (c_{k^0}, a_{k^0}) and slopes θ_k satisfying $a_{k^0} \leqslant b/m < a_{s(k^0)}$, for all $k \in M$.

Step 2. Set $\bar{\theta} \leftarrow \sum_{k \in M} \theta_k/m$.

Step 3. Determine a point (c_{k^*}, a_{k^*}) at which $\bar{\theta}$ supports the convex hull of N_k for all $k \in M$. Setting the corresponding variable $x_{k^*} = 1$ for all $k \in M$ gives a dual-feasible solution which will not in general be basic.

Step 4. Adjust the above dual-feasible solution by simultaneously 'rotating' the lines L_k (see Figs. 2 and 3) anticlockwise around the convex hull of N_k, $k \in M$, until one of them, L_q, coincides with a facet of the convex hull of N_q. Let q', $q^* \in N_q$ be the indices of the points defining the facet and θ_q, the slope of the facet, be the initial estimate of θ^*. Note that an anticlockwise rotation corresponds to an increase in θ, a clockwise rotation corresponds to a decrease in θ.

2.4. *The dual-simplex procedure and reoptimisation*

The dual-simplex procedure as specialised to $\overline{\text{MCK}}$ (for a formal description see [10]) is outlined from a geometrical viewpoint. Starting from the dual-feasible solution provided by IBASIS the lines L_k are simultaneously 'rotated' (see Step 4 of IBASIS) clockwise (if the right-hand side of (8) is negative) around the convex hulls of N_k, $k \in M$. Each rotation corresponds to the coincidence of one of the supporting lines L_q with a facet of the convex hull of the fractional multiple-choice set N_q. At each rotation the basic dual-feasible solution is updated and the iterative process terminates when the 'current' dual-feasible solution is also primal-feasible (when the righthand side of (7) and (8) are both nonnegative) and hence optimal.

In the branch and bound procedure to be outlined in Section 3 reoptimisation of $\overline{\text{MCK}}$ is required when constraints of the following type are appended to the problem:

$$x_{q^*} = 1 \tag{13}$$

$$\text{or} \quad x_{q^*} = 0, \tag{14}$$

where x_{q^*} is a fractional variable in the 'current' optimal solution to $\overline{\text{MCK}}$. This reoptimisation is carried out by modifying the dual-simplex procedure above.

If constraint (13) is appended to the problem then effectively the multiple-choice set N_q is deleted from the problem and the solution (putting $x_{q'} = 0$, $x_{q^*} = 1$) becomes infeasible ($\Sigma_{k \in M} a_{k^*} > b$) with the righthand side of (8) negative. The solution can then be 'adjusted' in a manner similar to Step 4 of IBASIS and the dual-simplex procedure used to reoptimise the new problem.

If constraint (14) is appended to the problem then effectively the index q^* is deleted from the set N_q and the solution (putting $x_{q'} = 1$, $x_{q^*} = 0$) becomes nonbasic. The LP-linkage between q' and $s(q^*)$, the successor to q^*, updated to account for the deletion of q^*. The solution can then be 'adjusted' in a manner similar to Step 4 of IBASIS (in this case 'rotating' clockwise) and the dual-simplex procedure used to reoptimise the new problem.

In the 'backtracking' phase of the branch and bound procedure outlined in Section 3 consideration is also required for relaxing constraints of the above type which were previously appended. This is readily achieved by methods similar to those just described.

3. The branch and bound procedure

Branch and bound procedures are commonly used to solve integer programming problems (see [7,8]). The branch and bound procedure for solving MCK presented in this paper is of a fairly standard type and consists of solving a structured hierarchy of subproblems. Each branch corresponding to a subproblem, MCK_{NODE}, say, of MCK is generated from its predecessor by adding an appropriate constraint. The corresponding subproblem relaxation $\overline{\text{MCK}}_{\text{NODE}}$ is solved to obtain an upper bound $\overline{Z}_{\text{NODE}}$ on the optimal value of Z. Termination of a branch occurs when one of the usual 'fathoming' rules (see [7]) apply, i.e.,

(a) $\bar{Z}_{\text{NODE}} < Z_{\text{INCUMBENT}}$;

(b) 'current' $\overline{\text{MCK}}_{\text{NODE}}$ solution is integer ($x_q^* = 0$) and $Z_{\text{INCUMBENT}}$ is updated; or

(c) 'current' $\overline{\text{MCK}}_{\text{NODE}}$ solution is infeasible.

When any branch is terminated, the next subproblem considered is chosen by the LIFO policy, i.e., a depth-first search.

3.1. *The branching phase*

The branching scheme adopted in the branch and bound procedure is depth-first search. Other schemes are obviously possible but the simplicity of depth-first search made it an obvious first choice. Whenever an optimal solution to $\overline{\text{MCK}}_{\text{NODE}}$ is found and the NODE cannot be fathomed then a new subproblem is generated by 'branching on' the fractional variable x_q^* in the current optimal solution. The following strategies have been investigated (computational results are reported in Section 5).

