Discrete Applied Mathematics 29 (1990) 171-185
North-Holland

THE BASIC ALGORITHM FOR PSEUDO-BOOLEAN PROGRAMMING REVISITED

Yves CRAMA
Department of Quantitative Economics, University of Limburg, Postbus 616, 6200 MD Maastricht, The Netherlands

Pierre HANSEN
RUTCOR, Rutgers University, P.O. Box 5062, New Brunswick, NJ 08903, USA

Brigitte JAUMARD
GERAD and Department of Applied Mathematics, École Polytechnique de Montréal, Campus de l'Université de Montréal, Case Postale 6079, Succursale "A" Montréal, Québec, Canada H3C 3A7

Received 10 November 1988
Revised 10 April 1989

The basic algorithm of pseudo-Boolean programming due to Hammer and Rudeanu allows to minimize nonlinear 0-1 functions by recursively eliminating one variable at each iteration. We show it has linear-time complexity when applied to functions associated in a natural way with graphs of bounded tree-width. We also propose a new approach to the elimination of a variable based on a branch-and-bound scheme, which allows shortcuts in Boolean manipulations. Computational results are reported on.

1. Introduction

The *basic algorithm* for pseudo-Boolean programming was originally proposed by Hammer, Rosenberg and Rudeanu 25 years ago [10, 11]. A streamlined form due to Hammer and Rudeanu is presented in their book "Boolean Methods in Operations Research and Related Areas" [12]. This algorithm determines the maximum of a nonlinear function in 0-1 variables, or pseudo-Boolean function, by recursively eliminating variables. Following the dynamic programming principle, local optimality conditions are exploited so as to produce at each iteration a function depending on one less variable, but having the same optimal value as the previous one.

No systematic effort for programming this method appears to have been made. Possible reasons for this are the relative sophistication required to implement Boolean manipulations of formulas, and the large memory space necessary for the storage of intermediary results. Hence, with the advent of new methods, the basic algorithm has progressively fallen into oblivion during the last decade.

For several reasons, this seems unwarranted. First, while the basic algorithm may not be the most efficient for maximizing arbitrary pseudo-Boolean functions, it might be the case that it is very well adapted to the maximization of functions belonging to particular classes. For instance, several polynomial algorithms (see [6, 8, 15]), which are in fact subsumed by the basic algorithm, have been recently proposed for the maximization of quadratic pseudo-Boolean functions associated with trees or series-parallel graphs. Second, some steps of the basic algorithm might be simplified. Third, the explosive development of algorithm design and data structuring renders the programming of complex combinatorial methods much easier to do efficiently than even a decade ago. Fourth, the availability of powerful computers with large internal memory substantially increases the size of potentially solvable problems.

In this paper, we reconsider the basic algorithm both from the theoretical and from the computational point of view. We recall the details of the basic algorithm in Section 2, and we present the variant of it we will consider in the sequel of the paper. In Section 3, we establish that the basic algorithm has linear-time complexity when applied to functions associated in a natural way with graphs of bounded tree-width.

In Section 4, we propose a new approach to the elimination of a variable, based on a branch-and-bound scheme. This subroutine is directly applied to the subfunction defined by the terms containing the variable to be eliminated, and several steps of the basic algorithm, using Boolean manipulations, are thus cut short. We study in Section 5 the data structures required for the representation of pseudo-Boolean functions, and for an efficient implementation of the algorithm. We also report on computational experiments.

2. The basic algorithm

A *pseudo-Boolean function* is a real-valued function of 0–1 variables. Any such function can be (nonuniquely) expressed as a polynomial in its variables $x_1, x_2, \ldots, x_n$ and their complements $\bar{x}_1 = 1 - x_1, \bar{x}_2 = 1 - x_2, \ldots, \bar{x}_n = 1 - x_n$, of the general form:

$$f(x_1, x_2, \ldots, x_n) = \sum_{t=1}^{m} c_t \prod_{j \in T(t)} x_j^{\alpha_{jt}},$$

where m is the number of *terms* of the polynomial, $T(t)$ is a subset of $\{1, 2, \ldots, n\}$ $(t = 1, 2, \ldots, m)$, $\alpha_{jt} \in \{0, 1\}$ $(t = 1, 2, \ldots, m;\ j \in T(t))$, and $x^1 = x$, $x^0 = \bar{x}$.

We present now the scheme of the *basic algorithm* of Hammer, Rosenberg and Rudeanu, for maximizing over $\{0, 1\}^n$ a pseudo-Boolean function in polynomial form.

Let f_1 be the function to be maximized, and write:

$$f_1(x_1, x_2, \ldots, x_n) = x_1 g_1(x_2, x_3, \ldots, x_n) + h_1(x_2, x_3, \ldots, x_n),$$

where the functions g_1 and h_1 do not depend on x_1. Clearly, there exists a maximizing point of f_1, say $(x_1^*, x_2^*, \ldots, x_n^*)$, such that $x_1^* = 1$ if $g_1(x_2^*, x_3^*, \ldots, x_n^*) > 0$, and such that $x_1^* = 0$ if $g_1(x_2^*, x_3^*, \ldots, x_n^*) \leq 0$. This leads us to define a function $\psi_1(x_2, x_3, \ldots, x_n)$, such that $\psi_1(x_2, x_3, \ldots, x_n) = g_1(x_2, x_3, \ldots, x_n)$ if $g_1(x_2, x_3, \ldots, x_n) > 0$, and $\psi_1(x_2, x_3, \ldots, x_n) = 0$ otherwise. Assume that a polynomial expression of ψ_1 has been obtained. Letting $f_2 = \psi_1 + h_1$, we have thus reduced the maximization of the original function in n variables to the maximization of f_2, which only depends on the $n-1$ variables $x_2, x_3, \ldots, x_n$.

