Journal of Global Optimization **5**: 277–290, 1994.
© 1994 *Kluwer Academic Publishers. Printed in the Netherlands.*

An Ordering (Enumerative) Algorithm for Nonlinear 0-1 Programming

BÉLA VIZVÁRI[1] and FATIH YILMAZ[2]
[1]*Department of Industrial Engineering, Bilkent University, Ankara; current address: Rudgers University RUTCOR, P.O. Box 5062, NJ 08903-5062, U.S.A., vizvari@rutcor.rutgers.edu;* [2]*Dept. of Industrial Engineering, Bilkent University, 06533 Bilkent, Ankara, Turkey, yilmaz@trbilun.bitnet*

(Received: 12 July 1993; accepted: 11 February 1994)

Abstract. In this paper, a new algorithm to solve a general 0–1 programming problem with linear objective function is developed. Computational experiences are carried out on problems where the constraints are inequalities on polynomials. The solution of the original problem is equivalent with the solution of a sequence of set packing problems with special constraint sets. The solution of these set packing problems is equivalent with the ordering of the binary vectors according to their objective function value. An algorithm is developed to generate this order in a dynamic way. The main tool of the algorithm is a tree which represents the desired order of the generated binary vectors. The method can be applied to the multi-knapsack type nonlinear 0–1 programming problem. Large problems of this type up to 500 variables have been solved.

Key words: Nonlinear integer programming, enumeration, lexicographic order, global optimum.

1. Introduction

The general 0–1 programming problem with linear objective function can be stated as:

$$\begin{aligned} &\max \quad cx \\ &x \in S \subseteq \{0,1\}^n \end{aligned} \tag{GBP}$$

where $c \in N_+^n$ and without loss of generality it is assumed in the whole paper that

$$c_1 \geq c_2 \geq \cdots \geq c_n > 0. \tag{1}$$

It is well-known that (GBP) is a hard problem to solve if S has no special properties. A new algorithm to solve (GBP) is presented in this paper. The algorithm is tested on problems where the set S is defined with polynomial inequalities. Experiences show that the method is competitive, i.e. problems have been solved with at least the same size as those reported in [1] and [5]. The method can be applied to the multi-knapsack type nonlinear 0–1 programming problem which is the following

$$\begin{aligned} &\max \sum_{i=1}^{p} c_i \prod_{j \in S_i} x_j \\ &\sum_{k=1}^{n_i} a_{ik} \prod_{j \in S_{ik}} x_j \leq b_i \quad i = 1, \ldots, m \\ &x_j \in \{0, 1\} \quad j = 1, \ldots, n, \end{aligned} \tag{MKTNL}$$

where it is assumed that all coefficients are positive and $S_i, S_{ik} \subseteq \{1, ..., n\}$. Large problems of this type have been solved exactly by the method described in Section 4. [5] has collected a lot of applications of nonlinear 0–1 programming. In many cases the underlying mathematical problem is (GBP). In some applications, e.g., in testing vision loss discussed in [6], this problem is (MKTNL).

The new method is an enumerative algorithm. Enumeration is a well-known technique to solve NP-hard problems. This type of algorithm can be found, e.g., in [2], [4], [7], [12], [14], [15] and [16].

The organization of the paper is as follows. Section 2 describes a solution scheme, which was applied earlier in several papers. A basic observation is presented in the same section. The tree representation of the ordering of binary vectors, which is the main algorithmic tool, is discussed in Section 3. The application of the algorithm to the multi-knapsack type nonlinear 0–1 programming problem can be found in Section 5. Finally the last section discusses the computation experiences.

2. Solution of GBP via a Sequence of Set Packing Problems

In [4] and [7], the following scheme is applied to solve Problem (GBP).

If $e \in S$, then it is optimal where e is the all 1 vector. If $e \notin S$, then the optimal solution of (GBP), if it exists, must satisfy the following inequality

$$\sum_{j=1}^{n} x_j \leq n - 1.$$

Let us consider the following problem :

$$\begin{aligned} Z_2 = {} & \max\ cx \\ & \textstyle\sum_{j=1}^{n} x_j \leq n - 1 \\ & x \in \{0, 1\}^n \end{aligned} \tag{2}$$

and assume that optimal solution of (2) is β_2. It can be seen that

$$\beta_{2j} = \begin{cases} 1 & \text{if } \ j \in \{1, ..., n-1\} \\ 0 & \text{if } \ j = n. \end{cases}$$

Then, it is obvious that if $\beta_2 \in S$, so it is an optimal solution of (GBP). Now assume that $\beta_2 \notin S$, it follows that the optimal solution of original problem must satisfy

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

$$\beta_2 x \leq \beta_2 e - 1$$

$$x \in \{0, 1\}^n.$$

If we continue this procedure, the following generalization is obtained.