(a) 'Branching through one', the constraints $x_q^* = 1$ is appended to the current subproblem prior to subsequent backtracking and then 'branching through zero', the constraint $x_q^* = 0$ is appended to the current subproblem.

(b) 'Branching through zero', the constraint $x_q^* = 0$ is appended to the current subproblem prior to subsequent backtracking and then 'branching through one', the constraint $x_q^* = 0$ is appended to the current subproblem.

(c) Beale–Tomlin [6] branching, the constraint $\sum_{j\in N_q, j\text{"to right of } q^0\text{"}} x_j = 1$ appended to the current subproblem prior to subsequent backtracking and then branching by the constraint $\sum_{j\in N_q, j\text{"at and to left of } q^0\text{"}} x_j = 1$ appended to the current subproblem where q^0 is as defined in Section 3.2 below.

(d) Beale–Tomlin branching in conjunction with the criterion for selection of the first branch to be explored as proposed by Armstrong et al. [2].

Note. In the case of (c) and (d) the subsequent reoptimisation of the $\overline{\text{MCK}}_{\text{NODE}}$ can be achieved in a manner similar to that employed when the constraint (14) is appended to the current subproblem. In this case several indices are deleted from the set N_q prior to updating of the LP-linkage.

3.2. *The bounding phase*

Upper bound on the optimal value of Z. In this paper the upper bound on the optimal value of Z is simply $\bar{Z}$ the optimal value of $\overline{\text{MCK}}$.

Lower bound on the optimal value of Z. A lower bound is calculated in the following manner. From Section 2.2 inequalities (11) hold when an optimal solution to $\overline{\text{MCK}}_{\text{NODE}}$ is found. The left hand inequality in (11) corresponds to an obvious feasible solution to MCK and, therefore, an attempt is made to 'utilise' the slack in this inequality by finding an index in the IP-linkage, $q^0 \in N_q$, with $a_{q'} < a_{q^0} < a_{q^*}$ such that

$$\sum_{\substack{k\in M\\ k\neq q}} a_{k^*} + a_{q^0} \leqslant b < \sum_{\substack{k\in M\\ k\neq q}} a_{k^*} + a_{s'(q^0)}$$

where $s'(q^0)$ is the successor index to q^0 in the IP-linkage. A lower bound is then given by

$$Z_{\mathrm{LB}} = \sum_{\substack{k \in M \\ k \neq q}} c_{k^*} + c_{q^0}.$$

In each subproblem, Z_{LB} is calculated and $Z_{\mathrm{INCUMBENT}}$ is updated when possible. If the incumbent is updated then a search is made for a stronger lower bound using a special case of the 'single-complement' technique of Balas and Martin [4]. An outline of its implementation is as follows:

SINGLE

```
for $i \leftarrow 1$ to $m$ do
    set $x_{k^*} \leftarrow 1$ for all $k \in M - \{i\}$
    set $i^0 \leftarrow$ predecessor of $i^*$ in the IP-linkage of $N_i$
    while $\sum_{k \in M, k \neq i} a_{k^*} + a_{i^0} > b$ do
        set $i^0 \leftarrow$ predecessor of $i^0$ in the IP-linkage of $N_i$
    repeat
    set $Z_{\mathrm{LB}} \leftarrow \sum_{k \in M, k \neq i} c_{k^*} + c_{i^0}$
    set $Z_{\mathrm{INCUMBENT}} \leftarrow$ maximum$\{Z_{\mathrm{INCUMBENT}}, Z_{\mathrm{LB}}\}$
repeat.
end.
```

3.3. A pegging test, PEGTOØ

In Section 2.1 IP- and LP-dominance was used to eliminate variables that are inessential to the optimal solutions to MCK and $\overline{\text{MCK}}$ respectively. The elimination of variables inessential to MCK is further strengthened by using a 'pegging test' before and (optionally) during the branch and bound procedure. This test is derived as follows. Consider the Lagrangian dual relaxation (see [9]) LDR_λ of MCK.

