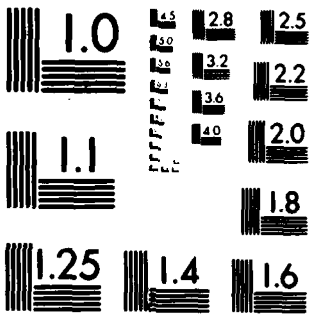

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963 A

AN ACCELERATED COVERING RELAXATION ALGORITHM FOR SOLVING 0-1 POSITIVE POLYNOMIAL PROGRAMS

BY

DANIEL GRANOT, FRIEDA GRANOT
and
WILLEM VAESSEN

TECHNICAL REPORT NO. 93
JUNE 1980

PREPARED UNDER CONTRACT
N00014-76-C-0418 (NR-047-061)
FOR THE OFFICE OF NAVAL RESEARCH

Frederick S. Hillier, Project Director

DEPARTMENT OF OPERATIONS RESEARCH
STANFORD UNIVERSITY
STANFORD, CALIFORNIA

LESS THAN OR =

f sub i

Abstract

The purpose of this paper is to present an accelerated algorithm for solving 0-1 Positive Polynomial (PP) problems of finding a 0-1 vector x that maximizes cx subject to $f(x) \leq b$, where $f(x) = (f_i(x))$ is an m-vector of polynomials with nonnegative coefficients. Like our covering relaxation algorithm (1979) the accelerated algorithm is a cutting-plane method in which each relaxed problem is a set covering problem and the cutting planes are linear covering constraints. However by contrast with other cutting-plane methods in integer programming (including our original method), we do not solve the relaxed problems to optimality after the introduction of the cutting-plane constraints. Rather, we first solve each relaxed set-covering problem heuristically and only if the heuristic solution is feasible for PP do we solve the relaxed problem to optimality. The promise of such an approach stems from the excellent performance of the various heuristic algorithms for solving integer programs. Indeed the extensive computational experimentation we performed reveals that the accelerated approach has reduced significantly both the number of covering problems solved to optimality and the computational time required to solve a PP problem. For example the average computational time required to solve a PP problem with 40 variables and an average of 10.5 terms per constraint and 3 variables per term was reduced using the accelerated method from 84.7 to 25.8 seconds while the average number of covering problems solved to optimality was reduced from 11.5 to 3.8.

1. Introduction

We consider in this paper the 0-1 positive polynomial (PP) problem of finding a 0-1 n vector $x = (x_j)$ that maximizes cx subject to $f(x) \leq b$ where $c = (c_j)$ and $b = (b_i)$ are non-negative, $f(x) = (f_i(x))$ where $f_i(x) = \sum_{k=1}^{p_i} a_{ik} \pi_{j \in N_k} x_j$ with $a_{ik} > 0$ and $N_k \subseteq N = \{1,2,\ldots,n\}$ for each $1 \leq k \leq p_i$ and $i \in M = \{1,\ldots,m\}$.

Polynomial terms in 0-1 variables have been found to be extremely useful in modeling diverse problems in business and engineering. They are used, e.g., when formulating media-selection problems [27], to incorporate risk into capital budgeting [18], for planning of irrigation systems [17], in cluster analysis [20], and in various scheduling problems [13,19]

The two major approaches proposed in the literature for solving PP problems are: the linearization approach and the Boolean-algebra approach.

In the linearization approach [8,9,26] each distinct polynomial term is replaced by a new variable, and additional side constraints are introduced to ensure that the values of the term and the variable replacing it coincide. The main drawback of the linearizaton approach is the radical increase in the size of the linear 0-1 problem obtained. An implicit-enumeration algorithm for solving the transformed linear 0-1 problem in which the additional side constraints introduced by the linearization are considered only implicitly was developed by Taha [22,23]. His computational experiments reveal, though, that such an approach could solve only very modest size problems.

Using Boolean-algebra techniques, F. Granot and Hammer [12] have shown constructively that every PP problem is equivalent to a linear covering problem in the complementing 0-1 variables. They have further suggested an algorithm for solving the PP problem that generates the equivalent linear covering problem and then solves it by available methods. Unfortunately, an attempt to solve

PP problems along the above lines failed, mainly due to the large number of constraints in the equivalent linear covering problem. In [10] Granot et al. have developed a covering relaxation method for solving PP problems. This method produces an optimal solution to PP by solving a nested sequence of linear set-covering problems, each of which is a relaxation of the original PP problem. Computational results reported in [10] indicate that the covering relaxation approach is a viable method for solving modest size sparse PP problems. However, an increase in the density of PP (i.e., in the number of distinct polynomial terms in each constraint) significantly increases the number of covering problems required to be solved, which in turn results in excessively high computational times.

The purpose of this paper is to accelerate the performance of the covering relaxation algorithm by reducing the number of covering problems solved to optimality. Both the covering relaxation and the newly introduced accelerated method are cutting plane algorithms in which relaxation is achieved via the set-covering problem and the cutting planes are linear covering constraints. However, in contrast with other cutting-plane methods in integer programming, we do not solve the augmented problem to optimality after the introduction of the cutting-plane constraints. Rather, we first solve the augmented covering problem heuristically, and only if the heuristic solution is feasible for PP do we solve it to optimality. The promise of such an approach stems from the excellent performance of the various heuristic algorithms for solving integer programming problems, see, e.g., [3,6, 14,15,21,24]. Indeed, the extensive computational experiments we performed, which are reported in section 5, show that the accelerated approach has reduced both the number of covering problems solved to optimality and the computational time required to solve a PP problem by over 70%. This reduction enabled us to solve PP problems with a larger number of polynomial terms in each constraint. For example nine out of ten PP problems with 40 variables, 40 constraints in which the number of terms per constraint is uniformly distributed between 1 and 30 were solved in less than 200 seconds. The average CPU time required to solve those problems was 64.2 seconds while the number of covering problems solved to optimality was on the average 4.8.

2. The Accelerated Covering Relaxation Algorithm

We start by presenting a general framework for our Accelerated Covering Relaxation (ACR) algorithm for solving PP problems. This is followed by a detailed description of the various steps in the algorithm.

The ACR Algorithm

Step 0. Generate an initial linear covering problem that is a relaxation of PP, to be referred to as the covering relaxation (CR) problem.

