A Simple Enumerative Algorithm for Unconstrained $0-1$ Quadratic Programming

Pierre Hansen

École des Hautes Études Commerciales
GERAD & Département MQ
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7
pierreh@crt.umontreal.ca

Brigitte Jaumard

GERAD and École Polytechnique de Montréal
Département de Génie Électrique et de Génie Informatique
C. P. 6079, succ. Centre-ville
Montréal (Québec) Canada H3C 3A
brigitt@crt.umontreal.ca

Christophe Meyer

Institut für Mathematik B
Technische Universität Graz
Steyrergasse 30
A-8010 Graz, Austria
meyer@opt.math.tu-graz.ac.at

November, 2000

Les Cahiers du GERAD
G–2000–59

Abstract

We modify the algorithm of Pardalos and Rodgers [40] for the minimization of a pseudo-boolean quadratic function by introducing an easy to compute lower bound as well as a variable fixation test, based on the concept of roof-duality. Numerical results show that the new algorithm outperforms the original algorithm of Pardalos and Rodgers.

Keywords: quadratic $0-1$ programming, roof-duality, posiform.

Résumé

Nous modifions l'algorithme de Pardalos et Rodgers [40] pour la minimisation d'une fonction quadratique pseudo-Booléenne en introduisant une borne inférieure simple à calculer ainsi qu'un test de fixation de variables, basés sur le concept de dualité des toits. Les résultats numériques montrent que le nouvel algorithme est supérieur à l'algorithme original de Pardalos and Rodgers.

Mots clés: programmation quadratique $0-1$, roof-dualité, posiforme.

Acknowledgments: Work of the first two authors has been supported by FCAR (Fonds pour la Formation de Chercheurs et l'Aide à la Recherche) grant 2001-ER-70686. Work of the first author was also supported by NSERC (National Sciences and Engineering Research Council of Canada) grant GP0105574; work of the second author was also supported by NSERC grant GP0036426. Work of the third author was supported by NSERC-network grant NET0200815 and by the Spezialforschungsbereich F003 "Optimierung und Kontrolle", Projektbereich Diskrete Optimierung.

1 Introduction

Unconstrained quadratic $0-1$ programs appear in a great variety of applications (see the surveys [21][24][25][28][38]). As an example, solution of such problems are often required in columns generation methods for finding new columns with the appropriate reduced cost's sign (see e.g., Jaumard *et al.* [31][30] for applications in cellular telecommunication, and du Merle *et al.* [14] for applications to clustering).
The unconstrained quadratic $0-1$ minimization problem can be formulated as follows:

$$\min_{x\in\{0,1\}^n} \quad f(x) = \sum_{i=1}^{n} \left(a_{ii}x_i + \sum_{j=i+1}^{n} a_{ij}x_i x_j \right). \tag{1}$$

(Note that there is no loss of generality by assuming the absence of terms x_i^2 as $x_i^2 = x_i$ for all $x_i \in \{0,1\}$).
Various heuristics and exact methods have been proposed to solve this problem. Recent heuristics include [4, 5, 17, 34, 39, 42]. Exact methods can be approximatively separated in the following approaches: algebraic (or dynamic programming) methods (Hammer and Rudeanu [24], Crama *et al.* [13]), linearization followed by linear $0-1$ programming (Fortet [15, 16], Glover and Woolsey [18, 19]), cutting-plane algorithms (Granot and Granot [20], Adams and Sherali [3], Barahona, Jünger and Reinelt [6], Helmberg and Rendl [29]), reformulation to a continuous concave minimization problem (Konno [33], Kalantari and Bagchi [32], Thoai [41]) and enumerative algorithms (Hansen [26, 27], Lu [35], Pardalos and Rodgers [40], Billionnet and Sutter [9]). See also Carraresi, Farinaccio and Malucelli [12]. In this paper we focus on the enumerative methods. A well-known algorithm is then that one proposed by Pardalos and Rodgers [40]. It exploits systematically first order boolean derivatives, as we do here. The main difference with their algorithm is that we first transform the function to be minimized into a posiform. This yields a tighter bound that belongs to the family of roof-dual bounds, and an additional elimination test.