$$\nu(\lambda) = \max\left(\sum_{j \in N} c_j x_j + \lambda \left[b - \sum_{j \in N} a_j x_j \right] \right),$$

$$\text{subject to } \sum_{j \in N_k} x_j = 1 \text{ for all } k \in M,$$

where $\lambda \geq 0$ is a specified Lagrangian multiplier. It is well known that this is related to the LP relaxation $\overline{\text{MCK}}$ (see [9]) and it follows that $\nu(\lambda) \geq \bar{Z}$ for all $\lambda \geq 0$, and $\nu(\lambda^*) = \min_{\lambda \geq 0}(\nu(\lambda)) = \bar{Z}$ when λ^* is the optimal Lagrangian multiplier. It can be shown (see [9]) that $\lambda^* = \theta^*$ where θ^* is defined in Section 2.3. Let $x_{k^*} = 1$, for all $k \in M$ now denote an optimal solution to LDR_λ. Setting $x_{k^*} = 0$, $x_j = 1$, $x_r = 0$, $j, r \in N_k$, $j \neq r$ then (in fairly obvious notation)

$$\nu(\lambda | x_j = 1) = \nu(\lambda) + (c_j - \lambda a_j) - (c_{k^*} - \lambda a_{k^*}) \geq \bar{Z}(x_j = 1)$$

When $\lambda = \lambda^*$, i.e., $\nu(\lambda^*) = \bar{Z}$ then

$$\bar{Z} + (c_j - \lambda^* a_j) - (c_{k^*} - \lambda^* a_{k^*}) \geq \bar{Z}(x_j = 1).$$

If Z_{LB} is any lower bound for Z in MCK then x_j, for any $j \in N_k$, $k \in M$, can be set to zero without loss if $\overline{Z}(x_j = 1) \leqslant Z_{\mathrm{LB}}$ which is certainly true if $\overline{Z} + (c_j - \lambda^* a_j) - (c_{k^*} - \lambda^* a_{k^*}) \leqslant Z_{\mathrm{LB}}$.

Thus, if $(c_{k^*} - \theta^* a_{k^*}) - (c_j - \theta^* a_j) \geqslant \overline{Z} - Z_{\mathrm{LB}} = \Delta$ say, then x_j can be 'pegged to zero', i.e., deleted from MCK.

The procedure, PEGTO∅ may be outlined from a geometrical viewpoint. See Fig. 4. Any point lying a vertical distance greater than or equal to Δ below the optimal supporting line L_k of slope θ^* is pegged to zero, i.e., all points below or on the line L'_k.

It is hoped that as a result of this procedure a 'reasonable' fraction of the variables can be pegged to zero before the branching phase starts (it is obvious that the number of variables pegged to zero is dependent on the sharpness of the lower bound).

Two strategies for this test have been investigated (computational results are reported in Section 5):

(a) PEGTO∅ used once only prior to the branching phase; or

(b) PEGTO∅ used in every subproblem.

Remark 2 The deletion of IP-dominating variables may require additional book-keeping, eg. updating of the IP- and LP-linkages. For example, in Fig. 4, if the point a is deleted then point f

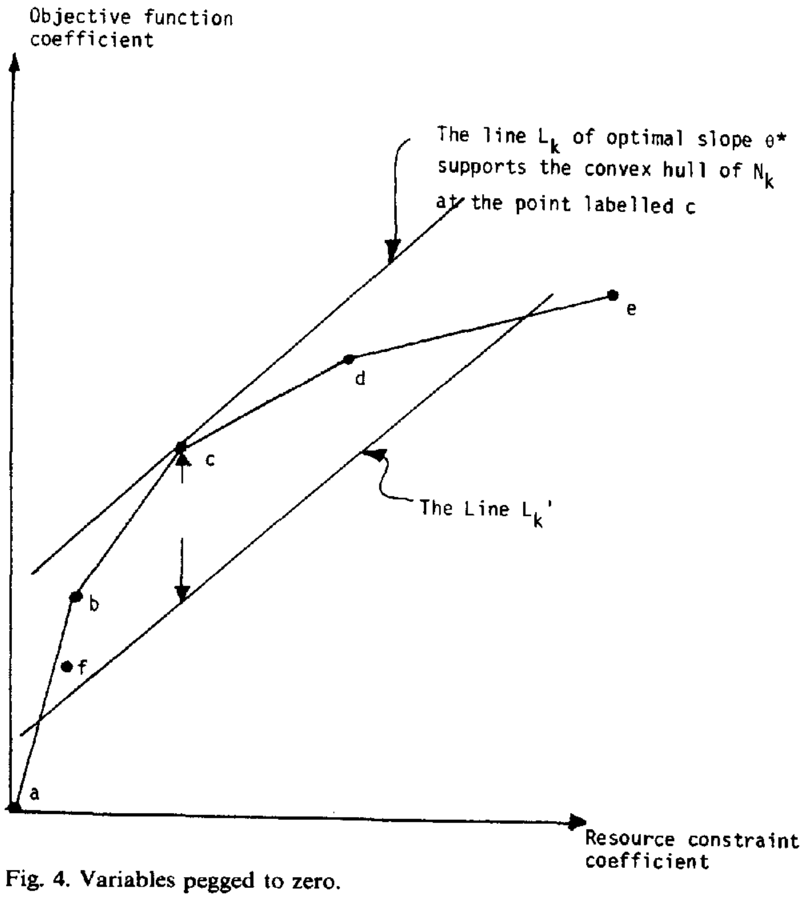
Objective function coefficient
The line L_k of optimal slope θ^* supports the convex hull of N_k at the point labelled c
e
d
c
The Line L_k'
b
f
a
Resource constraint coefficient

Fig. 4. Variables pegged to zero.

becomes the first point in the IP-linkage and LP-linkage and the LP-linkage updated between points f and b. Clearly, the deletion and reinstatement of variables conditionally pegged to zero in subproblems will also require additional book-keeping operations of this kind.