Step 1. Solve CR heuristically, and let $\bar{x}$ denote the heuristic solution. If $\bar{x}$ is infeasible for PP, go to Step 2; otherwise go to Step 3.

Step 2. Generate covering constraints from violated polynomial constraints at $\bar{x}$. Augment CR with the newly generated constraints to form the new CR problem. Go to Step 1.

Step 3. Solve CR to optimality, and let $\bar{x}$ denote an optimal solution. If $\bar{x}$ is feasible for PP, it is also optimal; otherwise, go to Step 2.

The improved performance of the ACR algorithm, when compared with the covering relaxation method [10], is mainly due to the addition of step 1. In the ACR algorithm we always attempt to generate the cutting planes from the heuristic solution $\bar{x}$ found at step 1. Only if this heuristic solution is feasible for PP, we solve the CR problem to optimality.

Propositions. The ACR algorithm coverges in finitely many iterations to an optimal solution of PP.

Proof. Optimality follows since each CR problem is a relaxation of the original

PP problem. Further, with each application of step 2 of the algorithm we eliminate the current solution $\bar{x}$ to CR, which is infeasible to PP. Since the number of binary vectors is finite, the convergence follows.

The heuristics algorithms that are used in step 1 are presented in the next section. In section 4 we present various strategies for generating cutting planes from violated polynomial constraints, which are employed in step 2 of the ACR algorithm. We further present there a method for generating an initial CR problem.

3. Heuristic Algorithms for Solving Linear Covering Problems

We develop in this section four different heuristic algorithms for solving linear covering problems. In step 1 of the ACR Algorithm we apply the four heuristics to CR, and choose as the heuristic solution $\bar{x}$ the best one produced from these four heuristic algorithms.

Consider again the covering relaxation problem of finding a 0-1 n-vector $\bar{x} = (\bar{x}_i)$ that minimizes $c\bar{x}$ subject to $D_i(\bar{x}) \geq 1$ for all $i \in M$ where $D_i(\bar{x}) \equiv \sum_{j \in N} a_{ij}\bar{x}_j$ and a_{ij} is 0-1 for each $i \in M$ and $j \in N$. We assume, of course, that the CR problem is feasible, i.e., $D_i(1) \geq 1$ for all $i \in M$.

At each stage in the various heuristics there will be a set F of indices j of "fixed" variables $\bar{x}_j$ whose values have been previously determined and a set $M \backslash F$ of indices j of "free" variables whose values may still be revised. To describe the heuristics, we need a bit of notations. To this end, let $M_F = \{i: i \in M$ and $\sum_{j \in F} a_{ij}\bar{x}_j \geq 1\}$ and let J_F be the set of indices j in $N \backslash F$ such that $\bar{x}_j = 1$ and $D_i(\bar{x}) = 1$ for some $i \in M \backslash M_F$ with $a_{ij} = 1$. Denote by $P_k(\bar{x})$ the set of indices j in N for which $\bar{x}_j = k$ and $a_{ij} = 1$ for some i in $M \backslash M_F$ minimizing $D_i(\bar{x})$, for $k = 0, 1$.

Heuristic I. (A primal greedy heuristic that starts with the initial point $\bar{x} = 1$

and $F=J_{\emptyset}$)

Step 1. If $M_F=M$ then $\bar{x}^*$ is the desired heuristic solution where $\bar{x}_j^*=1$ for $j \in F$ and $\bar{x}_j^*=0$ for $j \notin F$. Otherwise

Step 2. Let k be an index j in $P_1(\bar{x})\setminus F$ that maximizes $c_j/\sum_{i \in M\setminus M_F} a_{ij}(D_i(\bar{x})-1)^{-1}$ where ties are broken by choosing j the smallest index that maximizes c_j. Revise $\bar{x}$ by reducing $\bar{x}_k$ to zero. Set $F=F\cup\{k\}\cup J_{F\cup\{k\}}$ and go to step 1.

Clearly, Heuristic I will terminate in at most n iterations.

Heuristic II. (A dual greedy procedure that starts with the initial point $\bar{x}=0$ and $F=\emptyset$).

Step 1. Let k be an index j in $P_0(\bar{x})$ that minimizes $c_j/\sum_{i \in M\setminus M_F}(a_{ij}/\sum_{j \in N} a_{ij})$ where ties are broken by choosing j the smallest index that minimize c_j. Revise $\bar{x}$ by setting $\bar{x}_k=1$, and $F = F\cup\{k\}$. If $M_F \neq M$ go to step 1. Otherwise put $F = \{j: \bar{x}_j=0\}$. If $F\cup J_F = N$, $\bar{x}$ is our desired heuristic solution. Otherwise set $F = F\cup J_F$ and go to step 1 of Heuristic I or III.

Heuristic III. The same as Heuristic I except that the maximization in step 2 is done over all indices j of free variables, and not only those free variables in minimal constraints.

Heuristic IV. The same as Heuristic II except that the minimization at step 1 is done over all indices j for which $\bar{x}_j = 0$.

Remark 1. The reason for returning to Heuristic I or III at the end of Heuristic II is to check whether some of the variables whose values are one can be reduced back to zero.

Remark 2. The four greedy heuristics developed above differ slightly from the ordinary greedy heuristics suggested in the literature for solving integer programs, see, e.g. [15,21,24]. This difference is due to the fact that our criterion for selecting the variable whose value is either decreased (Heuristics I and III) or increased (Heuristics II and IV) depends on the weights $D_i(\bar{x})$ and $\sum_{j\varepsilon N} a_{ij}$, respectively. Dropping these weights, i.e. choosing to increase or decrease the value of a variable in accordance with the ratio $c_j/\sum_i a_{ij}$, results with slightly inferior heuristics, as exhibited in section 5 table 5. Similar conclusions were independently reported in [16].

4. Generating Covering Constraints

In this section we present various strategies for generating covering constraints from violated polynomial constraints. These strategies are employed in step 2 of the ACR algorithm.