Let $\beta_1 = e,\ \beta_2,\ ...,\beta_k\ \ (2 \le k \le 2^n)$ be the vectors generated so far and none of them was feasible in (GBP). Then the following set packing problem

$$\begin{aligned} Z_{k+1} &= \max\ cx \\ x \in P_k = &\ \{x \in \{0,1\}^n\ :\ \beta_l x \le \beta_l e - 1,\ l = 1, ..., k\} \end{aligned} \qquad \text{(SPP)}$$

is solved in the next iteration and its optimal solution is denoted by β_{k+1}.

LEMMA 1. *Assume that $\beta_1, ..., \beta_k \notin S$. Then $S \subseteq P_k$.*

Proof. Let us assume that $y \in S,\ y \notin \{\beta_1, ..., \beta_k\},$ *but* $y \notin P_k$. Consider the following index r such that

$$r\ =\ \min\{l\ :\ \beta_l y\ >\ \beta_l e - 1;\ \ 1 \le l \le k\}.$$

It is obvious that $r \ge 2$. Otherwise, $r = 1,\ y = \beta_1$. It follows that

$$\beta_r y = \beta_r e$$

i.e.,

$$y \ge \beta_r\ \text{ and }\ y \ne \beta_r.$$

Hence, it follows from (1) that

$$cy\ >\ c\beta_r.$$

This is a contradiction to optimal property of β_r, because $y \in P_{r-1}$. □

Thus (SPP) is a relaxation of (GBP).

THEOREM 1. *Let k be the minimal index of iterations, such that $\beta_{k+1} \in S$. Then it is optimal to (GBP).*

Proof. It follows from the previous lemma. □

A similar model is given in [8] for the matrix equipartition problem, but in that case it is possible to exclude not only the current optimal solution.

Let L be any list of all binary vectors. If the objective function values of the vectors form a decreasing order in L then the first vector of L which is feasible in (GBP), is an optimal solution.

It was Boros [3] who made the following observation. As it is stated in Lemma 1, the constraint set of the current set packing problem excludes only the points, generated so far. Therefore the optimal solution of the next set packing problem is the binary vector having the greatest objective function value among those being not generated so far. Thus the algorithm discussed above generates the first part of such a list L up to the first feasible solution.

3. The Tree Representation of the Ordering of the Binary Vectors

In this section a method is provided to arrange the binary vectors in such a way that the values of the linear form cx are in a decreasing order. The method is based on two well-known notions which are successor of a binary vector and search tree. The first one is used, e.g., in [9] in two algorithms to generate all k subsets of a set of n element. In the algorithm of this paper a similar subproblem occours, but the algorithms of [9] cannot be used here because of the special requirement that the enumeration must be done in a decreasing order of the value of the objective function. Search trees are discussed in [13]. [13] defines four operations on search trees which are the followings: insertion and deletion of an object and to join two trees and to split one tree into two trees. Here only insertion and a special kind of deletion are needed. [13] does not provide a pseudo-code of the algorithm. Therefore all details of the method are discussed here.

First, the following subproblem is solved, in which exactly k components of the vectors x have the value 1, i.e.

$$\sum_{j=1}^{n} x_j = k, \tag{3}$$

are ordered. Let u and v be two binary vectors satisfying (3). Let $\{i_1, i_2, ..., i_k\}$ and $\{j_1, j_2, ..., j_k\}$, resp., the set of indices of the 1's in the vector u and v, resp. It follows immediately from (1) that if

$$i_p \leq j_p, \quad p = 1, \cdots, k \tag{4}$$

then $cu \geq cv$.

DEFINITION 1. Let $\{i_1, ..., i_k\}$ be the set of indices of 1's in the binary vector u. Assume that for some index p the inequality

$$i_p + 1 < i_{p+1}$$

holds, where $i_{k+1} = n + 1$. Let u' be the vector defined by

$$u'_j = \begin{cases} u_j & \textit{if } j \neq i_p, i_p + 1 \\ 0 & \textit{if } j = i_p \\ 1 & \textit{if } j = i_p + 1 \end{cases}$$

Then u' is an immediate successor of u.