Continuing this elimination process for $x_2, x_3, \ldots, x_{n-1}$, successively, we produce two sequences of functions $f_1, f_2, \ldots, f_n$ and $\psi_1, \psi_2, \ldots, \psi_{n-1}$, where f_i depends on $n-i+1$ variables. A maximizing point $(x_1^*, x_2^*, \ldots, x_n^*)$ of f_1 can then easily be tracked back from any maximizer x_n^* of f_n, using the recursion:

$$x_i^* = 1 \quad \text{if and only if} \quad \psi_i(x_{i+1}^*, x_{i+2}^*, \ldots, x_n^*) > 0 \quad (i = 1, 2, \ldots, n-1).$$

Obviously, the efficiency of this algorithm hinges critically on how difficult the polynomial expressions of $\psi_1, \psi_2, \ldots, \psi_{n-1}$ are to obtain. And, as already observed in [12], this can in turn be influenced by the elimination order chosen for the variables. We return to this issue in Section 3.

Notice that the original version of the basic algorithm is slightly more complex than the one presented above. We summarize it here to allow comparison with the algorithm of Section 4. A logical expression ϕ_1 for x_1 is first obtained by: (i) linearizing $g_1(x_2, x_3, \ldots, x_n)$, by replacing products of variables by new variables y_i $(i = 1, 2, \ldots, p)$, (ii) computing the characteristic function ϕ_1 of the linear inequality $g_1(y_1, y_2, \ldots, y_p) > 0$ (by definition, $\phi_1(y)$ is a Boolean function equal to 1 if $g_1(y) > 0$ and equal to 0 otherwise), (iii) eliminating the y_i from ϕ_1 and simplifying it through Boolean operations. Then, (iv) x_1 is replaced by ϕ_1 in $x_1 g_1(x_2, x_3, \ldots, x_n)$ and pseudo-Boolean simplifications are made, yielding $\psi_1(x_2, x_3, \ldots, x_n)$. The values x_i^* are recursively obtained from $\phi_i(x_{i+1}^*, x_{i+2}^*, \ldots, x_n^*)$. All optimal solutions may be obtained by analyzing the equality $g_1(y_1, y_2, \ldots, y_p) = 0$ in a similar way as the inequality $g_1(y_1, y_2, \ldots, y_p) > 0$ above, and introducing parameters.

3. Pseudo-Boolean functions with bounded tree-width

We use the graph-theoretic terminology of Bondy and Murty [4]. All graphs in this paper are simple and undirected. A vertex v is *simplicial* in a graph G if the neighbors of v induce a complete subgraph of G. For $k \geq 0$, a *k-perfect elimination scheme* of a graph G is an ordering $(v_1, v_2, \ldots, v_n)$ of its vertices, such that v_j is simplicial and has degree k in the subgraph G_j of G induced by $\{v_j, v_{j+1}, \ldots, v_n\}$ $(j = 1, 2, \ldots, n-k)$. A graph is a *k-tree* if it has a k-perfect elimination ordering. A *partial k-tree* is any graph obtained by deleting edges from a k-tree.

Since the complete graph on n vertices is an $(n-1)$-tree, every graph is also a

partial $(n-1)$-tree. The *tree-width* of a graph is the smallest value of k for which this graph is a partial k-tree (see [1, 19] for equivalent definitions). For instance, trees and forests have tree-width at most 1, and series-parallel graphs have tree-width at most 2. Arnborg, Corneil and Proskurowski [2] proved that determining the tree-width of a graph is NP-hard. However, for every fixed k, there is a polynomial algorithm to decide if a graph has tree-width k [1, 19]. In fact, the algorithm in [1] is constructive: if G has tree-width k, then the algorithm returns a k-perfect elimination scheme of some k-tree containing G as a subgraph. For brievity, we will refer to such a scheme as a k-scheme of G.

Let us now consider a pseudo-Boolean function $f(x_1, x_2, \ldots, x_n)$ expressed in polynomial form. The *co-occurrence* graph of f, denoted $G(f)$, has vertex set $V = \{x_1, x_2, \ldots, x_n\}$, with an edge between x_i and x_j $(i \neq j)$ if these variables occur simultaneously, either in direct or in complemented form, in at least one term of f. The *tree-width* of f is the tree-width of $G(f)$.

Remark. Strictly speaking, the graph $G(f)$ and its tree-width are not completely determined by f, but only by the particular expression considered for f. Nevertheless, it will be a convenient, and harmless abuse of language to speak of the graph and the tree-width of f.

For every fixed k, there is a polynomial-time algorithm to decide whether a given pseudo-Boolean function $f(x_1, x_2, \ldots, x_n)$ has tree-width at most k. Moreover, using the algorithm of Arnborg et al. [2], the variables of such a function can always be relabelled, so that $(x_1, x_2, \ldots, x_n)$ is a k-scheme of $G(f)$.

