A DYNAMIC PROGRAMMING ALGORITHM FOR MULTIPLE-CHOICE CONSTRAINTS

MARY W. COOPER and KEVAN FARHANGIAN
Southern Methodist University, Department of Operations Research and Engineering
Management, School of Engineering and Applied Science, Dallas, TX 75275, U.S.A.

(Received April 1983, in revised form October 1983)

Communicated by Ervin Y. Rodin

Abstract—This report concerns a discrete mathematical programming problem in which the variables are binary or integer, the objective function separable or factorable, and the constraints are in either of two classes: linear or multiple-choice constraints. The problem is solved using a dynamic programming approach with fathoming by bounds and by infeasibility.

INTRODUCTION

Beale and Tomlin[1] have presented a paper on a non-convex problem with special restrictions. These restrictions grouped variables into “ordered sets of variables”. Within a set, only a single variable could take on a positive value. Other researchers have used the terminology, multiple-choice constraints or GUB constraints, for the constraints which express the ordered set relationship. Bean[2] cites numerous applications for linear integer problems with such constraints: scheduling, facilities location, assembly line balancing, project selection, menu planning, catalogue space planning, and school time tabling. The algorithm we consider in this report can be used for linear objective functions; however, our emphasis is on problems with separable or factorable objective functions with linear and nonlinear terms. Many of the linear applications above might be extended to include nonlinear terms in the objective function, particularly in the areas of scheduling, facilities location, capital budgeting, and portfolio selection. In addition, we are not restricted to problems where all variables occur in some multiple-choice constraint.

The problem we investigate has the following form:

$$\text{max } Z = \sum_{j=1}^{k} c_j x_j + \sum_{j=k+1}^{n} f(x_j)$$

subject to

$$\sum_{j=1}^{n_1} x_j = 1$$

$$\sum_{j=n_1+1}^{n_2} x_j = 1$$

$$\vdots$$

$$\sum_{j=n_{l-1}+1}^{n_l} x_j = 1$$

$$\sum_{j=n_{l+1}}^{n} c_j x_j + \sum_{j=n+1}^{n} h(x_j) \leq b_i, \quad i = 1, \ldots, m$$

$$x_j = 0, 1 \quad \text{for } j = 1, \ldots, l$$

$$x_j \geq 0, \text{integer for } j = l + 1, \ldots, n.$$
of \( x_p \) that \( a_p \geq 0 \), and finally, that some feasible solution exists. We have assumed separability of the objective function. Therefore, the functions containing higher powers of the binary decision variables can be reduced to linear functions. Research has been done on problems similar to (MCNIP). Gallo et al.\[3\] have considered a singly constrained quadratic knapsack problem with binary variables. They report solution times of 1.23 CPU seconds for 30 binary variables with a branch and bound cutting plane approach, using upper planes. Their model differs from ours in having nonseparable terms in the objective function in addition to having only a single constraint. Another problem related to (MCNIP) is identified by Granot et al.\[4\] as a 0–1 positive polynomial problem (PP). PP has a linear objective function but polynomial constraints. A sequence of nested covering problems is solved, each of which is a relaxation of PP. Problems of 40 constraints and variables are solved with crossproduct constraint terms containing from 2 to 20 variables, the solution times seemingly to be related to the number of variables in each term.

A third recent study by Bean\[2\] investigates the multiple-choice integer linear problem (MCIP). We rewrite Bean's formulation as follows:

\[
\begin{align*}
\text{maximize } & Z = \sum_{j=1}^{n} c_j x_j \\
\text{subject to } & \sum_{j=1}^{n} a_i x_j \leq b_i, \quad i = 1, \ldots, m \\
& \sum_{j \in N_q} x_j = 1, \quad q = 1, \ldots, p \\
& \bigcup_{q=1}^{p} N_q = \{1, 2, \ldots, n\} \\
& \bigcap_{q=1}^{p} N_q = \emptyset \\
& x_j = 0, 1 \quad \text{for } j = 1, \ldots, n.
\end{align*}
\]

This is a closer relative of the problem under study. Bean uses a branch and bound algorithm where large parts of the branch and bound tree are eliminated because they are infeasible for the multiple-choice constraints. He gives computational results for two problems with network constraints.

