ACTA MATHEMATICA VIETNAMICA
TOM 4, № 1 (1979)

SOLVING AN INTEGER PROGRAMMING PROBLEM

HOANG HAI HOC

Department of Electrical Engineering Ecole Polytechnique Montreal, Canada.

Introduction

We consider the following family of zero-one integer programming problems *(IP)*

$$\underset{x_1,\ldots,x_n}{\text{Minimize}} \quad \underset{i=1,\ldots,m}{\text{maximize}} \quad \left(d_i - \sum_{j=1}^{n} a_{ij}x_j\right)$$

subject to

$$\sum_{j=1}^{n} c_j x_j \leq b; \quad x_j = 0 \text{ or } 1,\ j = 1,\ldots,n$$

where $m = 1,2,\ldots$.

These problems can be viewed as nonlinear zero-one knapsack problems with nonadditive objective functions. One needs to consider this sequence of problems when one solves a nonlinear mixed integer programming model for optimal network topologies (10) using generalized Benders decomposition (3). However this is not a peculiar case. In fact, the application of generalized Benders decomposition to any linear (nonlinear) mixed integer programming model always gives rise to a similar sequence of master problems, each of which is obtained from the previous one by adding one or more constraints. Moreover, although Benders decomposition principle is not a complete algorithm, in the sense that it does not specify how to solve either master programs or subprograms, it was successfully used to solve practical problems such as electric power generation planning (11), multicommodity distribution system design (6), etc.

The major conclusion arising from these previous studies (6, 10, 11) is the remarkable effectiveness of Benders decomposition as a computational strategy for the classes of mixed integer programming models reported. The numerical experience shows that only a few (less than a dozen) cuts are needed to find and verify a solution within one percent of the global optimum. Computational time is then determined completely by the efficiency of algorithms used to solve sequences of master problems and subproblems. For this reason, we are presently interested in investigating computational techniques suited to a sequence of master problems, in particular the family of problems *(IP)*. However, we shall restrict ourselves to the class of enumerative techniques for its effectiveness as a practical tool to solve combinatorial problems

We note that the members of the family of problems (*IP*) are closely related. For $m = 2,3,...$ the m-th problem is actually obtained by adding to the $(m-1)$-th problem a new Benders cut. One can then examine the family of problems (*IP*) from the viewpoint of parametric and postoptimality analysis in integer linear programming (7). We propose to solve the family of problems (*IP*) employing implicit enumeration based algorithms for postoptimizing zero—one programs (12, 14). Computational experience performed on the family of problems (*IP*) should then shed more insight into postoptimality analysis by implicit enumeration.

Implicit Enumeration

This section presents an implicit enumeration which solves a problem (*IP*) and gathers the information required for postoptimizing it. For this purpose let us define the equivalent problem (*IPE*) as follows.

Minimize y

subject to

$$\sum_{j=1}^{n} -c_j x_j \leq B \tag{1}$$

$$D_i + \sum_{j=1}^{m} a_{ij} x_j \leq y, \; i = 1, 2,...,m \tag{2}$$

$$x_j = 0 \text{ or } 1, \; j = 1, 2,..., n \tag{3}$$

where

$$B = b - \sum_{j=1}^{n} c_j < 0$$

$$D_i = d_i - \sum_{j=1}^{n} a_{ij} > 0, \; i = 1,..., m$$

and all the coefficients a_{ij}, c_j, and d_i are positive.

If we take only constraints (3) into consideration there are 2^n possible assignments of values to $(x_1, x_2,..., x_n)$. Let an assignment of values to a subset of these variables be called a partial solution. Let the variables x_j not assigned values be called free variables. Any assignment of values to the free variables is called a completion of the associated partial solution. In particular, each partial solution has its zero-completion.

As the implicit enumeration search proceeds partial solutions are generated in an attempt to find a feasible zero-completion, since nonzero - completion of a feasible partial solution (i.e. a partial solution with feasible zero-completion) cannot have a better objective function value.

A partial solution is fathomed (a) if its zero-completion is feasible, or (b) if it can be shown that none of its completions can yield a feasible solution better than the best feasible solution found to date. Each fathomed partial solution of problem (*IPE*) with infeasible zero-completion will then be classified in two ways: (a) by the single resource constraint (1), and (b) by the

objective function constraints (2). For the purpose of this paper, only partial solutions obtained during the search, which are feasible or fathomed by objective function constraints need be saved for postoptimization. Consequently, let us define

$A_0 = \{s|$ the zero-completion of s is infeasible, and one or more constraints (2) fathom s $\}$

$A_f = \{s|$ the zero-completion of s is feasible $\}$

and associate the following index sets with each partial solution s obtained during the search

$$J_s^d = \{j \in N : x_j(s) = d\}, \; d = 0,1$$

$$N_s = N - J_s^0 - J_s^1$$

where $N = \{1,2,\ldots n\}$ and $x_j(s) = d$ implies that the j variable is assigned value d in the partial solution s. Hence, J_s^0 and J_s^1 are index sets for fixed variables and N_s is the index set of free variables with respect to s.