Theorem. *For every fixed k, the basic algorithm can be implemented to run in $O(n)$ time on those pseudo-Boolean functions $f(x_1, x_2, \ldots, x_n)$, expressed in polynomial form, for which $(x_1, x_2, \ldots, x_n)$ is a k-scheme of $G(f)$.*

Proof. Fix k, and let $f(x_1, x_2, \ldots, x_n)$ be as in the statement of the theorem.

(1) In order to carry out efficiently the computations required by the basic algorithm, we need a little bit of data structuring. Namely, we need a *list representation* of f consisting, for each variable x_i, of a list of the terms of f containing x_i but not $x_1, x_2, \ldots, x_{i-1}$, together with the coefficients of these terms $(i = 1, 2, \ldots, n)$. This representation can be obtained easily. Indeed, since x_1 has degree at most k in $G(f)$, it is contained in at most $2 \cdot 3^k$ terms of $G(f)$. Similarly, each variable x_i occurs in at most $2 \cdot 3^k$ terms not containing $x_1, x_2, \ldots, x_{i-1}$. Therefore, f has at most $2 \cdot 3^k n$ terms, i.e., $O(n)$ terms. Also, since $G(f)$ has no clique on $k+2$ vertices, all terms of f have at most $k+1$ variables. It follows that a list representation of f can be obtained in time $O(n)$, by scanning through the terms of f.

(2) Now, we show that, when applying the basic algorithm to $f_1 = f$, we can

obtain a polynomial expression of the function ψ_1 in constant time (where ψ_1 is as described in Section 2).

Indeed, an expression of g_1 is immediately available from the list representation of f. Let $N=\{x_j: j\in J\}$ be the set of neighbors of x_1, in $G(f)$. Then, g_1 depends only on the variables in N. Also, it is easy to see that:

$$\psi_1 = \sum \max(0, g_1(\alpha)) \prod_{j\in J} x_j^{\alpha_j}, \tag{1}$$

where the sum ranges over all vectors α in $\{0,1\}^{|J|}$.

Since $|J|\leq k$, and k is fixed, this expression of ψ_1 can be obtained in constant time. In fact, it is clear that we can even obtain in constant time a list representation of $f_2=\psi_1+h_1$.

(3) We claim that $f_2(x_2, x_3, \ldots, x_n)$ is a function of tree-width at most k, and that $(x_2, x_3, \ldots, x_n)$ is a k-scheme of $G(f)$.

To see this, let H denote a k-tree containing $G(f)$ as a subgraph, and such that $(x_1, x_2, \ldots, x_n)$ is a k-perfect elimination scheme of H. Notice that N induces a complete subgraph in H. Moreover, $G(f_2)$ is identical to $G(f)\setminus\{x_1\}$, except possibly for some edges between vertices of N (since g_1 and ψ_1 depend only on the variables in N). Therefore, $G(f_2)$ is also a subgraph of $H\setminus\{x_1\}$, and the claim follows.

(4) Since f_2 satisfies the hypotheses of the theorem, and its list representation is immediately available, we conclude as in step (2) that the second iteration of the basic algorithm takes again constant time.

Clearly, the same reasoning can be inductively extended to all iterations of the basic algorithm, including the n steps needed to trace back the optimal solution. Altogether, this leads to the $O(n)$ bound for the time complexity of the algorithm. □

This result admits straightforward extensions to various related problems in 0–1 variables. Consider for instance the well-known *satisfiability problem*, defined as follows. A clause $C_i=\prod_{j\in J_i} x_j^{\alpha_j}$ is a product of variables in direct or complemented form. The satisfiability problem for a set of clauses $C_1, C_2, \ldots, C_m$ is then to decide if there exists an assignment of 0–1 values to the variables such that all clauses are equal to 0. This last condition is satisfied if and only if the minimum of the pseudo-Boolean function $C_1+C_2+\cdots+C_m$ is equal to 0. Defining the *co-occurrence graph* of a set of clauses $C_1, C_2, \ldots, C_m$ in a similar way as for a pseudo-Boolean function immediately yields the following result.

Corollary. *For every fixed k, the basic algorithm can be implemented to run in $O(n)$ time on those satisfiability problems for which $(x_1, x_2, \ldots, x_n)$ is a k-scheme of the co-occurrence graph.*

It is also known that the basic algorithm can be generalized to handle *minimax*

problems of the form:

$$\min_x \max_y \quad f(x_1, \ldots, x_n, y_1, \ldots, y_m),$$

$$\text{s.t.} \qquad x_i \in \{0,1\} \ (i = 1, \ldots, n),$$

$$y_j \in \{0,1\} \ (j = 1, \ldots, m)$$

(see [12]). The resulting algorithm can again be implemented to run in polynomial time when applied to functions of bounded tree-width.

4. An implementation of variable elimination

4.1. The basic algorithm revisited

In this section we consider again the problem of maximizing a general pseudo-Boolean function, and propose a branch-and-bound approach to the variable elimination subproblem in the basic algorithm. Alternatives to this approach are to use the original basic algorithm summarized at the end of Section 2, or to apply the definition (1) of ψ, i.e., to enumerate all vectors of variables appearing in g_1 in direct or complemented form, compute the corresponding terms and group them using Boolean manipulations.

An intuitively appealing measure of the difficulty of solving pseudo-Boolean programming problems with the basic algorithm is the maximum number of variables in an expression g_i for $i = 1, 2, \ldots, n$ (which is equal to k if the function satisfies the conditions of the theorem of Section 3). Of course, if this number is fixed, all three variable elimination methods take constant time in worst case. But average times may differ considerably, as branch-and-bound is much quicker than complete enumeration.