The paper is organized as follows. The lower bound and the variable fixation tests are described respectively in Section 2 and 3. In Section 4, we present formally the algorithm. This algorithm is then compared in Section 5 with that one of Pardalos and Rodgers on some difficult instances of the unconstrained quadratic $0-1$ problem.

2 Lower bound computation

2.1 The roof-dual lower bound

Let us introduce the complemented variables $\overline{x}_i = 1 - x_i$, $i = 1, \ldots, n$. Together with the variables $x_i, i = 1, \ldots, n$, they form the set of *literals*. A literal will be denoted by y_ℓ and we assume that we have the following correspondence: $y_{2i-1} = x_i$ and $y_{2i} = \overline{x}_i$ for $i = 1, \ldots, n$. Note that $\overline{\overline{y}}_\ell = y_\ell$ for all $\ell = 1, \ldots, 2n$.
Using the $2n$ literals, it is always possible to write $f(x) = \psi_0 + \psi(y)$ with

$$\psi(y) = \sum_{(k,\ell)\in T} w_{k\ell} y_k y_\ell + \sum_{k=1}^{2n} c_k y_k \tag{2}$$

where $T \subseteq \{(k,\ell), 1 \leq k < \ell \leq 2n\}$, $w_{k\ell} \geq 0$ for all $(k,\ell) \in T$, $c_k \geq 0$ for $k = 1, \ldots, 2n$ and ψ_0 is a constant not restricted in sign (see, e.g., Hammer *et al.* [23]). To do that, we use the equality

$$-x_i x_j \quad = \quad -x_j + \overline{x}_i x_j, \qquad 1 \leq i < j \leq n \tag{3}$$

to substitute all terms $x_i x_j$ whose coefficient is negative, and then using

$$-x_i \quad = \quad \overline{x}_i - 1, \qquad i = 1, \ldots, n, \tag{4}$$

we substitute all variables whose resulting coefficient is negative.
ψ is called a *quadratic posiform*. We will assume that $\min\{c_{2i-1}, c_{2i}\} = 0$ for $i = 1, \ldots, n$. If this is not the case, we can find another posiform ψ' and a constant $\psi'_0 > \psi_0$ such that $f = \psi' + \psi'_0$ by using the identity $x_i + \overline{x}_i = 1$.
As a consequence of the positivity of ψ, ψ_0 is a lower bound on $\min\limits_{x \in \{0,1\}^n} f(x)$. This class of lower bounds was studied in [23] where it was shown that the best possible bound, is identical to the bounds given by two different approaches (minorization by linear functions called *paved upper planes*, and continuous relaxation of a mixed-integer formulation). This result was extended in [10] to the case where posiforms of higher degree are considered (see also [36]). In [2], Adams and Dearing show how these techniques can be interpreted as taking the Lagrangean dual of some mixed-integer reformulation of the quadratic $0-1$ program (see also Adams, Billionnet and Sutter [1], Michelon and Maculan [37]). Billionnet and Sutter [9] exploit these techniques with posiforms of degree ≤ 4 to compute a good lower bound, which is incorporated in a best-first branch-and-bound algorithm.
The best possible bound ψ_0 is called the *roof-dual bound*.

2.2 Formulation of the roof-dual as a maximum flow problem

Hammer, Hansen and Simeone [23] showed that the roof-dual bound can be computed by solving a maximum flow problem in a special graph, called SAM graph, with $2n + m$ vertices and $2m + n$ arcs where $m = |T|$. Boros and Hammer [11] improved on this result by showing that the roof-dual bound can be computed by solving a maximum flow problem in a graph with $2n + 2$ nodes and $2(m + n')$ arcs where $n' \leq n$ is the number of non-null coefficients in the linear part of the initial symmetric posiform (we say that a posiform is symmetric if for each term $y_i y_j$ with a positive coefficient, the term $\overline{y}_i \overline{y}_j$ is also present with the same coefficient). In this paper, we give a more direct derivation of this maximum flow formulation, that works with any starting posiform (symmetric or not). Moreover we describe explicitly an algorithm for solving the maximum flow problem that exploits some of the properties of the network.