4. Data structures and implementation

In order to implement the various procedures efficiently, adequate data structures must be used. For more details on the data structures used see [11]. The data structures used by the procedures are determined by the three major types of entity involved namely the variables x_j, $j \in N$; the multiple-choice sets N_k, $k \in M$; and the nodes of the branch and bound search tree, NODE. In Figs. 5–8 an arrow indicates that the data item is a pointer to an entity of the stated type.

4.1. Data structures for entities

Variables. Associated with each variable x_j, $j \in N$, is the following record of data items (see Fig. 5). The first and second pointers point to the predecessor and successor respectively of variable x_j in the IP-linkage ($p'(j) = 0$ if variable x_j is the first variable in the IP-linkage, $s'(j) = 0$ if variable x_j is the last variable in the IP-linkage).

The third and fourth pointers point to the predecessor and successor respectively of variable x_j in the LP-linkage (with obvious meaning for $p(j) = 0$ and $s(j) = 0$).

There are three levels of structuring associated with the procedures and a variable has to qualify for at least one of these levels:

(a) level 1: variables which are IP-dominated;

(b) level 2: variables which are LP-dominated, but IP-undominated; or

(c) level 3: variables which are LP-undominated.

During the branch and bound phase, variables may be temporarily 'suspended' as they are branched upon, i.e., they disappear from levels 2 and 3. Consequently other variables may change from level 2 to level 3. However, in all cases, their associated data must be left intact so that they can be reinstated correctly on 'backtracking'. The double-linked lists used in levels 2 and 3 allow the rapid 'suspension' and 'retrieval' of variables in this way.

Multiple-Choice Sets. Associated with each multiple-choice set N_k, $k \in M$, is the following

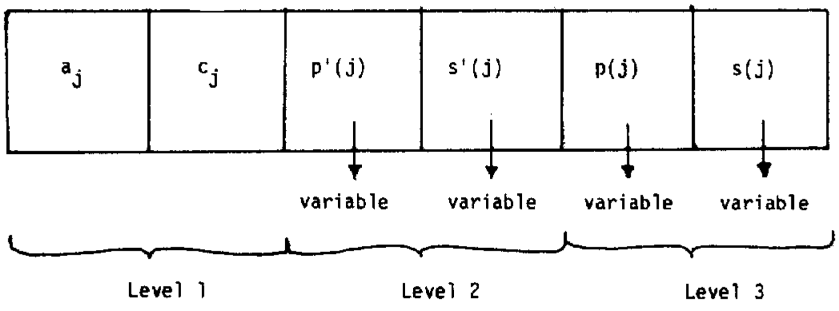

Fig. 5. Data structure for variables.

record of data items (see Fig. 6). The first pointer points to the variable x_{k^*}. The second pointer points to the variable in N_k that has the value 1 in the incumbent solution to MCK. The third and fourth data items are defined as follows:

$$\text{'forward slope':fslope} = (c_{s(j)} - c_j)/(a_{s(j)} - a_j) \quad (\text{zero if } s(j) = 0),$$

$$\text{'backward slope':bslope} = (c_{p(j)} - c_j)/(a_{p(j)} - a_j) \quad (+\infty \text{ if } p(j) = 0).$$

The third and fourth pointers point into two subsidiary data structures stored as linear arrays, the 'max (or forward) heap' denoted by fheap, and the 'min (or backward) heap' denoted by bheap. These have the form shown in Fig. 7. Note that the heap ordering is by the slopes (fslope of bslope) of the sets pointed to, rather than the entries themselves. Since fractional multiple-choice sets may be temporarily 'suspended' as the branch and bound phase progresses the data associated with the multiple-choice sets will be continually changing. The two way pointers, e.g. heapf, fheap, may be used to facilitate the rapid retrieval and updating of the data blocks associated with the multiple-choice sets.

Nodes. Each node in the branch and bound search tree is implicitly associated with an array position NODE in an array which essentially implements a simple push-down stack. Associated with each stack item NODE is the following item of data records (see Fig. 8).