It is obvious from previous remarks, that if u' is an immediate successor of u, then $cu \geq cu'$. Equality holds if $c_{i_p} = c_{i_p+1}$. One vector can have several immediate successors.

THEOREM 2. *Let $\{i_1, ..., i_k\}$ and $\{j_1, ..., j_k\}$, resp., be the sets of indices of the 1's in the binary vectors u and v, resp. Assume that (4) holds. Then there is a sequence of binary vectors*

$$u = w_0,\ w_1, ..., w_t = v \tag{5}$$

such that w_l is an immediate successor of w_{l-1} $(l = 1, ..., t)$.

Proof. Assume that $u \neq v$, otherwise $t = 0$ and the statement holds. Let

$$p = \max \{q \ : \ i_q \ < \ j_q\}.$$

Hence $u_{i_p+1} = 0$, otherwise

$$i_p + 1 = i_{p+1} \leq j_p \ < \ j_{p+1}$$

and this contradicts the maximal property of p. Let

$$w_{1j} = \begin{cases} w_{0j} & \text{if } j \neq i_p, i_p + 1 \\ 0 & \text{if } j = i_p \\ 1 & \text{if } j = i_p + 1 \end{cases}$$

Then the set of the indices of 1's in w_1 is

$$\{i_{11}, ..., i_{1k}\} = \{i_1, ..., i_{p-1}, i_p + 1, ..., i_k\}.$$

It is obvious that

$$i_{1l} \leq j_l, \quad l = 1, ..., k.$$

Then either $w_1 = v$, or the process can be repeated for w_1 □

It is trivial that all of the binary vectors containing exactly k 1's are the successors of the vector x, where $x_1 = ... = x_k = 1,\ x_{k+1} = ... = x_n = 0$. Therefore all of these vectors can be enumerated with the following algorithm, where L is the list of binary vectors to be enumerated and the vector e_j is the j-th unit vector.

Algorithm 1

```
1. Begin
2.    $L := \{\sum_{j=1}^{k} e_j \ = \ (1, ..., 1, 0, ...., 0)\}$;
3.    while $L \neq \emptyset$ do
4.    begin
5.       choose $u \in L$;
6.       $L := (L \cup \{v \ : \ v \text{ is an immediate successor of } u\}) \setminus \{u\}$;
7.    end;
8. end
```

Without any deeper organization this algorithm will work unnecessarily too much, because the sequence (5) is not unique, i.e., a binary vector can be generated several times as the immediate successor of different vectors. This phenomena is illustrated with the following example. Let $k = 2,\ n = 4$. Then Algorithm 1 starts from the point $x = (1, 1, 0, 0)$. In the first iteration it has only one immediate successor, which is $u_1 = (1, 0, 1, 0)$. The immediate successors of u_1 are $u_2 = (0, 1, 1, 0)$, and $u_3 = (1, 0, 0, 1)$. The only immediate successor of u_2 is $u_4 = (0, 1, 0, 1)$ which is at the same time the immediate successor of u_3, too. Thus L contains u_3 and u_4 after generating the successors of u_2. If in this situation u_4 is selected in the next iteration in Row 5, then it leaves L, but it will return as the successor of u_3, i.e., one vector may be enumerated several times. To avoid this disadvantageous effect the following ordering of binary vectors is introduced.

DEFINITION 2. The vector u is 'greater' than v, denoted $u \rhd v$ if either

- $cu > cv$ or
- $cu = cv$ and u is lexicographically greater than v which is denoted by $u \succ v$.

It is obvious that any two distinct vectors are comparable by this ordering, i.e. in any set of binary vectors there is a unique maximal element in this ordering. Thus the selection of the vector u in the 5-th row of Algorithm 1 can be executed in the following way

$$5' \quad u := \max \rhd \{v \in L\}.$$

In Row 6 the operation $\cup$ is the well-known set operation. This means that if a vector v is in L and at the same time among the successors of u, then after executing this command it will have only one copy in L. This is arranged in Rows 22–26 of Algorithm 3 (see below).

It follows from (1), that if u' is an immediate successor of u, then $u \rhd u'$. Hence the following statement is obtained.