Moreover, the proposed branch-and-bound algorithm differs from usual ones in that instead of yielding one or all optimal solutions to a problem, it yields a new equivalent expression of it, with one variable less. No further Boolean manipulations are then required.

Let us consider the function $f(x_1, x_2, \ldots, x_n)$ to be maximized and the variable x_1 to be eliminated. After replacing every occurrence of $\bar{x}_1$ by $1 - x_1$, and grouping terms in which x_1 appears, we have:

$$f(x_1, x_2, \ldots, x_n) = x_1 g_1(x_j : j \in J) + h_1(x_2, x_3, \ldots, x_n),$$

where J is the index set of the variables appearing in some term containing also x_1. We can then write g_1 in the form:

$$g_1 = c_0 + \sum_{j \in J} c_j x_j^{\alpha_j} + \sum_{t \in \Omega} c_t \prod_{j \in T(t)} x_j^{\alpha_{jt}}. \tag{2}$$

The variable elimination problem consists in determining a polynomial expression

of the pseudo-Boolean function ψ_1, defined by: $\psi_1 = \max(g_1, 0)$ for all Boolean vectors $(x_j: j \in J)$.

If we can show that g_1 is always positive we copy it out; if we can show that g_1 is never positive we delete it. Otherwise we branch by choosing a variable x_s, rewriting g_1 as:

$$g_1 = x_s \cdot g' + \overline{x_s} \cdot g''$$

where g' (respectively g'') is the restriction of g_1 induced by $x_s = 1$ (respectively x_s) and considering g' and g'' in turn.

A lower bound $\underline{g_1}$ on g_1 is readily obtained by summing the constant and all negative coefficients in (2):

$$\underline{g_1} = c_0 + \sum_{j \in J} \min(0, c_j) + \sum_{t \in \Omega} \min(0, c_t). \tag{3}$$

If $\underline{g_1}$ is nonnegative, then $\psi_1 = g_1$. Moreover, penalties p_j^1 and p_j^0 associated respectively with the fixation of a variable x_j at 1 or 0 can be derived:

$$p_j^1 = \max(\alpha_j c_j, (\alpha_j - 1)c_j) + \sum_{t \in \Omega \mid j \in T(t)} (1 - \alpha_{jt}) \max(-c_t, 0) \tag{4}$$

and

$$p_j^0 = \max(-\alpha_j c_j, (1 - \alpha_j)c_j) + \sum_{t \in \Omega \mid j \in T(t)} \alpha_{jt} \max(-c_t, 0). \tag{5}$$

These penalties can be added to $\underline{g_1}$ if x_j is fixed.

Moreover, we have the tighter lower bound:

$$\underline{\underline{g_1}} = \underline{g_1} + \max_{j \in J} \min(p_j^1, p_j^0). \tag{6}$$

Again, if $\underline{\underline{g_1}} \geq 0$, then $\psi_1 = g_1$. Similarly, an upper bound $\bar{g}_1$ on g_1 is provided by the sum of the constant and the positive coefficients in (2):

$$\bar{g}_1 = c_0 + \sum_{j \in J} \max(0, c_j) + \sum_{t \in \Omega} \max(0, c_t). \tag{7}$$

If $\bar{g}_1$ is nonpositive, then one can conclude that $\psi_1 = 0$. Penalties q_j^1 and q_j^0 associated with the fixation of variable x_j at 1 and 0, respectively, can be computed as follows:

$$q_j^1 = \max(-\alpha_j c_j, (1 - \alpha_j)c_j) + \sum_{t \in \Omega \mid j \in T(t)} (1 - \alpha_{jt}) \max(c_t, 0) \tag{8}$$

and

$$q_j^0 = \max(\alpha_j c_j, (\alpha_j - 1)c_j) + \sum_{t \in \Omega \mid j \in T(t)} \alpha_{jt} \max(c_t, 0). \tag{9}$$

These penalties can be subtracted from $\bar{g}_1$ if x_j is fixed.

A tighter upper bound is given by:

$$\overline{\bar{g}_1} = \bar{g}_1 - \max_{j \in J} \min(q_j^1, q_j^0), \tag{10}$$

and we have $\psi_1 = 0$ if $\overline{\bar{g}_1} \leq 0$.

Let us now state the precise rules of the algorithm for the computation of $\psi = \max(0, g(x_j: j \in J))$. We use here the branch-and-bound terminology of [14], with the following extended meaning: a *resolution test* exploits a sufficient condition for a particular formula to be the desired expression of the current pseudo-Boolean function: a *feasibility test* exploits a sufficient condition for the null function to be such an expression. We note by l the product of variables in direct or complemented form corresponding to the variables fixed at 1 and at 0 respectively in the current subproblem.

(a) *Initialization.* Set $\psi = \emptyset$, $l = 1$.

(b) *First direct feasibility test.* Compute $\bar{g}$ by (7). If $\bar{g} \leq 0$, then go to (i).

(c) *Second direct feasibility test.* Compute $\bar{\bar{g}}$ by (8), (9), (10). If $\bar{\bar{g}} \leq 0$, then go to (i).

(d) *First direct resolution test.* Compute $\underline{g}$ by (3). If $\underline{g} \geq 0$, then $\psi \leftarrow \psi + l \cdot g$ and go to (i).

(e) *Second direct resolution test.* Compute $\underline{\underline{g}}$ by (4), (5), (6). If $\underline{\underline{g}} \geq 0$, then $\psi \leftarrow \psi + l \cdot g$ and go to (i).