The first pointer points to the fractional variable x_{q^*} which is to be branched upon. The

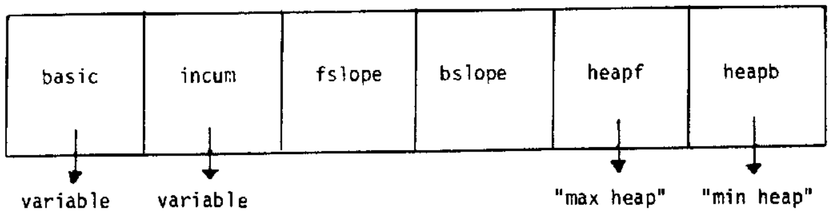

Fig. 6. Data structure for multiple-choice sets.

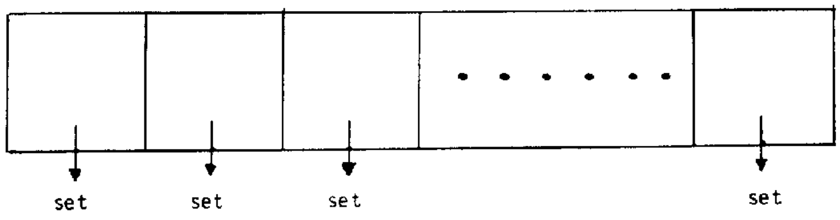

Fig. 7. 'Max/min heap' array of length m.

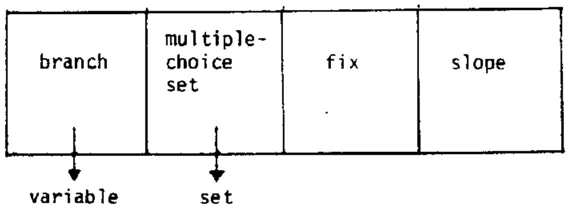

Fig. 8. Data structure for stack items.

second pointer points to the multiple choice set N_q to which the index q^* belongs. The third data item is given the values 0 to 1 according to the branch taken, i.e., $x_{q^*} = 0$, $x_{q^*} = 1$. The fourth data item records the 'current' optimal slope associated with $\overline{\mathrm{MCK}}_{\mathrm{NODE}}$ and together with fix is used as an aid in backtracking.

Branching 'pushes' down the stack while backtracking 'pops' the stack (possibly more than once since several subproblems may become fathomed at one go). The implementation of the optional strategy of 'pegging' variables is implemented in a similar manner, i.e., 'pushed down' with the newly pegged variables on branching and 'popped' on backtracking.

Clearly, the meaning of the nodes on the stack will change as the branch and bound search proceeds since the stack only identifies unfathomed nodes. The data structures outlined are sufficient to represent a depth-first search but, for a more general branching strategy, a more complex data structure of a tree type would be needed.

4.2. *IP- and LP-linkage*

The doubly-linked IP- and LP-linkages allow the rapid scan of the coefficients (c_j, a_j), $j \in N_k$, $k \in M$ and also the efficient suspension and retrieval of variables. The establishment of the IP-linkage is carried out in two stages. Stage one sorts the coefficients a_j, $j \in N$ in non-decreasing order (see [11, HEAPSORT]). Note that the variable entities themselves are not swapped with one another as in 'pure' HEAPSORT but rather the pointers $s'(j)$ (and subsequently the pointers $p'(j)$ are established). Stage two deletes all IP-dominated variables. The establishment of the IP-linkage is done prior to the LP-linkage, since it clearly facilitates the formation of the LP-linkage.

4.3. *Solving $\overline{MCK}$*

Section 2.4 outlined the dual-simplex procedure used to solve $\overline{\mathrm{MCK}}$. The dual-simplex procedure operates only on the data structures for the variables and multiple-choice sets as defined in Section 4.1. The procedure selects the incoming basic variable according to the direction the lines are rotated. To do this it is necessary to be able to select the minimum backward slope or the maximum forward slope at each iteration. Since this step is repeated frequently, it is more efficient to store these slopes in heap structures [11]. The algorithm used to 'heapify' the slopes is as in [11, p. 67], with the exception that the keys of the corresponding slopes (i.e., indices of the multiple-choice sets) are swapped with each other rather than the slopes themselves. The 'heapf' and 'heapb' pointers are used here together with the heap arrays 'fheap' and 'bheap'. This is done because the interest is in the identity of the multiple-choice set which will be the new fractional multiple-choice set. These heaps must obviously be updated from one iteration to the next. Algorithm ADJUST [11, p. 67], is used for this purpose. (The multiple-choice set index is used as a secondary key to resolve ties.)