Let $\bar{x}^*$ denote either a heuristic or an optimal solution to the current CR problem and let

(1) $\sum_{i=1}^{p'} a_i \pi_{j\varepsilon N_i} x_j \leq b$

be any polynomial inequality derived from a violated constraint of PP at $x^* = 1-\bar{x}^*$ after dropping the vanishing terms at x^*, i.e., $\sum_{i=1}^{p'} a_i \pi_{j\varepsilon N_i} x_j^* = \sum_{i=1}^{p'} a_i > b$. Further, let $S \subseteq N$ be such that $\sum_{N_i \subseteq S} a_i > b$ and its corresponding covering constraint

(2) $\sum_{j\varepsilon S} \bar{x}_j \geq 1$

where $|S|$ is the cardinality of S. Clearly, when (2) is added to the constraint set of CR it eliminates $\bar{x}^*$. In fact (2) eliminates $2^{n-|S|}$ 0-1 vectors from the feasible set of CR (some of which might have been previously eliminated by other covers), and it seems plausible therefore that a best covering constraint (2) that can be produced from (1) is one in which $|S|$ is minimal. Moreover, choosing covers with minimum cardinality will produce covering problems with lower density which, in turn, will decrease the computational time for solving a PP problem. For this reason we have developed an implicit-enumeration algorithm for generating a cover with minimal cardinality. In this algorithm we associate with each term $\pi_{j\epsilon N_i} x_j$ a variable y_i, and search for a set I for which $\sum_{i\epsilon I} a_i > b$ and $|\cup_{i\epsilon I} N_i|$ is minimum.

When the number of terms in a constraint is large, finding a cover with minimum cardinality may require excessive computational time. In this event one can use the following two greedy heuristic methods for generating good covers.

I. The Ordinary Method

Assume that $a_i \geqslant a_{i+1}$, $i=1,\ldots,p'-1$ in (1), and let $S = \cup_{i=1}^{\ell} N_i$ where ℓ is the smallest index for which $\sum_{i=1}^{\ell} a_i > b$. Then, the cover produced by the Ordinary Method is $\sum_{j\epsilon S} \bar{x}_j \geqslant 1$.

II. The Weighted Method

Assume that $a_i/|N_i| \geqslant a_{i+1}/|N_{i+1}|$, $i=1,\ldots,p'-1$, and let $S = \cup_{i=1}^{\ell} N_i$ where ℓ is the smallest index for which $\sum_{i=1}^{\ell} a_i > b$. Then the cover produced by the Weighted Method is $\sum_{j\epsilon S} \bar{x}_j \geqslant 1$.

Remark 3. Let $\sum_{j\epsilon S} \bar{x}_j \geqslant 1$ be the cover produced by the weighted method and let $I = \{i: N_i \subseteq S\}$. Clearly, $\sum_{i\epsilon I} a_i > b$. However, one can easily construct examples for which the set I produced by the weighted method strictly contains a subset I' of I such that $\sum_{i\epsilon I'} a_j > b$. That is, the cover produced by the weighted method can be strengthened.

Remark 4. Let $\sum_{j\epsilon S} \bar{x}_j \geqslant 1$ be a cover produced by the ordinary method. Then one can easily construct examples in which the subset S produced by the ordinary method strictly contains a subset S' of S for which $\sum_{j\epsilon S'} \bar{x}_j \geqslant 1$ is a cover of the polynomial inequality (1).

Example

Consider the polynomial inequality

$8x_1x_2x_3 + 7x_2x_3 + 6x_1x_4 + 5x_1x_5 + 5x_4 + 4x_5 + 2x_3 \leq 15$.

Then, the ordinary method produces the cover $x_1 + x_2 + x_3 + x_4 \leq 3$, the weighted method produces the cover $x_2 + x_3 + x_4 + x_5 \leq 3$, whereas the minimum cardinality covers are $x_1 + x_2 + x_3 \leq 2$ and $x_1 + x_4 + x_5 \leq 2$.

Generating the Initial CR Problem

The initial CR problem used in the Covering Relaxation algorithm [10] is the unconstrained 0-1 problem of finding a 0-1 vector $\bar{x}$ minimizing $c\bar{x}$. The optimal solution is $\bar{x}=0$. Clearly, any subset of the constraints of Granot and Hammer's equivalent covering problem [12] can be used as the constraints of the initial CR problem. In step 0 of the ACR algorithm the initial CR problem is produced by employing F. Granot's [11] efficient greedy heuristics for solving PP problems. This choice of an initial CR problem was found to reduce the number of iterations as well as total computational time required to solve a PP problem. Explicitly let x designate a heuristic solution to PP obtained by using any of the methods in [11] and let $T(x) = \{j: x_j = 0\}$. For each $j \epsilon T(x)$ denote by PP^j the polynomial problem obtained by substituting $x_k=0$ in PP for all $k \epsilon T(x) \setminus \{j\}$. Further, let F^j be the collection of covering constraints derived by generating one covering constraint from each violated polynomial constraint in PP^j using the ordinatry method. The initial CR consists of all covering constraints in $\cup_{j \epsilon T(x)} F^j$.

One can advance the following intuitive argument for explaining the effectiveness of using this initial CR problem in step 0. Any heuristic solution $\bar{x}$ produced by the greedy methods in [11] is a near-optimal solution for PP, and is in fact quite frequently optimal. Therefore, the set of covering constraints in the initial CR problem and hence in every subsequent CR problem tends to form a fairly tight relaxation of the feasible set of PP in the neighbourhood of its optimal solution.

5. Computational Results

Our algorithm was coded in Fortran IV and tested on an AMDAHL 470 V-6 computer model II using the IBM optimizing compiler. It was tested on randomly-generated problems that were generated as follows.

The cost vector c was determined by setting $c_0 = 1$ and $c_{j+1} = c_j + \bar{c}$ where $\bar{c}$ is uniform on [0,10]. Every constraint is of the form

$$\textstyle\sum_{j=1}^{p} a_{ij} y_j \leq b_i \text{ with } y_j = \pi_{i=1}^{e} x_{R_i}$$

where the number of terms p is uniform on [1,P]; the number of variables e in each term is uniform on [1,E]; a_{ij} is uniform on [1,20]; R_i is uniformly chosen between 1 and n; α is a constant between 0 and 1 that measures the tightness of the constraints; the right-hand side b_i is chosen to equal $\alpha \sum_{j=1}^{p} a_{ij}$ (we have $\alpha = 0.5$ in all problems reported since for that choice of α the PP problems are the most difficult to solve [10]); m is the number of constraints and n is the number of variables. Some specific test problems can be found in [25].

The various modifications that were introduced in sections 2, 3 and 4 to accelerate the performance of the covering relaxation algorithm [10] are primarily designed to reduce the number of covering problems that have to be solved to optimality. In order to evaluate their effectiveness, therefore, we should test the performance of the ACR algorithm on PP problems whose optimal solution was produced by the covering relaxation algorithm at the cost of solving relatively many covering problems to optimality.