(f) *Conditional feasibility test.* If, for some $j \in J$, $\bar{g} - q_j^1 \leq 0$, set $l \leftarrow l \cdot \bar{x}_j$, $J \leftarrow J \setminus \{j\}$ and fix x_j to 0 in g. If, for some $j \in J$, $\bar{g} - q_j^0 \leq 0$, set $l \leftarrow l \cdot x_j$, $J \leftarrow J \setminus \{j\}$ fix x_j to 1 in g. If one variable at least has been fixed in this test, return to (b).

(g) *Conditional resolution test.* If, for some $j \in J$, $\underline{g} + p_j^1 \geq 0$, set $l \leftarrow l \cdot x_j$, $\alpha_j \leftarrow 1$, $J \leftarrow J \setminus \{j\}$, fix x_j to 1 in g, $\psi \leftarrow \psi + l \cdot g$; and go to (i) . If, for some $j \in J$, $\underline{g} + p_j^0 \geq 0$, set $l \leftarrow l \cdot \bar{x}_j$, $\alpha_j \leftarrow 0$, $J \leftarrow J \setminus \{j\}$, fix x_j to 0 in g, $\psi \leftarrow \psi + l \cdot g$, and go to (i).

(h) *Branching.* Choose a variable x_s to branch upon a branch, $\alpha_s = 1$ or $\alpha_s = 0$ (criteria for these choices are discussed in the next section). Set $l \leftarrow l \cdot x_s^{\alpha_s}$. $J \leftarrow J \setminus \{s\}$. Update g by setting $x_s^{\alpha_s}$ to 1. Return to (b).

(i) *Backtracking.* Find the last literal $x_s^{\alpha_s}$ chosen in either (f) or (h), for which the complementary value has not yet been explored. If there is none, stop. Otherwise, delete from l the literal $x_s^{\alpha_s}$ and the literals introduced after it, free the corresponding variables in g, update J, then fix $x_s^{\alpha_s}$ to 0 in g, set $l \leftarrow l \cdot x_s^{1-\alpha_s}$, $J \leftarrow J \setminus \{s\}$ and return to (b).

Adding a rule for the choice of the variable to be eliminated by the aforementioned algorithm, as well as the usual rules for obtaining the optimal solution from the expressions ψ_i, yields a variant of the basic algorithm, which we call basic algorithm revisited. We select for elimination the variable for which g_i contains the fewest variables. If f has width k but the k-scheme is unknown, we may use the same rule. This might however lead to some subproblem with a width larger than k. Whether this happens is explored empirically in the next section.

4.2. An example

Consider the maximization of the following pseudo-Boolean function f:

$$f(x) = -3x_1x_2\bar{x}_4 + 7x_3\bar{x}_6 + 2x_1x_2 - 9x_4x_5x_6 - 4x_5x_6 - 5\bar{x}_1x_6 - 2x_1\bar{x}_3x_6 + 2\bar{x}_2x_3x_5 - 6\bar{x}_4x_6 - 2\bar{x}_2 - 5x_4.$$

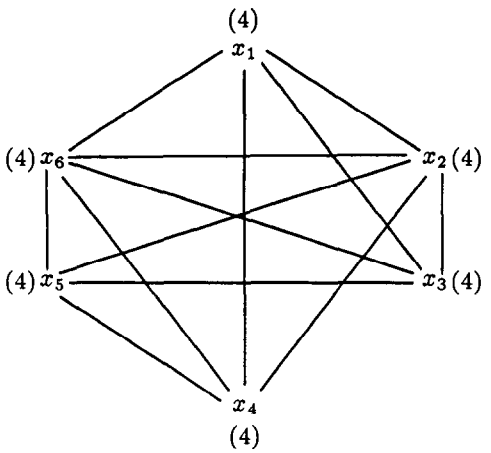

Fig. 1. Co-occurrence graph of f.

First we built the co-occurrence graph G_1 which is represented on Fig. 1.

For this example, we adopt the heuristic rule of eliminating the variable corresponding to the vertex of smallest degree in the current co-occurence graph. We break ties using the lexicographic order. All vertices of G_1 are of degree 4, so the lexicographic order gives x_1 to be eliminated. f is rewritten:

$$f(x) = x_1 g_1(x) + h_1(x)$$

where

$$g_1(x) = -3x_2\bar{x}_4 + 2x_2 + 5x_6 - 2\bar{x}_3 x_6,$$

$$h_1(x) = 7x_3\bar{x}_6 - 9x_4x_5x_6 - 4x_5x_6 - 5x_6 + 2\bar{x}_2x_3x_5 - 6\bar{x}_4x_6 - 2\bar{x}_2 - 5x_4.$$

The lower and upper bounds on $g_1(x)$ are respectively $\underline{g_1} = -5$ and $\bar{g}_1 = 7$. Since $\underline{g_1} < 0$ and $\bar{g}_1 > 0$, we compute the penalties. Their values are given in Table 1.

The conditional resolution test on the variable x_6 is successful:

$$\underline{g_1} + p_6^1 \geq 0,$$

so that:

$$\psi_1 \leftarrow x_6(-3x_2\bar{x}_4 + 2x_2 + 5 - 2\bar{x}_3)$$

and x_6 is fixed to 1.