4.4. *Branching*

As outlined in Section 3.1 all the branching strategies investigated can be implemented by 'branching through one' or 'branching through zero'. It may be noted that there is a kind of symmetry between branching through one and through zero. Branching through one allows the 'suspension' (or temporary deletion) of the multiple-choice set N_q until it is 'backtracked' to, while branching through zero allows the 'suspension' (or temporary deletion) of the variable x_{q^*} until it is 'backtracked' to. Similarly, in solving the resulting $\mathrm{MCK}_{\mathrm{NODE}}$ the backward slopes are needed when $x_{q^*}=1$ is appended and the forward slopes are needed when $x_{q^*}=0$ is appended. It is clearly necessary to have data structures which enable the storing and updating of both slopes in an efficient and rapid fashion. The heap structures of Section 4.3 allow this to be done, though the updating operations needed are slightly more extensive. These operations are implemented as in [11] (INSERT, p. 63 and ADJUST, p. 67). In 'suspending' the variable x_{q^*} (i.e., $x_{q^*}=0$), the pointers for the variables corresponding to $s'(q^*)$ and $p'(q^*)$ are adjusted by setting them to point each other. The convex hull of N_q is then updated by reforming the LP-linkage between q' and $s(q^*)$ (or $p'(q^*)$ if q^* is the last index in the LP-linkage. In 'suspending' the multiple-choice set N_q (i.e., setting $x_{q^*}=1$) the heaps and their corresponding pointers are adjusted by swapping these pointers with those corresponding to the last entries and reducing the sizes of the heaps each by one. The heaps are then reformed using INSERT and/or ADJUST [11].

Remark 3. In 'suspending' the variable x_{q^*}, the pointers $p(q^*)$, $s(q^*)$, $p'(q^*)$ and $s'(q^*)$ are left unchanged which helps in reinstating x_q^*. In 'suspending' N_q, the corresponding pointers are changed to indicate that its current slopes are to be found at the end of the current heap arrays (before they are reduced). This again helps in reinstating it.

Remark 4. In updating the heaps while pivoting forward (say) it is obvious that the maximum slope in the forward slopes will be the minimum slope in the backward slopes. Advantage is taken of this fact in implementing the heap operations. A similar approach is used in pivoting backwards, and in 'suspending' of a multiple-choice set or a variable.

Remark 5. When the option of 'pegging' at every subproblem is employed, the data structures corresponding to the variables (Fig. 5) are updated for every such variable, but the data structures for the sets (Figs. 6 and 7) only need updating when either the predecessor or successor to the current basic variable is pegged. The reverse operation must then be carried out when backtracking.

4.5. Backtracking

Backtracking is the complementary operation to branching. As in branching, there are similarities between backtracking through one and through zero. Backtracking through zero uses the backward slopes to solve the resulting $\overline{\mathrm{MCK}}_{\mathrm{NODE}}$, while backtracking through one uses the forward slopes. Thus most of the operations of backtracking simply reverse those of branching. In 'reinstating' the variable x_{q^*}, the pointers $p'(s'(q^*))$, $s'('p(q^*))$, $p(s(q^*))$ and $s(p(q^*))$ are all set to point to q^*. Correspondingly in reinstating the multiple-choice set N_q, the size of the heaps are increased by one and updated using INSERT [1]. (See Remark 3 above.)

5. Computational experience

The branch-and-bound algorithm was implemented through a program written in FORTRAN 77 and tested on a PRIME 750 computer. The times reported are in seconds. I/O time, together with time for sorting the coefficients and establishing the IP-linkage is excluded.

The test problems were randomly generated. The (integer) constraint and objective function coefficients were independently generated from a uniform distribution on an interval with lower limit 100 and of fixed (but selectable) width. However, the generator disallowed the repetition of coefficients within a multiple-choice set in such a way as to comply with the IP-linkage. Then b, the right-hand-side of (2), was calculated as follows:

$$b = \sum_{k \in M} \tfrac{1}{2}\left(\min_{j \in N_k}(a_j) + \max_{j \in N_k}(a_j)\right).$$

Initial analysis was carried out on the performance of the routine PEGTO∅ for implementing the pegging test of subsection 3.3 and the routine IBASIS of subsection 2.3 for determining an initial basic dual-feasible solution. Table 1 shows a significant proportion of IP-dominating variables are pegged to zero prior to any branching. Table 1 also shows the number of iterations needed to solve $\overline{\text{MCK}}$ starting from the basis given by IBASIS.

Table 2 shows the results of further analysis on various features of the program and possible modifications to it. The key to the program versions in columns 1 to 6 of Table 2 is as follows:

(1) Without SINGLE, or PEGTO∅ at every node.
(2) With SINGLE, but without PEGTO∅ at every node.
(3) With both SINGLE and PEGTO∅ at every node.
(4) As program (3), but with Beale–Tomlin [6] branching.
(5) As program (4), but with Armstrong et al. [2] branch selection rule.
(6) Program modified to implement algorithm of Armstrong et al. [2].