We first recall the formulation of the roof-dual. Let λ satisfy $0 \leq \lambda_{k\ell} \leq w_{k\ell}$, $(k,\ell) \in T$. Substituting

$$w_{k\ell} y_k y_\ell \quad = \quad \lambda_{k\ell}(\overline{y}_k \overline{y}_\ell + y_k - \overline{y}_\ell) + (w_{k\ell} - \lambda_{k\ell}) y_k y_\ell, \qquad (k,\ell) \in T$$

in (2), we obtain

$$\begin{aligned}\psi_\lambda(y) &= \sum_{(k,\ell)\in T} \lambda_{k\ell}\overline{y}_k\overline{y}_\ell + \sum_{(k,\ell)\in T}(w_{k\ell}-\lambda_{k\ell})y_k y_\ell \\ &\quad \underbrace{+\sum_{k=1}^{2n}\left(c_k + \sum_{\ell:(k,\ell)\in T}\lambda_{k\ell}\right)y_k - \sum_{k=1}^{2n}\left(\sum_{\ell:(\ell,k)\in T}\lambda_{\ell k}\right)\overline{y}_k}_{=L_\lambda(y)}\end{aligned}$$

Let $W \geq 0$ be a non-negative n-dimensional vector. Noting that $W_i = W_i x_i + W_i\overline{x}_i$ for $i = 1,\ldots,n$, the linear part $L_\lambda(y)$ can be rewritten

$$\begin{aligned}L_\lambda(y) &= -\sum_{i=1}^n W_i + \sum_{i=1}^n (d^i_\lambda x_i + \overline{d}^i_\lambda \overline{x}_i) \\ &= -\sum_{i=1}^n W_i + \sum_{i=1}^n \left(\min\{d^i_\lambda, \overline{d}^i_\lambda\}\right) + \sum_{i=1}^n \left(d^i_\lambda - \min\{d^i_\lambda, \overline{d}^i_\lambda\}\right)x_i + \sum_{i=1}^n \left(\overline{d}^i_\lambda - \min\{d^i_\lambda, \overline{d}^i_\lambda\}\right)\overline{x}_i\end{aligned}$$

where

$$\begin{aligned}d^i_\lambda &= W_i + c_{2i-1} + \sum_{\ell:(2i-1,\ell)\in T}\lambda_{2i-1,\ell} - \sum_{\ell:(\ell,2i)\in T}\lambda_{\ell,2i} \\ \overline{d}^i_\lambda &= W_i + c_{2i} + \sum_{\ell:(2i,\ell)\in T}\lambda_{2i,\ell} - \sum_{\ell:(\ell,2i-1)\in T}\lambda_{\ell,2i-1}.\end{aligned}$$

The roof-dual consists in maximizing the constant of the posiform, i.e.,

$$\begin{aligned}\text{(R-D)} \quad \max \quad & \rho_W(\lambda) = -\sum_{i=1}^n W_i + \sum_{i=1}^n \min\{d^i_\lambda, \overline{d}^i_\lambda\} \\ \text{s.t.} \quad & \left\{\ 0 \leq \lambda_{k\ell} \leq w_{k\ell}, \quad (k,\ell) \in T.\right.\end{aligned}$$

We now interpret this problem as a maximum flow problem in the graph $G_{\psi,W}$ defined as follows: the vertices of $G_{\psi,W}$ are the $2n$ literals plus a source s and a sink t. The arcs together with their capacities are described in Figure 1. Arcs with capacity equal to 0 are not introduced in the graph.

$$\begin{array}{llllll}
\overline{y}_k \xrightarrow{\left[\frac{w_{k\ell}}{2}\right]} y_\ell & \text{and} & \overline{y}_\ell \xrightarrow{\left[\frac{w_{k\ell}}{2}\right]} y_k & \text{for all } (k,\ell)\in T \\
s \xrightarrow{\left[\frac{W_i+c_{2i-1}}{2}\right]} x_i & \text{and} & \overline{x}_i \xrightarrow{\left[\frac{W_i+c_{2i-1}}{2}\right]} t & \text{for } i = 1,\ldots,n \\
s \xrightarrow{\left[\frac{W_i+c_{2i}}{2}\right]} \overline{x}_i & \text{and} & x_i \xrightarrow{\left[\frac{W_i+c_{2i}}{2}\right]} t & \text{for } i = 1,\ldots,n
\end{array}$$

Figure 1: Definition of the arcs

We define the value $F_W(\varphi)$ of a feasible flow φ as the sum of the constant $-\sum_{i=1}^{n} W_i$ plus the total flow arriving at t. We denote by F_W^* the maximum of $F_W(\varphi)$ over all feasible flows φ. Note that the problem of finding F_W^* is essentially a maximum flow problem in graph $G_{\psi,W}$. The graph $G_{\psi,W}$ has 2n+2 nodes and $2m + 4n$ arcs. We will see later that W_i can be taken equal to 0, which together with the assumption that $\min\{c_{2i-1}, c_{2i}\} = 0$ reduces the number of arcs in the graph to $2m + 2n$.