Table 1. Penalties.

	p^1	p^0	q^1	q^0
x_2	2	3	0	2
x_3	2	0	0	0
x_4	3	0	0	0
x_5	0	0	0	0
x_6	5	2	0	5

Table 2. Penalties.

	p^1	p^0	q^1	q^0
x_2	2	3	0	2
x_4	3	0	0	0

Then g_1 is rewritten:

$$g_1(x) = -3x_2\bar{x}_4 + 2x_2,$$

$$\underline{g_1} = -3, \qquad \bar{g}_1 = 2.$$

Penalties are recomputed. Their new values are given in Table 2.

The conditional feasibility test on the variable x_2 is successful:

$$\bar{g}_1 - q_2^0 \leq 0,$$

so that x_2 is fixed to 1 and ψ_1 is unchanged.

Then the conditional resolution test on the variable x_4 leads to: $\underline{g_1} + p_4^1 \geq 0$, so that: $\psi_1 \leftarrow \psi_1 + 2x_4x_2\bar{x}_6$, and x_4 is fixed to 1.

g_1 is then equal to 2 and the iteration corresponding to the elimination of x_1 is finished. The minimization of the function f now reduces to the minimization of $f_2 = h_1 + \psi_1$, i.e.,

$$f_2(x) = -2x_6\bar{x}_3 + 7x_3\bar{x}_6 - 9x_4x_5x_6 - 4x_5x_6 + 2x_2x_6 + 2\bar{x}_2x_3x_5 - 6\bar{x}_4x_6 - 2\bar{x}_2 - 5x_4 - 3x_2x_6\bar{x}_4 + 2x_2x_4\bar{x}_6.$$

Notice that the linear terms in x_6 have disappeared due to the grouping of the two terms $-5x_6$ (in h_1) and $5x_6$ (in ψ_1).

Updating the co-occurrence graph leads to focus on the elimination of x_3, and f_2 is rewritten:

$$f_2(x) = x_3g_3(x) + h_3(x)$$

where

$$g_3(x) = 2x_6 + 7\bar{x}_6 + 2\bar{x}_2x_5 \geq 0.$$

The direct test $\underline{g_3} \geq 0$ allows to deduce immediately the function $f_3(x)$ after elimination of x_3:

$$f_3(x) = \psi_3(x) + h_3(x),$$

where

$$\psi_3(x) = g_3(x).$$

Then the algorithm successively eliminates

$$x_2 \quad \text{with} \quad \psi_2(x) = x_4(2x_6 + 2 + 2\bar{x}_6 - 2x_5) + \bar{x}_4\bar{x}_6(2 - 2x_5) + \bar{x}_5x_6\bar{x}_4,$$

$$x_4 \quad \text{with} \quad \psi_4(x) = 4\bar{x}_5x_6,$$

$$x_5 \quad \text{with} \quad \psi_5(x) = 0,$$

$$x_6 \quad \text{with} \quad \psi_6(x) = 0,$$

and ends up with $f_6(x) = 7$.

The optimal solution is recursively obtained

$$x_6^* = 0 \quad \text{since} \quad \psi_6(x) \leq 0,$$

$$x_5^* = 0 \quad \text{since} \quad \psi_5(x) \leq 0,$$

$$x_4^* = 0 \quad \text{since} \quad \psi_4(x) \leq 0,$$

$$x_2^* = 1 \quad \text{since} \quad \psi_2(x) > 0,$$

$$x_3^* = 1 \quad \text{since} \quad \psi_3(x) > 0,$$

$$x_1^* = 0 \quad \text{since} \quad \psi_1(x) \leq 0.$$

So $x^* = (0, 1, 1, 0, 0, 0)$ with $f^* = 7$.

5. Computational experience

As mentioned in the introduction, memory requirements have long been an obstacle to the implementation of the basic algorithm. Memory size still appears to be the limiting factor for in-core resolution of pseudo-Boolean programs with many variables but few in the functions g_i. So the representation of f, and the functions ψ_i, must be compact. Moreover, efficiency in the implementation of the algorithm of Section 4 implies computing bounds and penalties with a minimum number of operations. This means access to information of f must be possible through efficient data structures both by terms and by variables. Finally all information should be updated from one iteration to the next and not recomputed. We describe in the next subsection the data structures used to reach these objectives. In Subsection 5.2 we present computational results obtained with a FORTRAN 77 program for the basic algorithm revisited on a Sun 3/160S microsystem (with the same processor as a MacIntosh 2).

5.1. An efficient implementation of the basic algorithm revisited

Reducing the cpu time is usually the main concern in algorithm design. Here we are also faced with memory space problems, due to the increasing number of terms in the function f after elimination of some variables. We shall discuss three points: (i) internal representations of the functions f_i and g_i, (ii) updating of the co-occurrence graph, and (iii) grouping of terms involving the same variables.

It is useful to have two kinds of access to the function f: (a) given the index of

a term, list all the literals of this term; (b) given a variable x_i, list all the terms (and their coefficient) containing x_i or $\bar{x}_i$. So, we have adopted a double representation of f, chaining terms containing each variable and variables in each term. We also chain free spaces. Indeed, at each iteration one variable x_i is selected and the derivative of f with respect to this variable is computed. Then the terms of the derivative are removed from f and thus free some space here and there in the internal representation of f. Later some terms corresponding to the function ψ will be added to f. In order to save memory space, we chain the free spaces obtained when removing the terms of the derivative and insert the terms of ψ using those spaces first.