LEMMA 2. *If in Algorithm 1 the selection of the vector u is done according to 5', then no vector can be selected twice.*

Proof. Assume that the vector u was selected in iteration t. Then it was the maximal vector in ordering $\rhd$ which was contained in L and it was substituted by a set of smaller vectors. If it returned to L then at least one vector had to be substituted by a greater vector, which is impossible. □

Hence the following algorithm is obtained to enumerate all of the binary vectors.

Algorithm 2

```
1. Begin
2.     $L := \{\sum_{j=1}^{k} e_j \ : \ k = 1, ..., n\}$;
```

```
3.    while $L \neq \emptyset$ do
4.    begin
5.       $u :=$ max $\rhd \{v \in L\}$;
6.       $L := (L \cup \{v \; : \; v \; is \; an \; immediate \; successor \; of \; u\}) \setminus \{u\}$;
7.    end;
8. end
```

THEOREM 3. *All of the binary vectors are selected in row 5 of Algorithm 2 exactly once.*

Proof. It follows from Lemma 2 that it is enough to prove that all of the points are selected at least once. Assume that the vector v containing exactly k 1's is not chosen. Let us consider a sequence (5) from the point $\sum_{j=1}^{k} e_j$ to v. Without loss of generality we may assume that in this sequence only v was not chosen. Then $w_t = v$ is an immediate successor of w_{t-1}, which was selected in an iteration. When in row 5 u was w_{t-1}, w_t entered to L according to Row 6. If v is in L and v has been never selected then L has become never empty. Thus it follows from Lemma 2 that it should be infinite many binary vectors being greater in the ordering $\rhd$ than v, which is a contradiction. □

For the sake of convenient handling of the list L, it is organized as a rooted tree. The root contains always the greatest point. All of the nodes of the tree, except the root, can have two children, a left and a right one. The root has only a left child. All of the binary vectors of the left (right) subtree of a node v are less (greater) than that of v according to the relation $\rhd$. Thus it is very easy to find the appropriate position of a newly generated binary vector in the tree. This is done by the Algorithm 3.

Algorithm 3

```
1. begin
2.    (* Let $L$ be the current tree *)
3.    (* $S :=$ Set of all immediate successor of $x$ *)
4.    $x :=$ Binary vector in the root;
5.    while $S \neq \emptyset$ do
6.    begin
7.       $y \in S$;
8.       $z := cy$;
9.       $p := root^{\wedge}.left$;
10.      $f := true$;
11.      while $p \neq nil$ and $f$ do
12.      begin
13.         $Temp := p$;
14.         if $z > p^{\wedge}.obfv$
15.         then $p := p^{\wedge}.rightnode$
16.         else if $z < p^{\wedge}.obfv$
```

```
17.                 then $p := p^\wedge.leftnode$
18.                 else if $y \succ p^\wedge.BVector$;
19.                     then $p := p^\wedge.rightnode$;
20.                     else if $p^\wedge.BVector \succ y$
21.                         then $p^\wedge.leftnode$
22.                         else
23.                         begin
24.                             $f := false$;
25.                             $S := S - \{y\}$;
26.                         end;
27.         end;
28.         if $f$
29.         then
30.         begin;
31.             $S := S - y$;
32.             if $z > Temp^\wedge.obfv$
33.             then $Temp^\wedge.rightnode := y$
34.             else $Temp^\wedge.leftnode := y$
35.         end;
36. end;
```

In Row 6 of Algorithm 2, the point contained in the root is omitted from the tree. Therefore a method is needed to find the point, which must go into the root, i.e., the maximal binary vector in the tree. But, notice that this is always the most right point of the (left) subtree of the root.

Algorithm 3 needs $O(log(n\sum_{j=1}^{n} c_j))$ operations. But in the practice it works far better. This order can be reached only if $c_n = 1$ and

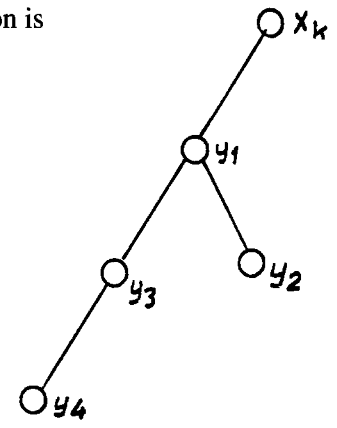
$$\forall j\ (1 \leq j < n) : c_j \ \leq \ \sum_{l=j+1}^{n} c_l \ + \ 1.$$