Observation 1 *Note that the arcs of $G_{\psi,W}$ come by pair. We say that a flow φ is* symmetric *if for each pair of arcs, the flow on the two arcs of the pair is equal. It is not difficult to see that there always exists an optimal flow that is symmetric. Indeed, let $\tilde{\varphi}$ be a feasible flow. Then we can build a feasible flow $\tilde{\varphi}'$ with same value that is the mirror of $\tilde{\varphi}$, i.e., such that for any arc, the value of $\tilde{\varphi}'$ on that arc is equal to the value of $\tilde{\varphi}$ on the other arc of the pair. The flow $\tilde{\varphi}'' = \frac{\tilde{\varphi}+\tilde{\varphi}'}{2}$ is still feasible, carries the same total flow from s to t as $\tilde{\varphi}$ and has the property that the flow passing on each arc of a pair is equal.*

Using this observation, we now show that the roof-dual is equivalent to the maximum flow problem in the graph $G_{\psi,W}$ under some condition on the vector W that will be relaxed later.

Proposition 1 *Assume that*

$$W_i \geq \max\left\{ \left(\sum_{\ell:(2i-1,\ell)\in T} w_{2i-1,\ell} + \sum_{\ell:(\ell,2i-1)\in T} w_{\ell,2i-1} \right), \left(\sum_{\ell:(2i,\ell)\in T} w_{2i,\ell} + \sum_{\ell:(\ell,2i)\in T} w_{\ell,2i} \right) \right\}$$

for $i = 1, \ldots, n$. Then the optimal value of problem (R-D) (i.e., the roof-dual bound) and the value of the maximum flow in graph $G_{\psi,W}$ are the same.

Proof:

Let us introduce the variables

$$u_i = W_i + c_{2i-1} - d^i_\lambda + \min\{d^i_\lambda, \overline{d}^i_\lambda\}, \qquad i = 1, \ldots, n \tag{5}$$

$$v_i = W_i + c_{2i} - \overline{d}^i_\lambda + \min\{d^i_\lambda, \overline{d}^i_\lambda\}, \qquad i = 1, \ldots, n \tag{6}$$

Substituting $\min\{d^i_\lambda, \overline{d}^i_\lambda\} = \frac{1}{2}\sum_{i=1}^{n}\left(u_i + v_i - 2W_i - c_{2i-1} + d^i_\lambda - c_{2i} + \overline{d}^i_\lambda\right)$ in the objective function of (R-D), we obtain:

$$\begin{aligned} \rho_W(\lambda) \;=\; & -\sum_{i=1}^{n} W_i + \frac{1}{2}\sum_{i=1}^{n}(u_i + v_i) \\ & + \underbrace{\sum_{i=1}^{n}\left(\sum_{\ell:(2i-1,\ell)\in T} \lambda_{2i-1,\ell} - \sum_{\ell:(\ell,2i)\in T} \lambda_{\ell,2i} + \sum_{\ell:(2i,\ell)\in T} \lambda_{2i,\ell} - \sum_{\ell:(\ell,2i-1)\in T} \lambda_{\ell,2i-1} \right)}_{=0}. \end{aligned}$$