A special case of (MCNIP) is the multiple-choice knapsack problem. Sinha and Zoltners\[5\] report very fast solution times for this single constraint knapsack problem by using dominance rules for eliminating and ordering variables within a branch and bound framework.

We undertook this review to try to identify solution times for (MCNIP), (MCIP) or closely related problems reported in the literature. However, the references above vary widely in the size of problems attempted, and in addition in assuming just a single constraint or other special structure, such as unimodularity in the linear constraints of (MCIP), and the times reported vary widely. We feel that these differences prevent any useful comparison with our method.

**The Algorithm**

We present a dynamic programming approach with forward pass fathoming by bounds and feasibility to reduce storage and to improve computation time. We identify all feasible integer points of the following problem:

\[
\begin{align*}
\text{maximize } & Z = \sum_{j=1}^{l} c_j x_j + \sum_{j=l+1}^{n} f_j(x_j)
\end{align*}
\]
subject to

\[ \sum_{j=1}^{i} c_j x_j + \sum_{j=i+1}^{n} f_j(x_j) \leq Z_0 \quad x_j \geq 0, \quad \text{integer, } j = 1, \ldots, n, \]

where \( Z_0 \) is an upper bound on the objective function. This is a single-constraint problem which is easily solved. Its purpose is not to find \( Z(=Z_0) \), but to identify all integer points for various levels of the hypersurface \( Z \). A previously reported algorithm by Cooper and Cooper[6] then sorted the integer points in nonincreasing order of their objective function value, and tested each point in the same order for feasibility. (Using that strategy, the first feasible point is optimal.)

In the algorithm presented here, fathoming rules are used to eliminate all points which are infeasible for multiple-choice constraints and linear constraints. Therefore, for (MCNIP) the fathomed dynamic programming tables will include only feasible integer points. The one with the best value of the objective function is optimal. Additional logic may be used to carry out the following procedure with nonlinear, even nonseparable constraints[7].

Recent work on the portfolio problems by Cooper and Farhangian[8] has improved computation time by using theory developed for shortest path problems which can be applied by analogy to dynamic programming. These ideas are attributed to Denardo and Fox[9] in a survey paper by Morin[10]. Each achievable value of the state variables is considered analogous to a node in a shortest path network. Rather than the usual method of calculating dynamic programming recursion functions (called “pulling”), a label setting procedure is used (“reaching”). The reaching calculation labels each node with its predecessor on the best “path” so far to that node. It turns out that only replacing the pulling calculation by reaching gives no improvement in calculation time. However, if reaching is used, values can be removed or fathomed from the tables as they are generated in each stage. The fathoming is done using bounds as in the branch and bound method for integer linear programming. An additional type of fathoming is due to testing partial solutions for infeasibility with constraints of (MCNIP). If the r.h.s. value of any of the original constraints of (MCNIP) is exceeded, that partial solution and implicitly any of its completions will be dropped from the tables. This is the key element which cuts down storage and computation time.

**FATHOMING BY BOUNDS**

A heuristic method for linear integer programmings by Kochenberger, McCarl and Wyman [11] has been extended by us to separable nonlinear objective functions and multiple-choice constraints and used to generate a good heuristic solution for MCNIP. The procedure calculates an “effective gradient” for each variable, and increments the variable with the greatest effective gradient. The resulting trial solution is then tested for feasibility; if infeasible, the variable is reduced to its previous value and the variable with the next best gradient estimate is incremented. If no feasible improvements can be obtained in this way, the algorithm halts. This point has been used to fathom by bounds as the dynamic programming problem tables are generated.

**COMPUTATIONAL RESULTS**

The following table gives solution times for randomly generated problems in CPU seconds on a CDC 6600.

**CONCLUSIONS**

Computational results have been presented for random nonlinear integer problems with multiple-choice and additional linking constraints. The variables include binary and
general integer variables and the objective functions and constraints may contain nonlinear terms.

The structure of the multiple-choice constraints has allowed improvements in the computation time over a problem with general linear constraints ([8] 1982b). The time is on the order of 1/6 previous time for 20 variable problems.

REFERENCES