The extensive computational experiments performed in [10] have revealed that the dominant factor affecting the number of iterations (i.e., number of covering problems solved to optimality) in the covering relaxation algorithm is the maximum number of polynomial terms P in each polynomial constraint. An increase in P increases very significantly the number of iterations (and, of course, therefore the computational time). On the other hand, an increase in the number of variables n has no significant effect on the number of iterations. Clearly, however, an increase in n significantly increases the computational time, but this is mainly due to the increase in the number of variables n in the covering problems which become therefore much more difficult to solve.

For these reasons we tested the ACR algorithm on PP problems for which P is relatively large. We did not attempt to solve PP problems for which n is larger than 40 since (i) (as explained above) the number of variables n has no significant effect on the number of iterations in the covering relaxation algorithm and (ii) our algorithm for solving the covering problems to optimality is a very rudimentary implicit enumeration algorithm which requires excessive computational time to solve covering problems with more than 40 variables.

We use at step 3 of the ACR algorithm Balas' implicit enumeration algorithm to solve the linear covering problem to optimality. No surrogate constraints or subgradient optimization techniques are employed. However, various simplification and elimination rules are applied to the constraint matrix of the covering problem before the problem is solved. Clearly, the sequence of CR problems solved, either heuristically or to optimality, by the ACR algorithm is nested. Therefore, we always start the search for an optimal solution of a CR problem from that node in the solution tree corresponding to the first feasible solution of the previous CR problem which was solved to optimality (if any). This approach certainly reduces the computational time for solving a PP problem. However, still, most of the computational time for solving a PP problem is spent on solving CR problems to optimality. This is evident from Figure 1 below, which presents the percentage of total CPU time required by the ACR algorithm (Method VA) to solve covering problems to optimality as a function of CPU time per iteration. We may further conclude from Figure 1 that the overall computational time for solving a PP problem can be significantly reduced by employing a more efficient algorithm, e.g., [2,4,5], for solving the covering problems to optimality.

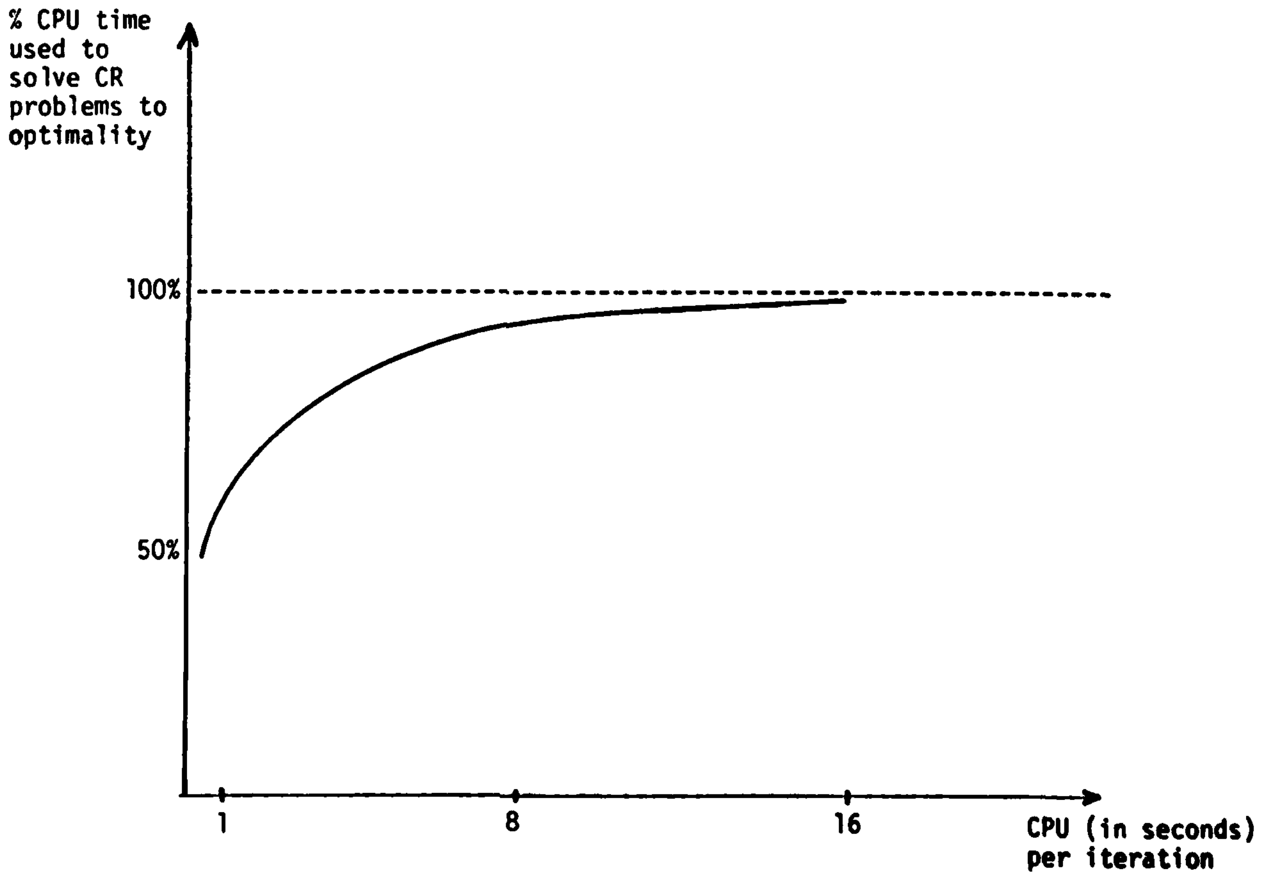

Figure 1

We have investigated the computational efficiency of the following methods for solving PP problems.

Method I - This is the covering relaxation method studied in [10], i.e., the ACR algorithm with step 1 omitted. The initial CR problem is the unconstrained problem of choosing a 0-1 vector $\bar{x}$ minimizing $c\bar{x}$. At each iteration: (i) we generate one covering constraint from each violated polynomial constraint using the ordinary method, and (ii) we solve the covering problem to optimality.

Method II - Same as Method I except that step 1 of the ACR algorithm is carried out.