Moreover by definition of W_i, we have $0 \leq u_i \leq W_i + c_{2i-1}$ and $0 \leq v_i \leq W_i + c_{2i}$ for $i = 1, \ldots, n$. We claim that problem (R-D) is equivalent to

$$\begin{array}{ll} \max & -\sum_{i=1}^{n} W_i + \frac{1}{2}\sum_{i=1}^{n}(u_i + v_i) \\ \text{s.t.} & \left\{ \begin{array}{l} u_i + \sum_{\ell:(2i-1,\ell)\in T} \lambda_{2i-1,\ell} + \sum_{\ell:(\ell,2i-1)\in T} \lambda_{\ell,2i-1} \\ \qquad = v_i + \sum_{\ell:(2i,\ell)\in T} \lambda_{2i,\ell} + \sum_{\ell:(\ell,2i)\in T} \lambda_{\ell,2i}, \qquad i = 1, \ldots, n \\ 0 \leq \lambda_{k\ell} \leq w_{k\ell}, \qquad (k,\ell) \in T \\ 0 \leq u_i \leq W_i + c_{2i-1}, \qquad i = 1, \ldots, n \\ 0 \leq v_i \leq W_i + c_{2i}, \qquad i = 1, \ldots, n \end{array} \right. \end{array} \tag{7}$$

Indeed, since the objective function tends to maximize $u_i + v_i$ for each $i = 1, \ldots, n$, at optimum either u_i or v_i will be at its upper bound. Assume that it is u_i. Then

$$\begin{aligned} v_i &= W_i + c_{2i-1} + \sum_{\ell:(2i-1,\ell)\in T} \lambda_{2i-1,\ell} + \sum_{\ell:(\ell,2i-1)\in T} \lambda_{\ell,2i-1} - \sum_{\ell:(2i,\ell)\in T} \lambda_{2i,\ell} - \sum_{\ell:(\ell,2i)\in T} \lambda_{\ell,2i} \\ &= W_i + c_{2i} - \overline{d}^i_\lambda + d^i_\lambda. \end{aligned}$$

Since v_i must be less than $W_i + c_{2i}$, this implies that $d^i_\lambda \leq \overline{d}^i_\lambda$, and hence that u_i and v_i satisfy their definition (5) and (6). A similar reasoning holds when v_i instead of u_i attains its upper bound.
Conversely, it is easy to see that any feasible solution to $(R - D)$ is feasible to (7) and has the same objective value.
Now (7) is nothing else than the formulation of the maximum flow problem with symmetric flow in graph $G_{\psi,W}$ with the variables $\frac{\lambda_{k\ell}}{2}, (k,\ell) \in T$ for the flow on the arcs $\overline{y}_k \to y_\ell$ and $\overline{y}_\ell \to y_k$, the variables $\frac{u_i}{2}$ for the flow on the arcs $s \to x_i$ and $\overline{x}_i \to t$, and finally the variables $\frac{v_i}{2}$ for the flow on the arcs $s \to \overline{x}_i$ and $x_i \to t$ for $i = 1, \ldots, n$. By Observation 1, the maximum flow problem with symmetric flow is equivalent to the general maximum flow problem. ∎

2.3 Interpretation in terms of posiform

In terms of posiform, a flow of 1 unit on an arc $y_k \to y_\ell$ can be interpreted as combining 1 unit of y_k and 1 unit of $\overline{y}_k y_\ell$ to produce 1 unit of y_ℓ and 1 unit of $y_k \overline{y}_\ell$ via the equality:

$$y_k + \overline{y}_k y_\ell = y_\ell + y_k \overline{y}_\ell \tag{8}$$

Note that this equality, written as

$$\overline{y}_\ell + \overline{y}_k y_\ell = \overline{y}_k + y_k \overline{y}_\ell \tag{9}$$

can also be interpreted as combining 1 unit of $\overline{y}_\ell$ and 1 unit of $\overline{y}_k y_\ell$ to produce 1 unit of $\overline{y}_k$ and 1 unit of $y_k \overline{y}_\ell$. This is why there are two arcs associated to each $(k,\ell) \in T$.
The capacities on the arcs $y_k \to y_\ell$ and $\overline{y}_\ell \to \overline{y}_k$ ensure that we cannot use a larger quantity

of $\overline{y}_k y_\ell$ than that available in the posiform ψ.
At the beginning, we have an amount $W_i + c_{2i-1}$ of x_i, and an amount $W_i + c_{2i}$ of $\overline{x}_i$ for $i = 1, \ldots, n$. The initial amount of x_i is separated in two parts: one half may be injected in the network to produce other literals, the other half can be used in conjunction with some amount of $\overline{x}_i$ produced by the network to increase the constant of the posiform by the equality $x_i + \overline{x}_i = 1$. The initial amount of $\overline{x}_i$ is divided similarly into 2 parts. The capacities on the arcs of the form $y_r \to t$ and $s \to y_r$, together with the conservation of flows, ensure that we do not use a larger amount of y_r and $\overline{y}_r$ than that which is available. The total flow carried from s to t corresponds to the increase of the constant.