Algorithm I, which follows immediately, may be used to classify and collect partial solutions while solving problem *(IPE)*. Certain details are omitted. Termination criteria, rules for generating new partial solutions, and rules for backtracking from old ones, are not given. Any methods in (1, 4, 5, 9) can be used. Moreover, we denote by $\overline{Z}$ a valid upper bound on the optimal solution. This bound is continually updated during the search.

ALGORITHM I

1. Initialize $N_s = N$, $p = 0$, $A_f = A_0 = J_s^0 = J_s^1 = \varnothing$

2. Compute $t_0(s) = B + \sum_{j \in J_s^1} c_j$

$$v_0(s) = t_0(s) + \sum_{j \in N_s} c_j = b - \sum_{j \in J_s^0} c_j$$

$$v_i(s) = \overline{Z} - D_i - \sum_{j \in J_s^1} a_{ij}, \; i = 1,2,\ldots,m$$

3. If $t_0(s) < 0$, go to 4. Otherwise, s is feasible with objective function value $Z(s) = \overline{Z}$ − maximum $(v_i(s), i = 1,\ldots, m)$, set $p = p + 1$, add s to A_f. If $Z(s) < \overline{Z}$, set $\overline{Z} = Z(s)$. Backtrack, and go to 2.

4. For each free variable j in N_s define $G^0(j)$ and $G^1(j)$ as follows:

(i) If $v_0(s) < c_j$ then $0 \in G^1(j)$

(ii) For $i = 1,2...,m$, if $v_i(s) < a_{ij}$ then $i \in G^0(j)$ and perform the following steps:

(a) If $G^0(\bar{j})$ and $G^1(\bar{j})$, $\bar{j} \in N_s$, are nonempty then there are no better feasible completions of s. Add $\bar{s}$ to A_0, where $J^0_{\bar{s}} = J^0_s$ and $J^1_{\bar{s}} = J^1_s \cup \{\bar{j}\}$. Backtrack, and go to 2.

(b) If $G^1(j)$ *is* nonempty then skip to partial solutions $\bar{s}$, where $J^0_{\bar{s}} = J^0_s$ and $J^1_{\bar{s}} = J^1_s \cup \{\bar{j}\}$, and go to 2.

(c) If $G^1(\bar{j})$ is nonempty then add $\bar{s}$ to A_0, where $J^0_{\bar{s}} = J^0_s$ and $J^1_{\bar{s}} = J^1_s \cup \{\bar{j}\}$. Skip to partial solution s^*, where $J^0_{s^*} = J^0_s \; u \; \{\bar{j}\}$, $J^1_{s^*} = J^1_s$, and go to 2.

5. Generate, and go to 2.

Postoptimization using Implicit Enumeration

It has been shown by Roodman (14) that useful postoptimization capabilities for the zoro-one integer programming problem can be obtained from an implicit enumeration algorithm modified to classify and collect fathomed partial solutions such as Algorithm I of the previous section. The underlying principle is as follows: Whenever search along a branch of the enumeration tree is terminated, the fathomed partial solution s can be attributed to a constant k. Unless constraint k is somehow relaxed, s and its completions will remain infeasible, regardless of other changes to the 0-1 integer program. By considering the set of A_k of all partial solutions attributed to k, one can obtain the minimum relaxation in k before any partial solution in A_k becomes potentially feasible. Only partial solutions in A_k need be examined if one relaxes constraint k. More recently, Piper and Zoltners (12) presented a storage structure to cope with difficult data collection task inherent to the approach, and a set of algorithms using this storage structure to postoptimize after one or more problem parameter changes.

Let us consider now the family of probiems (*IPE*) as a sequence of closely related problems. Each problem in this sequence is obtained from the previous one by adding a new constraint (Benders cut). This added constraint is actually not verified by the previous optimal solution, and it is always necessary to reoptimize to reoptimize the problem after adding a constraint. To carry out this reoptimization, we used the approach suggested by Roodman, and refined by Piper and Zoltners. Conceptually, we obtained the following procedure:

ALGORITM II

1. Update z(s) for all pratial solutions s in A_f and A_0.
   Determine the best solution $\bar{s}$ in A_f.
   Let $\bar{x} = x(\bar{s})$, $\bar{z} = z(\bar{s})$.
2. For each partial solution s in A_0 with $z(s) < \bar{z}$, perform the following steps:

   (a) Let $A_0 = A_0 - \{s\}$.

(b) Examine the completions of s using Algorithm I.

We remark that when Algorithm I is used, during postoptimization, to examine the completions of partial solution s then J_S^0, J_S^1, N_S describe s; and p, A_0, and A_f describe the current state of the postoptimization. Step 1 of the Algorithm I should then be modified appropriately.

ALGORITM IMPLEMENTATION AND COMPUTATIONAL RESULTS

Algorithms I and II were implemented in order to study the computational behavior of implicit enumeration based algorithms for reoptimizing zero-one programs. No particular attention was given to the efficiency of the resulting computer programs.

The computer program implementing Algorithm I represents essentially a variation of the additive algorithm (1) specialized to problem (*IPF*) with an efficient bookeeping scheme (4,5) which keeps track of the enumeration. This variation includes only ceiling test and cancellation tests. As branching strategy it is chosen to fix at 1 the free variable j for which the ratio a_{mj}/c_j is minimum.