Example. Assume that $n = 6$, $c = (7,5,4,3,2,1)$ and the current tree L_k consists of the following binary vectors

$$x_k = (1,0,1,0,1,0),\ \ y_1 = (1,0,1,0,0,0),\ \ y_2 = (1,1,0,0,0,0),$$

$$y_3 = (0,1,0,1,1,0),\ \ y_4 = (1,0,0,0,0,0)$$

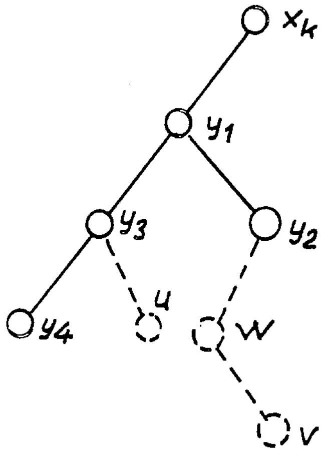
and the tree representation is



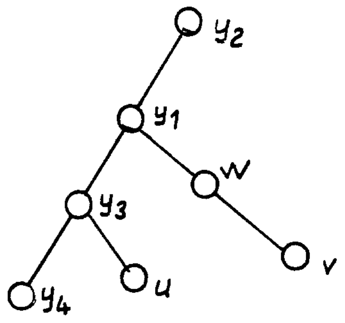
The immediate successors of x_k are

$$u = (0,1,1,0,1,0), \quad v = (1,0,1,0,0,1), \quad w = (1,0,0,1,1,0).$$

Then the tree becomes



The point y_2 goes into the root and the shape of tree is as follows



4. The Multi-Knapsack Type Nonlinear Problem

In this section the algorithm is applied to the multi-knapsack type nonlinear 0–1 programming problem.

Let us consider the products in the objective function. Let

$$y_i = \prod_{j \in S_i} x_j \quad i = 1, ..., p. \tag{6}$$

Thus $y_i \in \{0, 1\} (i = 1, ..., n)$. Equation (6) means that x uniquely determines y, but it is not true in the opposite direction. If y is given a priori then it is possible that (i) there is no x, (ii) there is exactly one x, (iii) there are several x's satisfying (6). Case (i) occurs if and only if there is an index k such that $y_k = 0$ and

$$S_k \subseteq \bigcup_{i: y_i = 1} S_i.$$

Thus

$$\begin{array}{l}\{ \sum_{i=1}^{p} c_i \prod_{j \in S_i} x_j \mid \forall j : \ x_j \in \{ 0, 1 \} \} \\ \quad \subseteq \{ \sum_{i=1}^{p} c_i y_i \mid \forall i : \ y_i \in \{ 0, 1 \} \}.\end{array} \tag{7}$$

If Case (i) can occur then equation in (7) is not necessarily true.

Let $\hat{y}$ be a fixed vector. Then it follows from its definition that for any vector x satisfying (6)

$$\forall j \in \bigcup_{i: \hat{y}_i = 1} S_i : \ x_j = 1 \tag{8}$$

holds. Then

$$\sum_{i=1}^{p} c_i \prod_{j \in S_i} x_j \geq \sum_{i=1}^{p} c_i \hat{y}_i. \tag{9}$$

for all vectors x satisfying (8). Among all these vectors $\hat{x}$ is giving the smallest value in the objective function with

$$\forall j \in \{ 1, ..., n \} \setminus \bigcup_{i: \hat{y}_i = 1} S_i : \ \hat{x}_j = 0. \tag{10}$$

The same is true for the right-hand sides of the constraints, because their coefficients are positive, as well. Thus if there is any feasible solution among the vectors satisfying (8), then $\hat{x}$ is one of them because all of the coefficients are positive.