Method III - Same as Method II except that the initial CR in step 0 of the ACR algorithm is generated from a heuristic solution to the PP problem as explained in section 4.

Method IV - Same as Method III except for the way a cover is generated from a violated polynomial constraint. In this method we first arrange the terms of the polynomial inequality in decreasing order of $a_i/|N_i|$, and then the implicit enumeration method is used to generate five covers. These five covers are the first five encountered in the depth-first search of the solution tree. In fact, the first of these five covers is the one produced by the weighted method. The covering constraint chosen is the first minimum-cardinality cover generated.

Method IVA - Same as Method III except that the cover generated from each violated polynomial inequality has minimum cardinality.

Method V - The difference between Method V and Methods III, IV and IVA is in the way we generate the covering constraints from the violated polynomial inequalities. The number k of covers generated at each iteration is equal to the number of violated polynomial inequalities. However, those k covers generated are the ones with minimum cardinality i.e., one cannot generate another covering constraint at that iteration whose cardinality is strictly smaller than the cardinality of any of these k covers.

Method VA - Same as Method V except that the number of covers generated at each iteration depends on the number k of violated polynomial inequalities. Explicitly, the number of covering constraints generated equals Max{k,[.2m]}.

Method VAA - Same as Method VA except that the weights $D_i(\bar{x})$ or $\sum_{j\epsilon N} a_{ij}$ a~e not used in Heuristics I, II, III, IV for solving the CR problem (see remark 2).

In tables 1-6 below we use the following notation:

CPU	- The true execution time in seconds required to solve a PP problem excluding the time required to generate the problem.
OPT	- The number of covering problems solved to optimality.
HEUR	- The number of covering problems solved heuristically.
COVERS	- The number of constraints in the last covering problem solved to optimality.
DENSITY	- The average density of those covering problems solved to optimality.
EFFECTIVENESS	- The average percentage of optimal value to best approximate value for all CR problems that were solved to optimality.

Table 1: α = .5, E = 5.

mxn	p	METHOD I				METHOD II				METHOD III			
		CPU	OPT.	HEUR.	COVERS	CPU	OPT.	HEUR.	COVERS	CPU	OPT.	HEUR.	COVERS
30x30	2	0.1	1.0	-	11.6	0.1	1.0	1.0	11.6	0.1	1.0	1.0	10.4
	5	0.3	2.3	-	18.7	0.3	1.1	2.6	19.1	0.3	1.1	2.0	17.7
	8	1.1	4.5	-	27.9	0.9	1.6	5.1	28.4	0.7	1.3	3.9	25.0
	11	2.0	5.7	-	37.8	1.2	1.5	6.0	38.4	1.0	1.1	4.7	36.4
	14	2.6	6.4	-	50.2	1.8	1.6	7.0	52.0	1.6	1.7	4.8	43.4
	17	8.1	10.5	-	78.9	5.5	3.1	12.3	78.6	5.3	2.8	9.1	67.9
	20	10.0	10.9	-	76.7	6.2	3.2	12.0	78.7	5.4	2.9	9.5	70.9
40x40	2	0.2	1.0	-	16.8	0.3	1.0	1.0	16.8	0.3	1.1	1.2	14.3
	5	3.8	3.2	-	25.4	3.2	1.1	3.5	25.8	2.0	1.0	2.2	23.1
	8	11.8	5.0	-	40.2	6.8	1.6	6.0	41.9	4.4	1.3	3.8	36.1
	11	21.3	5.9	-	50.4	9.7	1.6	7.4	51.8	6.6	1.4	5.0	46.5
	14	44.2	8.5	-	70.7	27.0	3.2	9.6	76.8	18.7	2.5	8.0	67.6
	17	71.6	10.3	-	100.6	38.1	3.5	13.2	109.9	38.0	3.5	11.0	99.8
	20	84.7	11.5	-	99.1	45.3	4.0	14.6	106.1	41.5	3.8	11.3	93.9

Table 2: α = .5, E = 5.

mxn	p	METHOD III					METHOD IV					METHOD IV A					METHOD V				
		CPU	OPT.	HEUR.	COVERS	DENSITY	CPU	OPT.	HEUR.	COVERS	DENSITY	CPU	OPT.	HEUR.	COVERS	DENSITY	CPU	OPT.	HEUR.	COVERS	DENSITY
30x30	2	0.1	1.0	1.0	10.4	0.09	0.1	1.0	1.0	10.4	0.09	0.1	1.0	1.0	10.4	0.09	0.1	1.0	1.0	10.3	0.09
	5	0.3	1.1	2.0	17.7	0.13	0.3	1.1	1.7	17.9	0.13	0.3	1.1	1.7	17.9	0.13	0.3	1.1	1.8	18.0	0.13
	8	0.7	1.3	3.9	25.0	0.17	0.7	1.3	3.6	25.4	0.16	0.7	1.3	3.6	25.4	0.16	0.7	1.3	3.6	25.2	0.16
	11	1.0	1.1	4.7	36.4	0.21	0.8	1.1	4.3	36.7	0.21	0.8	1.1	4.3	36.7	0.21	0.8	1.1	4.6	36.9	0.20
	14	1.6	1.7	4.8	43.4	0.23	1.1	1.4	4.1	45.6	0.23	1.1	1.4	4.1	45.6	0.23	1.1	1.5	4.1	42.5	0.22
	17	5.3	2.8	9.1	67.9	0.29	4.2	2.3	7.2	69.1	0.28	4.2	2.3	7.2	69.1	0.28	3.6	2.6	7.9	62.0	0.26
	20	5.4	2.9	9.5	70.9	0.31	4.4	2.9	6.9	66.9	0.30	4.2	2.7	6.9	66.7	0.30	3.8	2.8	7.2	63.9	0.28
40x40	2	0.3	1.1	1.2	14.3	0.07	0.3	1.1	1.2	14.3	0.07	0.3	1.1	1.2	14.3	0.07	0.3	1.1	1.2	14.3	0.07
	5	2.0	1.0	2.2	23.1	0.10	2.0	1.1	2.1	23.3	0.10	2.0	1.1	2.1	23.3	0.10	1.9	1.1	2.1	23.3	0.10
	8	4.4	1.3	3.8	36.1	0.14	4.6	1.3	3.9	37.8	0.13	4.5	1.3	3.9	37.8	0.13	4.5	1.3	4.0	37.9	0.13
	11	6.6	1.4	5.0	46.5	0.16	6.5	1.4	5.1	45.9	0.16	6.5	1.4	5.1	45.9	0.16	5.8	1.3	4.5	44.4	0.16
	14	18.7	2.5	8.0	67.6	0.19	19.0	2.9	7.7	69.1	0.19	19.0	2.9	7.7	69.1	0.19	11.6	2.8	6.2	61.0	0.18
	17	38.0	3.5	11.0	99.8	0.24	28.1	2.7	8.3	95.0	0.23	28.2	2.7	8.3	95.0	0.23	30.2	3.6	8.7	85.1	0.21
	20	41.5	3.8	11.3	93.9	0.25	34.3	3.6	9.3	93.5	0.24	34.8	3.6	9.4	94.9	0.24	32.2	3.8	9.4	86.8	0.22