Although Algorithm II is very simple conceptually, its implementation presents some interesting problems from the viewpoint of storage structure. First, to accomodate problems of about 30 variables each partial solution is packed into two 32 bit words. There are two bits attributed to each variable: one bit is used to indiacate that the variable is free or fixed, and the other represents the fixed value of the variable. Second, the set of partial solutions A_0 and A_f are too large to be stored in core. Random access disc files are used to save A_0 and A_f. Moreover, the set A_0 is actually subdivided into $A_0^1, A_0^2, \ldots, A_0^q$ according to the objective function values of partial solutions callected. For all partial solutions belonging to A_0^i the objective function values fall inside the i-th predetermined interval. Partial solutions in A_0 and A_f are stored sequentially in records which are chained by pointers.

In order to facilitate file processing a sufficiently large array is used as buffer, and organized into LIFO sublists. Each sublist contains partial solutions belonging to a subset A_0^i or to the set A_f, and residing temporarily in core. When this buffer array is full, and at the end of the reoptimization, all partial solutions are transferred on disc files. The output file obtained at the end of a reoptimization is used as input file for the next reoptimization.

All programs were written in FORTRAN IV Level G, and executed under the control of OS/360 MVT, on a system IBM/360 Model 75. Several families of test problems whose sizes range from 10 to 30 variables were stored. First we solved each problem in a family separately. Then we solved each family of problems as a sequence of related problems using reoptimization technique. Each

family consists of 5-8 problems. Computational results indicate that the time required to solve a family of problems using reoptimization technique is actually 2-3 times longer than the total time required to solve the problems separately. This is partially due to the time consuming operations of packing and unparking partial solutions (by means of integer arithmetic in FORTRAN IV) as well as the inefficient file processing procedure presently employed. Furthermore, an important explanatory factor is the continually increasing number of partial solutions collected and re-examined while reoptimizing. In fact, as reoptimization proceeds, one is going down the enumeration tree deeper and deeper, and generating more and more nodes which will eventually be fathomed, saved and re-examined. Hence, one should be prevented from going down the tree too deeply by means of efficient bounding process using embedded linear programs, surrogate constraints, etc. This represents a difficulty common to all implicit enumeration, and branch and bound algorithms. Last, the reoptimization technique considered will be efficient only if the part of the enumeration tree to be explored for solving the modified problems is not very different from the part of the tree explored while solving the original problem. This may be the case, if only a few coefficients of the zero-one integer program are subject to changes. Extensive computational experience is planned to study these aspects.

REFERENCES

1. E. Balas, *An Aditive Algorithm for Solving Linear Programs with Zero-One Variables*, Operations Research, Vol. 13, pp. 517-546 (1965).

2. J. F. Benders, *Partioning Procedures for Solving Mixed Variables Programming Problems*, Numerische Mathematik, Vol. 4, pp. 238-252 (1962).

3. A. M. Geoffrion, *Generalized Benders Decomposition*, Journal of Optimization Theory and its Applications, Vol. 10, pp. 237-260 (1972).

4. A. M. Geoffrion, *Integer Programming by Implicit Enumeration and Balas Method*, SIAM Review, Vol. 9, pp. 178-190 (1967).

5. A. M. Geoffrion, *An Improved Implicit Enumeration Approach for Integer Programming*, Operations Research, Vol. 17, pp. 437-454 (1969).

6. A M. Geoffrion and G. W. Graves, *Multicommodity Distribution System Design by Benders Decomposition*, Management Science, Vol. 20, pp. 822-844 (1974).

7. A. M. Geoffrion and R. Nauss, *Parametric and Postoptimality Analisis in Integer Linear Programming*, Management Science, Vol. 23, pp. 453-466 (1977).

8. F. Glover, *Surrogate Constraints*, Operations Research, Vol. 16, pp. 741-749 (1968)

9. F. Glover, *A Multiphase Dual Algorithm for the Zero-One Integer Programming Problem*, Operations Research, Vol. 13, pp. 879-919 (1965).

10. Hoang Hai Hoc, *Network Improvements via Mathematical Programming*, Paper presented at the IX th International symposium on Mathematical Programming, Budapest, August 23-27, 1976.

11. F. Noonan and R. J. Giglio, *Planning Electric Power Generation : A Nonlinear Mixed Integer Programming Model Employing Benders Decomposition»*, Management Science, Vol. 23, pp. 946-956 (1977).

12. C. J. Pipe and A. A. Zoltners, *Implicit Enumeration Based Algorithms for Postoptimizing Zero-One Programs*, Naval Research Logistics Quarterly, Vol. 22, pp. 791-809 (1975)

13. C. J. Piper and A. A. Zoltners, *Some Easy Postoptimality Analysis for Zero-One Programming*, Management Science, Vol. 22, pp. 759 - 765 (1976).

14. G. M. Roodman, *Postoptimality Analysis in Zero-One Programming by Implicit Enumeration*, Naval Research Logistics Quarterly. Vol. 19, pp. 435-447 (1972).