The algorithm of the previous section is applied in the following way. The vectors y are enumerated. Relation (7) ensures that no value of the objective function of Problem (MKTNL) is omitted in this way. For each particular vector $\hat{y}$ the vector $\hat{x}$ satisfying (8) and (10) is determined. In this way $\hat{x}$ is uniquely determined and

$$\sum_{i=1}^{p} c_i \hat{y}_i \leq \sum_{i=1}^{p} c_i \prod_{j \in S_i} \hat{x}_j.$$

THEOREM 4. *Assume that the vectors y are enumerated in the decreasing order of the value of the objective function. To each particular $\hat{y}$ a vector $\hat{x}$ satisfying (8) and (10) is determined. Then the first feasible $\hat{x}$ is the optimal solution of Problem (MKTNL).*

Proof. The proof is based on the following observation. If there is strict inequality in (9) then the point $\hat{x}$, which satisfies (8) and (10) is infeasible. The strict inequality implies the existence of an index l such that

$$y_l = 0 \text{ and } S_l \subseteq \{ j : x_j = 1 \}.$$

Thus the same vector x appeared at the enumeration of the vector $y + e_l$. That time it has turned to be infeasible. Thus the first feasible x belongs to the greatest possible value of $\sum_{i=1}^{p} c_i y_i$. □

5. Computational Experiences

Balas and Mazzola [1] deal with the problem

$$\max \sum_{j=1}^{n} c_j x_j$$

$$\sum_{k=1}^{n_i} a_{ik} \prod_{j \in S_{ik}} x_j \leq b_i \quad i = 1, ..., m$$

$$\forall j : \quad x_j \in \{ 0, 1 \}$$

where $\forall i, k$ $S_{ik} \subseteq \{1, ..., n\}$, $\forall i : n_i \in \{1, ..., n\}$, $\forall j$ $c_j \geq 0$. It is well-known that all subsets of binary vectors can be described with the same type of constraints. Thus the only restriction of generality in the above-mentioned problem is that the objective function is linear.

The computational experiences provided by [1] include problems with the following parameters:

- $m = 10$,
- $n = 30$,
- $\forall i : n_i$ is drawn from a uniform distribution U[3,T], where T goes from 10 to 60,
- $\forall i, k : \mid S_{ik} \mid$ is drawn from the U[2,6] uniform distribution,
- $\forall i, k : a_{ik}$ is drawn from the U[-5,15] uniform distribution,
- $\forall i : b_i$ is drawn from the U[0.3 $\sum_{k=1}^{n_i} a_{ik}$, 0.8 $\sum_{k=1}^{n_i} a_{ik}$] uniform distribution,
- $\forall j : c_j$ is drawn from the U[1,20] uniform distribution.

The first table contains the computational experiences with that type of problems, which were obtained on an IBM/AT-486 with 16 Mbyte memory and 33 Mhz speed. The order of the columns are:

- m,
- n,

TABLE I. Experiences with Balas–Mazzola type problems

m	n	$\mid S_{ik} \mid$	n_i	N	UN	binary vectors			CPU		
						min	avg	max	min	avg	max
5	30	U[1,6]	U[3,10]	10	0	57	327	899	0.11	1.13	3.57
5	30	U[1,6]	U[3,20]	10	0	51	267	705	0.06	0.93	2.96
5	30	U[1,6]	U[3,30]	10	0	57	208	690	0.11	0.61	2.31
5	30	U[1,6]	U[3,40]	10	0	65	373	1048	0.11	1.90	7.58
5	30	U[1,6]	U[3,50]	10	0	61	228	768	0.11	0.85	1.92
5	30	U[1,6]	U[3,60]	10	0	74	324	956	0.17	1.41	2.80
5	40	U[1,6]	U[3,10]	10	0	41	665	1561	0.02	4.39	16.48
5	40	U[1,6]	U[3,20]	10	0	96	450	1981	0.22	3.49	24.61
5	40	U[1,6]	U[3,30]	10	0	86	945	3214	0.17	10.93	61.52
5	40	U[1,6]	U[3,40]	12	2	106	1101	2830	0.27	11.32	48.89
5	40	U[1,6]	U[3,50]	10	0	201	1234	3038	0.66	10.78	28.84
5	40	U[1,6]	U[3,60]	12	2	88	1465	3383	0.17	13.38	36.36
5	60	U[1,6]	U[3,10]	12	2	87	1128	2938	0.16	16.58	54.81
10	30	U[1,6]	U[3,10]	11	1	158	946	3174	0.44	5.79	20.81
10	30	U[1,6]	U[3,20]	11	1	117	415	817	0.27	1.52	3.96
10	30	U[1,6]	U[3,30]	10	0	94	853	2729	0.22	5.33	25.27
10	30	U[1,6]	U[3,40]	10	0	221	1531	3078	0.66	12.39	28.40
10	30	U[1,6]	U[3,50]	11	1	114	830	2609	0.33	5.52	23.18
10	30	U[1,6]	U[3,60]	10	0	59	612	1719	0.11	3.54	11.81
10	40	U[1,6]	U[3,10]	14	3	614	2048	4646	2.18	20.22	59.81
10	40	U[1,6]	U[3,20]	13	3	524	2212	4448	1.98	18.59	46.69
10	40	U[1,6]	U[3,30]	13	3	175	1667	4203	0.39	16.47	56.17
10	40	U[1,6]	U[3,40]	15	5	683	1900	3393	3.02	18.22	56.30
10	40	U[1,6]	U[3,50]	14	4	198	1950	4451	0.66	23.38	69.64