It will be observed that the incorporation of the routine SINGLE (described in Section 3.2) to improve the incumbent at each update, significantly reduces computational times and numbers of

Table 1

Number of variables	Number per m/c set	Range width	% of IP-dominating variables pegged to zero	No. of iterations
250	5	400	56.4	64.4
	125	400	48.9	5.6
	5	3200	37.4	59.8
	125	3200	56.8	7.0
1000	5	400	41.16	255.6
	125	400	47.4	23.2
	5	3200	55.88	245.8
	125	3200	62.5	21.6
4000	5	400	46.0	1001.4
	125	400	39.5	90.4
	5	3200	51.0	987.6
	125	3200	35.47	76.6
8000	5	400	51.8	1990.8
	125	400	55.12	188.8
	5	3200	55.16	1980.4
	125	3200	63.84	174.0

Each row is the average of five problems.

Table 2
Computation times and number of nodes generated

Number of variables	Number per m/c-set	Range with	Time (seconds)						Number of nodes generated					
			1	2	3	4	5	6	1	2	3	4	5	6
4000	5	3200	210.8	68.2	126.6	125.8	118.6	207.1	41559	13129	935	941	881	39109
			180.7	13.7	79.5	75.4	54.7	93.5	37697	2269	569	569	401	18147
			125.2	40.4	111.4	104.2	37.8	224.4	24833	7935	795	789	269	41563
			114.2	89.2	104.5	101.7	174.3	284.8	21633	17089	757	757	1309	49057
			117.2	44.1	137.6	136.1	121.5	238.7	34465	8237	1011	1019	907	42181
4000	25	3200	39.4	13.1	9.1	8.0	9.5	27.4	8323	2837	245	229	261	4189
			127.0	10.0	11.0	10.0	5.3	23.7	29003	1819	303	303	133	3455
			141.2	35.1	27.3	24.3	14.0	36.3	30311	7517	815	771	399	5613
			90.8	8.6	5.6	5.3	13.6	56.8	17403	1375	141	139	401	9155
			89.0	46.8	21.2	20.0	7.5	21.5	19153	9845	633	617	195	3385
4000	125	3200	36.5	5.9	1.6	1.5	3.1	9.2	6353	1423	61	49	201	683
			175.9	34.6	4.7	3.5	1.3	4.0	33533	8151	397	281	59	249
			50.6	24.5	2.6	1.4	1.2	45.3	8387	4455	175	51	45	3651
			35.9	4.2	1.6	1.6	1.4	4.5	4741	513	85	75	61	305
			*	56.4	5.0	4.1	2.0	7.6	*	10599	321	291	97	629
8000	5	3200	534.3	214.0	631.4	611.3	296.1	288.0	98885	36000	2411	2411	1087	49925
			*	11.4	75.7	73.1	61.9	65.2	*	1447	273	273	215	10171
			38.7	16.2	124.4	121.2	300.9	276.6	6243	2353	463	263	1125	46443
			80.2	50.4	280.3	278.8	575.6	626.0	15165	8583	1053	1073	2023	106181
			241.8	88.1	299.3	299.3	140.2	543.0	43719	14075	1159	1159	519	95217
8000	25	3200	181.2	37.1	28.9	27.8	32.2	144.4	35149	6627	461	461	519	21669
			*	31.5	27.0	22.3	18.6	269.3	*	6115	411	357	269	39923
			50.1	13.4	12.2	11.5	9.9	29.5	8899	1723	175	175	135	4191
			*	2.3	5.7	5.5	5.0	14.6	*	153	61	61	53	1989
			87.5	24.4	29.8	26.5	6.8	98.5	15323	3325	479	439	83	14139
8000	125	3200	56.7	50.2	6.9	6.5	4.4	5.1	10607	9811	313	299	151	291
			71.5	11.8	4.2	3.7	2.6	7.8	9857	1391	169	137	69	483
			81.4	42.5	4.5	4.5	2.7	2.4	14657	7615	151	139	67	77
			24.6	21.9	9.8	7.9	2.8	7.5	5569	5015	523	403	87	499
			*	21.7	5.4	3.2	3.0	15.9	*	3249	257	111	97	1061

* Not solved due to storage limitation (tree depth greater than 1500).

nodes in the search tree. Also the inclusion of the routine PEGTO∅ (described in section 3.3), at every node, rather than only at the root, leads to further reductions for problems with large multiple-choice sets, but appears worse for problems with very small sets. Our program therefore includes the facility for pegging at every node as an option. (We recommend that this option is used for problems having, say, at least 25 variables per set.) The choice of branching strategy was also examined. The method employed in our program is to always branch first by setting the fractional variable with large resource coefficient to 1. (We examined the strategy of always setting this variable to zero, and found no significant difference, though this is not reported here.) The other two comparisons shown are for two different branching strategies. The first uses Beale–Tomlin [6] branching, branching first to the 'right' of the fractional variable with small resource coefficient. The second uses Beale–Tomlin branching in conjunction with the criterion for selection of the first branch to be explored as proposed by Armstrong et al. [2]. In all other respects both of these modified programs were the same as our own. The results indicate that the first of these modifications produces small improvements in most cases, the second produces larger improvements in the majority of cases but is substantially worse in a few. Since neither appears to dominate the strategy we have implemented in terms of computation time, we have not offered these facilities. However, the program is coded in a flexible way, so as to allow different branching strategies to be incorporated by the user, if so desired.