Table 3: $\alpha = .5$, $E = 5$

mxn	P	METHOD V					METHOD V A				
		CPU	OPT.	HEUR.	COVERS	DENSITY	CPU	OPT.	HEUR.	COVERS	DENSITY
30x30	2	0.1	1.0	1.0	10.4	0.09	0.1	1.0	1.0	10.4	0.09
	5	0.3	1.1	1.8	18.0	0.13	0.3	1.1	1.7	17.7	0.13
	8	0.7	1.3	3.6	25.2	0.16	0.6	1.1	3.1	26.2	0.16
	11	0.8	1.1	4.6	36.9	0.20	0.8	1.1	3.6	37.6	0.21
	14	1.1	1.5	4.1	42.5	0.22	1.1	1.3	3.6	44.6	0.22
	17	3.6	2.6	7.9	62.0	0.26	2.1	2.4	7.1	66.7	0.27
	20	3.8	2.8	7.2	63.9	0.28	2.6	2.2	6.7	67.5	0.28
40x40	2	0.3	1.1	1.2	14.3	0.07	0.3	1.1	1.2	14.3	0.07
	5	1.9	1.1	2.1	23.3	0.10	2.0	1.1	2.1	23.4	0.10
	8	4.5	1.3	4.0	37.9	0.13	4.8	1.3	3.6	39.4	0.14
	11	5.8	1.3	4.5	44.4	0.16	7.2	1.5	5.1	49.8	0.16
	14	11.6	2.8	6.2	61.0	0.18	13.6	2.4	6.0	70.4	0.18
	17	30.2	3.6	8.7	85.1	0.21	24.0	2.9	8.5	98.1	0.22
	20	37.2	3.8	9.4	86.8	0.22	25.8	3.0	8.9	100.9	0.23

Table 4: $\alpha = .5$

mxn	P	METHOD VA E = 1					METHOD VA E = 5					METHOD VA E = 9				
		CPU	OPT.	HEUR.	COVERS	DENSITY	CPU	OPT.	HEUR.	COVERS	DENSITY	CPU	OPT.	HEUR.	COVERS	DENSITY
30x30	5	0.2	1.1	1.6	11.9	0.07	0.3	1.1	1.7	17.7	0.13	0.2	1.2	1.8	16.8	0.17
	8	0.4	1.2	2.5	24.9	0.09	0.6	1.1	3.1	26.2	0.16	0.3	1.0	2.0	19.6	0.22
	11	0.9	1.3	4.0	48.2	0.11	0.8	1.1	3.6	37.6	0.21	0.4	1.2	3.0	26.3	0.26
	14	5.6	2.9	9.4	88.8	0.13	1.1	1.3	3.6	44.6	0.22	0.5	1.3	3.3	28.6	0.30
	17	8.7	3.6	12.5	111.8	0.14	2.1	2.4	7.1	66.7	0.27	0.7	1.4	3.3	31.2	0.31
	20	22.9	3.8	19.5	174.2	0.16	2.6	2.2	6.7	67.5	0.28	0.8	1.4	4.0	40.8	0.37
40x40	5	0.3	1.0	1.4	17.3	0.05	2.0	1.1	2.1	23.4	0.10	0.5	1.1	1.8	19.8	0.14
	8	1.4	1.2	2.7	36.6	0.06	4.8	1.3	3.6	39.4	0.14	1.1	1.3	2.5	26.7	0.17
	11	6.4	1.3	4.4	63.7	0.08	7.2	1.5	5.1	49.8	0.16	2.3	1.4	3.5	35.6	0.21
	14	24.1	2.0	8.5	119.6	0.10	13.6	2.4	6.0	70.4	0.18	1.7	1.3	4.0	39.1	0.25
	17	52.7*	4.3*	12.8*	164.3*	0.12*	24.0	2.9	8.5	98.1	0.22	3.2	2.3	4.4	46.5	0.27
	20	+	+	+	+	+	25.8	3.0	8.9	100.9	0.23	2.7	2.1	5.5	60.1	0.32

* In two of the ten runs the optimal solution was not obtained after 150 seconds of CPU time. The average is of the other eight runs.

+ No attempt was made to solve these problems.

Table 5: $\alpha = .5$, $E = 5$

mxn	P	Method VA					Method VAA				
		CPU	OPT.	HEUR.	COVERS	EFFECTIVENESS	CPU	OPT.	HEUR.	COVERS	EFFECTIVENESS
30x30	2	0.1	1.0	1.0	10.4	99.9	0.1	1.0	1.0	10.4	99.6
	5	0.3	1.1	1.7	17.7	99.5	0.3	1.1	1.9	18.0	99.2
	8	0.6	1.1	3.1	26.2	99.5	0.7	1.4	3.4	26.8	99.3
	11	0.8	1.1	3.6	37.6	98.5	0.8	1.3	3.7	36.5	98.4
	14	1.1	1.3	3.6	44.6	98.7	1.1	1.6	3.8	46.9	98.3
	17	2.1	2.4	7.1	66.7	98.7	2.2	2.5	7.6	69.0	98.1
	20	2.6	2.2	6.7	67.5	98.5	2.6	2.5	7.2	68.6	97.6
40x40	2	0.3	1.1	1.2	14.3	99.8	0.3	1.1	1.2	14.3	99.6
	5	2.0	1.1	2.1	23.4	99.7	2.1	1.1	2.4	23.8	99.3
	8	4.8	1.3	3.6	39.4	98.7	4.8	1.4	4.1	40.8	98.5
	11	7.2	1.5	5.1	49.8	98.9	9.0	1.7	4.5	48.2	98.4
	14	13.6	2.4	6.0	70.4	98.3	20.6	3.4	7.3	72.9	98.1
	17	24.0	2.9	8.5	98.1	98.7	25.9	3.0	9.3	98.1	98.0
	20	25.8	3.0	8.9	100.9	98.8	33.9	3.4	10.6	110.0	98.4