- the distribution of $\mid S_{ik} \mid$,
- the distribution of n_i,
- N = the number of problems,
- UN = the number of problems not solved, because of lack of memory,
- the number of the generated binary vectors of the solved problems: minimum, average, maximum,
- CPU times of the solved problems in seconds: minimum, average, maximum.

In these problems the generation of the a_{ik}'s and b_i's and c_j's was exactly the same as that in [1].

The problem generation of [1] differs from that of the problems contained in Table I, that $\mid S_{ik} \mid$ is drawn from the U[2,6] distribution instead of U[1,6]. But this makes the problems more difficult for the method described by the present paper. Table II provides evidences for the claim that *if the nonlinearity of the problem increases, i.e.* $\mid S_{ik} \mid$ *increases, than the problem becomes easier for the method*

TABLE II. Experiences with more nonlinear problems

m	n	$\mid S_{ik} \mid$	n_i	N	UN	binary vectors			CPU		
						min	avg	max	min	avg	max
10	30	U[6,12]	U[3,10]	10	0	189	547	2006	0.38	1.90	10.28
10	40	U[6,12]	U[3,10]	11	1	91	517	1272	0.11	2.29	6.97
10	50	U[6,12]	U[3,10]	11	1	67	1180	2597	0.06	15.48	64.91
10	60	U[6,12]	U[3,10]	11	1	203	830	2230	0.60	7.15	34.33
10	30	U[12,18]	U[3,10]	10	0	33	93	247	0.01	0.55	23.18
10	40	U[12,18]	U[3,10]	10	0	73	205	683	0.05	0.52	2.46
10	50	U[12,18]	U[3,10]	10	0	64	254	1240	0.06	1.47	11.70
10	60	U[12,18]	U[3,10]	10	0	77	480	1469	0.06	4.11	17.63

TABLE III. Experiences with nonlinear problems in Sun station

m	n	p	$\mid S_k \mid$	n_i	$\mid S_{ik} \mid$	binary vectors			CPU		
						min	avg	max	min	avg	max
10	80	20	10	U[3,10]	U[1,12]	37	390	1152	0.07	1.60	5.93
10	150	20	10	U[3,10]	U[1,12]	17	96	215	0.01	0.26	0.84
10	200	30	10	U[3,10]	U[1,12]	29	86	434	0.03	1.15	7.80
10	80	25	10	U[3,10]	U[1,12]	68	484	1346	0.62	5.95	21.13
10	150	25	10	U[3,10]	U[1,12]	14	91	482	0.03	0.79	5.55
20	150	20	10	U[3,10]	U[1,12]	19	52	166	0.02	0.32	1.10
30	250	25	10	U[3,10]	U[1,12]	24	45	121	0.03	0.92	1.96
30	250	40	10	U[3,10]	U[1,12]	41	476	2244	0.10	6.98	35.85
40	250	30	10	U[3,10]	U[1,12]	29	108	466	0.05	1.35	7.75
40	300	50	10	U[3,30]	U[1,12]	39	231	1330	0.05	4.49	34.95
30	250	40	15	U[3,10]	U[1,12]	79	562	2012	0.62	7.76	33.27
40	250	30	15	U[3,10]	U[1,12]	28	150	625	0.01	1.91	8.12
40	300	50	10	U[3,30]	U[1,12]	49	405	2405	0.08	7.60	43.15
10	500	50	20	U[3,30]	U[1,12]	49	324	1112	0.10	6.98	28.17
20	500	50	20	U[3,30]	U[1,12]	51	418	1837	0.17	12.07	65.92

of this paper. The explanation is the following. The value of a product is zero if at least one term in the product is zero. If the products are containing more terms, then one zero value of a variable makes more products to be zero, and therefore it is easier to satisfy the inequalities. Notice that the generation of the a_{ik} numbers in [1] is asymmetric, because they are drawn from the U[-5,15] distribution, i.e. they are more likely positive than negative. Because of the same reason the right-hand sides, i.e., the numbers b_i, are positive in almost all of the cases.