We have also attempted to compare our program with methods proposed in the literature. This proved difficult since, of those authors contacted, only R. Nauss was willing to supply actual code. Table 3 gives a comparison of a very rudimentary form of our algorithm (without the routines SINGLE and PEGTO∅) with Nauss' program. it is obvious that our program is markedly superior to Nauss', even in this form. We do not claim that Nauss' program is a state-of-the-art code, but was the only one available for comparison.

We also wished to conduct a comparison with the recently published method of Armstrong et al. [2]. For reasons already explained we could not obtain actual code. We therefore modified our own program so as to implement the algorithm described in [2]. (This approximates their version 7.) The results are shown in Table 2 column 6. Clearly our program (run with the recommenda-

Table 3

Number of variables	Number per m/c-set	Range width	Nauss' algorithm	Rudimentary algorithm
200	5	50	0.937	0.359
	10	100	1.325	0.933
	20	200	1.313	0.541
	40	400	1.928	1.236
500	10	100	3.351	1.194
	25	250	7.655	2.144
	50	500	52.866	4.573
	100	1000	23.078	4.683
1000	50	500	655.339	11.965
	100	1000	163.208	15.494
	200	2000	418.137	41.347

Each row is the average of ten problems, time reported in seconds.

tion above regarding pegging at every node in relation to size of the multiple-choice sets) is demonstrably superior. However, we were puzzled by the fact that this program was unable to obtain results anywhere nearly as good as those reported in [2], particularly with regard to the numbers of nodes generated. We can only suppose that the test problems of [2], which are not clearly described, differ from our own.

6. Conclusions

We believe that our results show that the use of appropriate data structure allows the development of an efficient branch-and-bound code for the MCK problem, and, moreover, lead us to suggest that in practice the MCK problem (contrary to worse-case complexity analysis [1]) is not computationally difficult. We believe further that the program we have developed represents the current state of the art in this area, and we are willing to supply any interested researcher with a copy.

References

[1] A. Aho, J. Hopcroft and J. Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, MA, 1974).

[2] R.D. Armstrong et al., A computational study of a multiple-choice knapsack algorithm, *ACM Trans. on Math. Software* **9** (1983) 184–198.

[3] J. Ashcroft, R. Eldridge, R. Paulson and G. Wilson, *Programming with Fortran 77* (Granada Publishing, 1982).

[4] E. Balas and C.H. Martin, Pivot and complement—A heuristic for 0-1 programming, *Management Sci.* **26** (1980) 86–96.

[5] J. Balintfy, G. Ross, P. Sinha and A. Zoltners, A mathematical programming system for preference and compatibility maximised menu planning and scheduling, *Math. Programming* **15** (1978) 63–76.

[6] E. Beale and J. Tomlin, Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables, in: J. Lawrence ed., *Proc. Fifth International Conference of Operational Research* (Tavestock, London, 1969) 821–828.

[7] R. Breu and C. Burdet, Branch and bound experiments in 0-1 programming, *Mathematical Programming Study* **2** (1974) 1–50.

[8] R. Garfinkel and G. Nemhauser, *Integer Programming* (Wiley, London, 1972).

[9] A.M. Geoffrion, Lagrangian relaxation for integer programming, *Mathematical Programming Study* **2** (1974) 82–114.

[10] F. Glover and D. Klingman, A $0(n \log n)$ algorithm for LP knapsack with GUB constraints, *Mathematical Programming* **17** (1979) 345–361.

[11] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms* (Pitman, London, 1978).

[12] T. Ibaraki, T. Hasegawa, K. Teranaka and J. Iwase, The multiple-choice knapsack problem, *J. Operations Res. Soc. Japan* **21** 91) (1978) 59–95.

[13] R. Nauss, The 0-1 knapsack problem with multiple-choice constraints, *European J. Oper. Res.* **2** (1978) 125–131.

[14] P. Sinha and A. Zoltners, The multiple-choice knapsack Problem, *Operations Res.* **27** (3) (1979) 503–515.

[15] F. Tillman, C. Hwang and W. Kuo, *Optimisation of Systems Reliability* (Dekker, New York, 1980).

[16] C. Witzgal, On one-row linear programs, *Extremal Methods and System Analysis, Symposium 1977* (Springer, Berlin, 1977) 384–414.

[17] E. Zemel, The linear multiple-choice knapsack problem, *Operations Res.* **28** (6) (1980) 1412–1423.