Table 6: $\alpha = .5, \varepsilon = 5.$

mxn	P	METHOD V A				
		CPU	OPT.	HEUR.	COVERS	DENSITY
30x30	2	0.1	1.0	1.0	10.4	0.09
	5	0.3	1.1	1.7	17.7	0.13
	8	0.6	1.1	3.1	26.2	0.16
	11	0.8	1.1	3.6	37.6	0.21
	14	1.1	1.3	3.6	44.6	0.22
	17	2.1	2.4	7.1	66.7	0.27
	20	2.6	2.2	6.7	67.5	0.28
	23	3.1	3.1	8.5	70.7	0.29
	26	7.7	4.6	11.2	95.5	0.32
	30	14.3	4.4	13.1	103.0	0.36
40x40	2	0.3	1.1	1.2	14.3	0.07
	5	2.0	1.1	2.1	23.4	0.10
	8	4.8	1.3	3.6	39.4	0.14
	11	7.2	1.5	5.1	49.8	0.16
	14	13.6	2.4	6.0	70.4	0.18
	17	24.0	2.9	8.5	98.1	0.22
	20	25.8	3.0	8.9	100.9	0.23
	23	31.7	3.7	10.3	108.0	0.25
	26	63.8	5.2	13.2	148.6	0.27 +
	30	64.2	4.8	14.7	142.6	0.31 +

+ In one of the ten runs the optimal solution was not obtained after 200 seconds of CPU time. The average is of the other nine runs and so is underestimated.

The following conclusions can be drawn from the above tables.

a) From Table 1 it is apparent that a substantial improvement in the performance of the covering relaxation method [10] (i.e. Method I) is realized with the introduction of step 1 of the ACR algorithm (i.e. Method II). Attempting to generate cutting planes from a heuristic solution to a CR problem results in a very significant decrease in both the number of covering problems solved to optimality and the computational time. Observe, though, that the introduction of step 1 has increased both the number of iterations (as measured by the number of covering problems solved heuristically in Method II and to optimality in Method I) and the average number of constraints in the last covering problem.

b) When the initial CR problem in Method II is generated from a heuristic solution to the PP problem (i.e. Method III) additional improvements are realized. Explicitly, there is a further significant reduction in both the number of CR problems solved heuristically and the size of the last CR problem, particularly for large P.

c) Table 2 measures the effectiveness of employing various strategies for generating cutting planes on computational performance. For large values of P(P=17,20), there is a significant decrease in the CPU time and the number of CR problems solved heuristically when we attempt to generate covering constraints with low cardinality. Observe that there is hardly any difference between Method IV, in which we choose the best of the first five covers generated and Method IVA, in which we generate a cover with minimum cardinality from each violated polynomial inequality. Of course, this might change when P is much larger. The density of the CR problems, as expected, is reduced when we generate minimal cardinality covers (i.e. Method V).

d) From Table 3 we conclude that by generating at each iteration at least .2m covering constraints with minimum cardinality (i.e. Method VA), both the number of CR problems solved to optimality and the CPU time are significantly decreased.

e) From Table 4 it is evident that PP problems with more variables in a term are easier to solve.

f) From Table 5 we see that the effectiveness of our greedy heuristics is uniformly superior to that of the ordinary greedy heuristics, in which the weights are not incorporated in the selection criterion of variables whose values are either decreased (Heuristics I, III) or increased (Heuristics II, IV), as explained in Remark 2. This superiority results in a better computational performance of our ACR algorithm.

g) From Table 6 we conclude that increasing P - the maximum number of terms in a polynomial constraint - beyond 20 increases significantly the CPU time required to solve a PP problem. Indeed the CPU time appears to grow exponentially in P. Observe that the increase in the number of covering problems solved to optimality is more moderate.

We used the data in Tables 1 and 6 to estimate the regression equation between OPT - the number of set covering problems solved to optimality and P. For method I the regression equations were found to be

a) $n=40$: $OPT = .27 + .57P$, $s_b=.03$

b) $n=30$: $OPT = -.66 + .59P$, $s_b=.07$

For method VA the regression equations were found to be

a) $n = 40$: $OPT = -.03 + .17P$, $s_b=.02$

b) $n = 30$: $OPT = -.37 + .16P$, $s_b=.02$

where s_b is an unbiased estimate of the standard deviation of the coefficient of P in the various regression equations. All four regressions were obtained after eliminating the runs for P=2.

By comparing the appropriate regression equations we realize that, on the average, method VA solves 70% less covering problems to optimality than method I. Further observe that in method VA, an increase in the number of variables from 30 to 40 has increased, on the average, the number of covering problems that need to be solved to optimality only by one half of a problem. This confirms our conclusion based on the computational results in [10] that the number of variables n has no significant effect on the number of covering problems that need to be solved to optimality by the covering relaxation methods.

Acknowledgement

The authors gratefully acknowledge some very constructive remarks made by Professor Arthur F. Veinott, Jr. on a previous version of this paper.

REFERENCES

[1] BALAS, E., "An Additive Algorithm for Solving Linear Programs with Zero-One Variables", Operations Research, Vol. 13 (1965), pp. 517-546.

[2] BALAS, E. and Andrew HO, "Set Covering Algorithms Using Cutting Planes, Heuristics, and Subgradient Optimization: A Computational Study", WP No. 6-79-80, GSIA (July 1979).

[3] BALAS,E. and C. H. MARTIN, "Pivot and Complement - A Heuristic for 0-1 Programming", MSR Report No. 414, Carnegie Mellon University, February 1978.

[4] CHRISTOFIDES, N. and S. Korman, "A Computational Survey of Methods for the Set Covering Problem", Management Science, Vol. 21 (1975), pp. 591-599.

[5] ETCHEBERRY, J., "The Set Covering Problem: A New Implicit Enumeration Algorithm", Operations Research, Vol. 25 (1977), pp. 760-772.