The feasibility of the generated problems is not guaranteed. If a problem is infeasible then the method of this paper enumerates all of the 2^n binary vectors. Practically it means that the program runs out of memory.

[5] reconstructed the results of [1] and tested some other algorithms and provided new methods. The size of the problems is mostly of 10 constraints and 30 variables. In some cases the greatest size is 20 constraints and 30 and 50 variables. The number of terms per constraint was drawn from U[30,40]. Another classes with parameters 50/10/U[50,60] and 30/10/U[3,200] have been investigated, too.

Because the key factor is the capacity of the memory and not the speed of the machine, the further experiences with larger nonlinear problems have been carried out on Sun stations. The memory of the used Sparc 490 is 64 Mbyte and the virtual memory is 4 Gbyte. Table III contains the computational experiences with (MKTNL) problems. In that case the zero vector is always a feasible solution because the right-hand sides are nonnegative. In the Sun environment all of the 10 problems in each class have been solved. Therefore columns N and UN are omitted. The number of nonzero coefficients in the objective function is denoted by p and $| S_k |$ is the number of variables in the products of the objective function. The CPU time is given in seconds.

References

1. Balas, E. and Mazzola, J. B., (1984) Nonlinear 0–1 Programming: II. Dominance Relations and Algorithms, *Mathematical Programming*, **30**, 22–45.
2. Beresnev, V.L. (1979), Algorithms for the minimization of polynomials with Boolean variables (Russian), *Problemy Kibernetiki (Moscow)* 36, 225–246.
3. Boros, E. (1985), Private communication.
4. Granot, D., Granot, F., and Kallberg, J. (1979), Converting Relaxation for Positive 0–1 Polynomial Programs, *Mng. Sci.* **25**, 264–273.
5. Hansen, P., Jaumard, B., and Mathon, V. (1989), Constrained Nonlinear 0–1 Programming, RUTCOR Research Report, RRR # 47-89, November 1989 (to appear in *ORSA Journal in Computing*).
6. Kolesar, P. (1980), Testing for Vision Loss in Glaucoma Suspects, *Management Science*, **26**, 439–449.
7. Maga, F. and Vizvári, B. (1986), The Relaxation of a Special Polynomial Zero-One Programming Problem to set Covering Problem, *Alkalmazott Matematikai Lapok* **12** 41–49.
8. Nicoloso, S. and Nobili, P. (1990), A Set Covering Formulation of the Matrix Equipartition Problem, Istituto di Analisi dei Sistemi ed Informatica, R.311, November 1990.
9. Nijenhaus, A. and Wilf, H.S. (1975), *Combinatorial Algorithms*, Academic Press, New York.
10. Pardalos, P.M. and Li, Y. (1993), Integer Programming, in *Handbook of Statistics*, Vol. 9. (Editor C.R. Rao), Elsevier, 279–302.
11. Pardalos, P.M., Phillips, A.T. and Rosen, J.B. (1993), *Topics in Parallel Computing in Mathematical Programming*, Science Press.
12. Schoch, M. and Lyska, W. (1978), Kombinatorische Algorithmen zur Lösung spezieller nichtlinearer 0–1 Optimierungsaufgaben, *Mathematische Operationsforschung und Statistik, Ser. Optimization* **9**, 9–20.
13. Tarjan, R.E., Data Structures and Network Flows, CBMS-NSF, Regional conference series in applied mathematics, vol. 44.
14. Vizvári, B. (1975), Enumerative Methods in Polynomial 0–1 Programming, (Hungarian), *Alkalmazott Matematikai Lapok* **1**, 373–384.
15. Wang, X.D. (1988), An Algorithm for Nonlinear 0–1 Programming and Its Application in Structural Optimization, *J. Num. Method & Comp. Appl.* **9**, 22–31.
16. Zak, Y.A. (1978), Algorithms for Nonlinear Pseudo-Boolean Programming, *Engineering Cybernetics* **16**, 29–40.