The variable to be eliminated corresponds to the vertex of minimal degree in the co-occurrence graph. In case of ties, we use the lexicographic order. So the co-occurrence graph must be updated at each iteration. This can be done by recomputing the degrees of the vertices corresponding to the variables involved in g_i. This is easy, using the second representation of f.

In order to save cpu time, the representation of the function g_i is not updated each time a variable is fixed in the branch-and-bound procedure, i.e., there is no compression to eliminate terms with a literal fixed at zero, and to reduce terms with a literal fixed at one. Instead, the effect of fixed literals is taken into account during the tests. Due to these fixations, several reduced terms of g_i might involve the same product of literals. These lead to obtaining several terms in ψ_i also involving the same product of literals. To avoid increase of the size of ψ_i due to such terms, they are merged, using an AVL-tree (see e.g. Knuth [16]). Moreover, to avoid having terms with the same product of literals in f, further merging is done between terms of ψ_i and those terms of h_i involving only variables of g_i. We note h_i' the sum of these latter terms.

Products of literals are coded by assigning to each of them a number corresponding to their position in the list of all products ranked first by increasing length and then by lexicographic order. To keep these numbers sufficiently small, we code only terms all variables of which appear in g_i. Complemented variables are coded as variables x_{l+n}. Literals of g_i are thus renumbered from 1 to $2k$. Each node of the AVL-tree has a value equal to the code number and a label equal to the coefficient of the corresponding term. Terms of h_i' are first inserted, then those of ψ_i as soon as they are obtained in the branch-and-bound procedure. The new expression of f (after elimination of x_i) is obtained by adding to $h_i - h_i'$ the terms contained in the AVL-tree.

5.2. *Experimental results*

The basic algorithm revisited has been tested on random pseudo-Boolean functions with tree-width at most k, for various fixed values of k. A partial k-tree $G=(X,E)$ on n vertices is first built in the following way [19]. Initially a set X_1 of at most $k+1$ vertices is chosen randomly. At iteration r, a subset X_r is constructed by: (i) choosing randomly a subset X_s among $X_1, X_2, \ldots, X_{r-1}$, (ii) choosing randomly in

X_s a subset X_s' of "attachment vertices", such that $0 < |X_s'| \leq k$, (iii) adding to $X_r = X_s'$ a random set X_s'' of new vertices (i.e., vertices not belonging to $X_1 \cup X_2 \cup \cdots \cup X_{r-1}$) such that $|X_r| = |X_s' \cup X_s''| \leq k+1$, $|X_s''| \geq 1$ and $|X_1 \cup X_2 \cup \cdots \cup X_r| \leq n$. Then a pseudo-Boolean function, the co-occurrence graph of which is a partial graph of G, is obtained by generating random pseudo-Boolean functions on each of the sets X_s.

In step (h) of the variable elimination procedure (Section 4.1), branching is done according to the following rule: the quantity $\delta = \max_{j \in J} \max(p_j^1, p_j^0, q_j^1, q_j^0)$ is computed; let $s \in J$ be the index achieving δ; then x_s is the next variable branched upon; if $\delta = p_s^1$ or $\delta = q_s^1$, then $\alpha_s = 0$; otherwise $\alpha_s = 1$.

Two orders of elimination have been compared: the first one corresponds to the selection at each iteration of the variable associated with the vertex of smallest degree and the second one follows the k-elimination scheme.

The results obtained are summarized in Table 3. Mean values and ranges over 10 problems are given for fixed values of n and k and increasing numbers of terms. The first two columns give the average number m of terms and the range of these numbers. The four last ones give, for both orders of elimination, the averages $\bar{m}$ of the maximum number of terms in f during resolution, and the cpu times in seconds.

The following conclusions can be drawn from these experiments: (i) problems with $k \leq 10$ and large n are readily solved on a small computer. Memory space appears to be the limiting factor; (ii) the maximum number of terms in the successive functions f_k is not much larger than in the original function f and only larger when $k \geq 7$; (iii) results obtained with the minimum degree selection rule are comparable to those obtained when following the k-elimination scheme. Only in rare cases does the minimum degree of one of the co-occurrence graphs exceed k when the former rule is used. This indicates that the basic algorithm revisited can be applied with this rule for maximizing pseudo-Boolean functions of bounded but unknown tree-width, and functions of known tree-width for which the k-elimination order is unknown.

Only comparisons with algorithms for related but slightly different problems than that one considered in this paper, can be made to evaluate the efficiency of the basic algorithm revisited. On the one hand, algorithms for nonlinear 0–1 programming may be considered. These are based on implicit enumeration (Lawler and Bell [17], Mao and Wallingford [18], Hansen [13], Taha [20, 21]) or cut generation and sequential resolution of generalized set covering problems (Granot and Granot [9], Balas and Mazzola [4, 5]). Only problems with up to 40 variables at most have been solved, but there are no bounds on the maximum degree of the co-occurrence graph, and there are constraints. On the other hand, while computational results for optimization problems solvable in linear time on partial k-trees do not appear to have been extensively published, Arnborg and Proskurowski [3] very recently estimated values for k for which such problems appear to be solvable in practive. They give two estimates, the first one being demonstrably feasible and the second of which "may be approached after careful analysis and implementation of the update step".

Table 3. Results obtained on bounded tree-width functions.