[6] FAALAND, B. H. and F. S. HILLIER, "Interior Path Methods for Heuristic Integer Programming Procedures", Operations Research, Vol. 27, (1979), pp. 1069-1087.

[7] GARFINKEL, R. and G. L. NEMHAUSER, Integer Programming, John Wiley & Sons, 1972.

[8] GLOVER, F. and E. WOOLSEY, "Further Reduction of Zero-One Polynomial Programming Problems to Zero-One Linear Programming Problems", Operations Research, Vol. 21, No. 1 (1973), pp. 141-161.

[9] GLOVER, F. and E. WOOLSEY, "Converting the 0-1 Polynomial Programming Problem to a 0-1 Linear Program", Operations Research, Vol. 22, No. 1 (1974), pp. 180-182.

[10] GRANOT, D., F. GRANOT and J. KALLBERG, "Covering Relaxation for Positive 0-1 Polynomial Programs", Management Science, Volume 25, No. 3 (1979), pp. 264-273.

[11] GRANOT, F., "Efficient Heuristic Algorithms for Positive 0-1 Polynomial Programming Problems", Working Paper 595, Faculty of Commerce and Business Administration, U.B.C., Vancouver, B.C., (August 1978).

[12] GRANOT, F. and P. L. HAMMER, "On the Use of Boolean Functions in 0-1 Programming", Methods for Operations Research, XII, (1971), pp. 154-184.

[13] HAMMER, P.L. and S. RUDEANU, Boolean Methods in Operations Research, Springer-Verlag (1968).

[14] HILLIER, F.S., "Efficient Heuristic Procedures for Integer Linear Programming with an Interior", Operations Research, 17 (1969), pp. 600-637.

[15] KOCKENBERG, G.A., B.A. MCCOIL and F.P. WYMAN, "A Heursitic for General Integer Programming", Decision Sciences, Vol. 5 (1974), pp. 36-44.

[16] LOULOU, R. and E. MICHAELIDES, "New Greedy-like Heuristics for the Multi-dimensional 0-1 Knapsack Problem", Operations Research, Vol. 27 (1979), pp. 1101-1114.

[17] ORON, G., "Optimizing Irrigation System Design", Ph.D. Dissertation, Ag. Eng. Dept., Technion, Haifa, Israel (1975).

[18] PETERSON, D.E. and D. LAUGHHUNN, "Capital Expenditure Programming and Some Alternative Approaches to Risk", Management Science, Vol. 17 (1971), pp. 320-336.

[19] PRITSKER, A., L. J. WATTERS and F. WOLFE, "Multiproject Scheduling With Limited Resources: A Zero-one Programming Approach", Management Science, Vol. 16 (1969), pp. 93-108.

[20] RAO, M.R., "Cluster Analysis and Mathematical Programming", Journal of the American Statistical Association, Vol. 66 (1971), pp. 622-626.

[21] SENJIU, S. and Y. TOYODA, "An Approach to Linear Programming with 0-1 Variables", Management Science, 15 (1968), pp. B196-207.

[22] TAHA, H.A., "A Balasian-Based Algorithm for Zero-One Polynomial Programming," Management Science, Vol. 18(1972), pp. B328-B343.

[23] TAHA, H.A., "Further Improvements in the Polynomial Zero-One Algorithm," Management Science, Vol. 19(1972), pp. B226-B227.

[24] TOYODA, Y., "A Simplified Algorithm for Obtaining Approximate Solutions to 0-1 Programming Problems", Management Science, 21 (1975), pp. 1417-1427.

[25] VAESSEN, W., "Covering Relaxation Methods for Solving the Zero-One Positive Polynomial Programming Problems", M.Sc. Thesis, Computer Science Department., IN 80-4, U.B.C. (May 1980).

[26] WATTERS, L.J., "Reduction of Integer Polynomial Programming Problems to Zero-One Linear Programming Problems", Operations Research, Vol. 15 (1967), pp. 1171-1174.

[27] ZANGWILL, W., "Media Selection by Decision Programming", Journal of Advertising Research, Vol. 5, (1965), pp. 30-36.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 93	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) AN ACCELERATED COVERING RELAXATION ALGORITHM FOR SOLVING 0-1 POSITIVE POLYNOMIAL PROGRAMS		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Daniel Granot, Frieda Granot and Willem Vaessen		8. CONTRACT OR GRANT NUMBER(s) N00014-76-C-0418
9. PERFORMING ORGANIZATION NAME AND ADDRESS Operations Research Program - ONR Department of Operations Research Stanford University, Stanford, Calif.		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR-047-061
11. CONTROLLING OFFICE NAME AND ADDRESS OFFICE OF NAVAL RESEARCH OPERATIONS RESEARCH PROGRAM CODE 434 ARLINGTON, VA. 22217		12. REPORT DATE JUNE 1980
		13. NUMBER OF PAGES 22
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Also issued as Technical Report No. 80-10, Dept. of Operations Research Stanford University, under National Science Foundation Grant MCS76-81259.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

INTEGER PROGRAMMING	POLYNOMIAL PROGRAMMING
INTEGER POLYNOMIAL PROGRAMMING	ALGORITHMS

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

See other side

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Technical Report No. 93

The purpose of this paper is to present an accelerated algorithm for solving 0-1 Positive Polynomial (PP) problems of finding a 0-1 vector x that maximizes cx subject to $f(x) \leq b$, where $f(x) = (f_i(x))$ is an m-vector of polynomials with nonnegative coefficients. Like our covering relaxation algorithm (1979) the accelerated algorithm is a cutting-plane method in which each relaxed problem is a set covering problem and the cutting planes are linear covering constraints. However by contrast with other cutting-plane methods in integer programming (including our original method), we do not solve the relaxed problems to optimality after the introduction of the cutting-plane constraints. Rather, we first solve each relaxed set-covering problem heuristically and only if the heuristic solution is feasible for PP do we solve the relaxed problem to optimality. The promise of such an approach stems from the excellent performance of the various heuristic algorithms for solving integer programs. Indeed the extensive computational experimentation we performed reveals that the accelerated approach has reduced significantly both the number of covering problems solved to optimality and the computational time required to solve a PP problem. For example the average computational time required to solve a PP problem with 40 variables and an average of 10.5 terms per constraint and 3 variables per term was reduced using the accelerated method from 84.7 to 25.8 seconds while the average number of covering problems solved to optimality was reduced from 11.5 to 3.8