				Minimum degree		Elimination scheme	
n	k	m	range	$\bar{m}$	tcpu	$\bar{m}$	tcpu
		989	883–1057	989	19	989	18
	3	1482	1241–1598	1482	28	1482	27
		1786	1620–1982	1786	34	1786	34
		1351	1016–1667	1351	114	1351	120
100	7	2393	2061–2899	2393	189	2393	186
		3459	3003–4286	3459	268	3459	265
		700	455– 882	1005	619	1127	644
	10	1603	1188–1830	1987	1393	2182	1381
		2409	2026–2605	3054	2317	2984	2220
		753	576–1000	753	21	753	20
	3	1325	1158–1462	1325	34	1325	34
		1670	1550–1798	1670	43	1670	42
		1501	1066–1767	1510	175	1515	167
150	7	2233	2078–2489	2233	289	2241	262
		3392	3133–3777	3392	386	3397	376
		1528	1035–1989	2465	1937	2243	1813
	10	2407	2139–2938	3373	2727	3255	2845
		3380	3054–3501	4058	3765	4179	3553
		711	586– 848	711	26	711	25
	3	1176	1036–1350	1176	41	1176	40
		1675	1527–1882	1675	55	1675	55
		1368	1139–1945	1368	116	1375	107
200	7	2193	2031–2583	2193	189	2198	178
		3261	2916–3911	3261	282	3265	265
		686	417– 840	1202	734	1193	732
	10	993	908–1093	1488	1152	1505	994
		1625	1138–2557	2240	2207	2327	2091

For INDEPENDENT SET these estimates are 9 and 13, and for DOMINATING SET 4 and 8. Both these problems are easily expressed as pseudo-Boolean programming ones (Hammer and Rudeanu [12, Theorems 5 and 10 of Chapter X]) and can be solved in their weighted and unweighted versions, by the proposed algorithm.

Acknowledgement

We are grateful to Peter Hammer for pointing out the relevance of the basic algorithm to the result described in Section 3.

This research was partially supported by AFOSR grant # 0271 and by the NSF grant # ECS 85-03212 to Rutgers University, by a University of Delaware Research Foundation grant to the first author and by NSERC grant # GP0036426 and FCAR grant # 89EQ4144 to the third author.

References

[1] S. Arnborg, Efficient algorithms for combinatorial problems on graphs with bounded decomposability. A survey, BIT 25 (1985) 2–23.

[2] S. Arnborg, D.G. Corneil and A. Proskurowski, Complexity of finding embeddings in a k-tree, SIAM J. Algebraic Discrete Methods 8 (1987) 277–284.

[3] S. Arnborg and A. Proskurowski, Linear time algorithms for NP-hard problems restricted to partial k-trees, Discrete Appl. Math. 23 (1989) 11–24.

[4] E. Balas and J.B. Mazzola, Nonlinear 0-1 programming: I. Linearization techniques, Math. Programming 30 (1984) 1–21.

[5] E. Balas and J.B. Mazzola, Nonlinear 0-1 programming: II. Dominance relations and algorithms, Math. Programming 30 (1984) 22–45.

[6] F. Barahona, A solvable case of quadratic 0-1 programming, Discrete Appl. Math. 13 (1986) 23–26.

[7] J.A. Bondy and U.S.R. Murty, Graph Theory with Applications (Elsevier, New York, 1976).

[8] G. Georgakopoulos, D. Kavvadias and C.H. Papadimitriou, Probabilistic satisfiability, J. Complexity 4 (1988) 1–11.

[9] D. Granot and F. Granot, Generalized covering relaxation for 0–1 programs, Oper. Res. 28 (1980) 1442–1449.

[10] P.L. Hammer (Ivănescu), I. Rosenberg and S. Rudeanu, On the determination of the minima of pseudo-Boolean functions, Stud. Cerc. Mat. 14 (1963) 359–364 (in Romanian).

[11] P.L. Hammer (Ivănescu), I. Rosenberg and S. Rudeanu, Application of discrete linear programming to the minimization of Boolean functions, Rev. Math. Pures Appl. 8 (1963) 459–475 (in Russian).

[12] P.L. Hammer and S. Rudeanu, Boolean Methods in Operations Research and Related Areas (Springer, New York, 1968).

[13] P. Hansen, Programmes mathématiques en variables 0–1, Thése d'aggrégation, Université Libre de Bruxelles, Bruxelles (1974).

[14] P. Hansen, Les procédures d'optimisation et d'exploitation par séparation et évaluation, in: B. Roy, ed., Combinatorial Programming (Reidel, Dordrecht, 1975) 19–65.

[15] P. Hansen and B. Simeone, Unimodular functions, Discrete Appl. Math. 14 (1986) 269–281.

[16] D.E. Knuth, The Art of Computer Programming, Vol. 3: Sorting and Searching (Addison-Wesley, Reading, MA, 2nd ed., 1975).

[17] E.L. Lawler and M.D. Bell, A method for solving discrete optimization problems, Oper. Res. 14 (1966) 1098–1112.

[18] J.C.T. Mao and B.A. Wallingford, An extension of Lawler and Bell's method of discrete optimization with examples from capital budgeting, Management Sci. 15 (1968) 51–60.

[19] N. Robertson and P.D. Seymour, Graph minors: II. Algorithmic aspect of tree-width, J. Algorithms 7 (1986) 309–322.

[20] H. Taha, A Balasian-based algorithm for zero-one polynomial programming, Management Sci. 18B (1972) 328–343.

[21] H. Taha, Further improvements in the polynomial zero-one algorithm, Management Sci. 19 (1972) 226–227.