Let $P : s \to y_{\ell_1} \to y_{\ell_2} \to \cdots \to y_{\ell_p} \to t$ be a path of G_ψ. From the symmetry of G_ψ, it follows that there exists a second path $P' : s \to \overline{y}_{\ell_p} \to \overline{y}_{\ell_{p-1}} \to \cdots \to \overline{y}_{\ell_1} \to t$: we will say that P' is the mirror path to P, and vice-versa. According to Observation 1, we can assume without loss of generality that a same flow is carried by the two paths. Let $\frac{\alpha}{2}$ be the flow carried by each path. By summing (8) and (9) over the arcs of these paths, we obtain:

$$y_{\ell_1} + \overline{y}_{\ell_p} + \sum_{r=1}^{p-1} \overline{y}_{\ell_r} y_{\ell_{r+1}} = 1 + \sum_{r=1}^{p-1} y_{\ell_r} \overline{y}_{\ell_{r+1}}$$

By definition of the capacities on the arcs and since α is a feasible flow,

$$\psi'(y) = \psi(y) - \alpha \left(y_{\ell_1} + \overline{y}_{\ell_p} + \sum_{r=1}^{p-1} \overline{y}_{\ell_r} y_{\ell_{r+1}} \right) + \alpha + \alpha \sum_{r=1}^{p-1} y_{\ell_r} \overline{y}_{\ell_{r+1}}$$

is still a posiform, with constant increased by α. The graph $G_{\psi',W}$ is obtained from $G_{\psi,W}$ by:

- Diminishing by $\frac{\alpha}{2}$ the capacity on the arcs $y_{\ell_r} \to y_{\ell_{r+1}}$ and $\overline{y}_{\ell_{r+1}} \to \overline{y}_{\ell_r}$ for $r = 1, \ldots, p-1$.
- Augmenting by $\frac{\alpha}{2}$ the capacity on the arcs $\overline{y}_{\ell_r} \to \overline{y}_{\ell_{r+1}}$ and $y_{\ell_{r+1}} \to y_{\ell_r}$ for $r = 1, \ldots, p-1$. Observe that these arcs are the reverse arcs of the previous ones.
- Diminishing by $\frac{\alpha}{2}$ the capacity on the arcs $s \to y_{\ell_1}$, $s \to \overline{y}_{\ell_p}$, $\overline{y}_{\ell_1} \to t$ and $y_{\ell_p} \to t$.

Observe that this graph is exactly the associated network used by the Ford-Fulkerson algorithm to find an augmenting path (see, e.g., Bertsekas [7]).

2.4 A maximum flow algorithm for computing the roof-dual bound

Our algorithm for computing the roof-dual bound is an adaptation of the Ford-Fulkerson algorithm for solving a maximum flow problem. Instead of storing the current flow somewhere, we compute at each iteration the resulting posiform (hence in a certain way, the current flow is stored in this posiform). The associated network in which we search for an augmented path is then simply the graph associated to the current posiform.

The algorithm uses in addition the two following results:

Proposition 2 *The optimal value of the flow problem does not depend upon W, i.e., $F^*_W = F^*_0$ for any $W \geq 0$.*

Proof:

For $i = 0, \ldots, n$, denote by $W^{(i)}$ the vector satisfying $W^{(i)}_j = 0$ for $j = 1, \ldots, i$ and $W^{(i)}_j = W_j$ for $j = i+1, \ldots, n$. Note that $W = W^{(0)}$ and $W^{(n)} = 0$. We will show that $F^*_{W^{(i+1)}} = F^*_{W^{(i)}}$ for $i = 0, \ldots, n-1$. Let i be fixed. If $W_i = 0$, the result is obvious, so assume that $W_i > 0$. Let φ^* be an optimal flow on $G_{\psi,W^{(i)}}$. In particular, $\varphi^*_{s\to x_i} = \varphi^*_{\overline{x}_i \to t} \leq \frac{c_{2i-1}}{2}$ and $\varphi^*_{s\to \overline{x}_i} = \varphi^*_{x_i \to t} \leq \frac{c_{2i}}{2}$. We can complete φ^* by a flow of $\frac{W_i}{2}$ units on the arcs $s \to x_i$, $s \to \overline{x}_i$, $x_i \to t$ and $\overline{x}_i \to t$ to obtain a feasible flow for $G_{\psi,W^{(i-1)}}$ with value $F^*_{W^{(i)}}$, hence $F^*_{W^{(i-1)}} \geq F^*_{W^{(i)}}$. Assume by contradiction that $F^*_{W^{(i-1)}} > F^*_{W^{(i)}}$, and let φ^* be the maximum flow associated to $F^*_{W^{(i-1)}}$. By Observation 1, we can assume that the flow φ^* is symmetric. We will construct a symmetric flow φ'^* feasible for $G_{\psi,W^{(i)}}$ with value $F^*_{W^{(i-1)}}$. Assume first that $\varphi^*_{s\to x_i} > 0$ and $\varphi'^*_{s\to \overline{x}_i} > 0$. Then there exist $\delta > 0$ and k and ℓ such that δ units of flow are carried to t through a path $P_1 : s \to x_i \to \cdots \to y_k \to t$, and another δ units are carried to t through a path $P_2 : s \to \overline{x}_i \to \cdots \to y_\ell \to t$. By symmetry of the flow, δ units of flow are carried through the path $P'_1 : s \to \overline{y}_k \to \cdots \to \overline{x}_i \to t$ and another δ units are carried through the path $P'_2 : s \to \overline{y}_\ell \to \cdots \to x_i \to t$. Note that these last two paths produce δ units of x_i and $\overline{x}_i$. Combining these paths respectively with P_2 and P_1, we define a new flow $\tilde{\varphi}$ that carries δ units of flow through the path $P_3 : s \to \overline{y}_k \to \cdots \to \overline{x}_i \to \cdots \to y_\ell \to t$ and another δ units through the mirror path $P'_3 : s \to \overline{y}_\ell \to \cdots \to x_i \to \cdots \to y_k \to t$, and we decrease by δ units the flow through the arcs $s \to x_i$, $s \to \overline{x}_i$, $x_i \to t$ and $\overline{x}_i \to t$. Let $\tilde{W}$ be the vector W, except for the i^{th} component that is decreased by 2δ. Then $F_{\tilde{W}}(\tilde{\varphi}) = F_{W^{(i-1)}}(\varphi^*)$. We iterate this procedure until the flow on one of the arc $s \to x_i$ or $s \to \overline{x}_i$ is 0. Assume that $\tilde{\varphi}_{s\to x_i} \neq 0$ (the proof is similar when $\tilde{\varphi}_{s\to \overline{x}_i} \neq 0$). Then there exists $\delta' > 0$ such that δ' units of flow are carried through a path $P_4 : s \to x_i \to \cdots \to y_m \to t$ and δ' units through a path $P'_4 : s \to \overline{y}_m \to \cdots \to \overline{x}_i \to t$. Let $\tilde{\varphi}'$ be the flow obtained by reducing by δ' the flow on all arcs of P_4 and P'_4. Then $\tilde{\varphi}'$ is a feasible flow for $G_{\psi,\tilde{W}'}$ such that $F_{\tilde{W}'}(\tilde{\varphi}') = F_{\tilde{W}}(\tilde{\varphi})$ where $\tilde{W}'_j = \tilde{W}_j$ for $j \neq i$ and $\tilde{W}'_i = \tilde{W}_i - 2\delta'$. This operation can be repeated until $\tilde{W}'_i = 0$ in which case $W^{(i)} = \tilde{W}'$. Let φ'^* be the flow $\tilde{\varphi}'$ at this stage. Then φ'^* is a feasible flow of $G_{\psi,W^{(i)}}$ and $F_{W^{(i)}} = F^*_{W^{(i-1)}}$, contradicting the assumption that $F^*_{W^{(i)}} < F^*_{W^{(i-1)}}$. ∎

By taking in Proposition 2 a W satisfying the condition of Proposition 1, and noting that in (2), the c_k can be chosen to satisfy $\min\{c_{2i-1}, c_{2i}\} = 0$ for $i = 1, \ldots, n$ (otherwise the equality $x_i + \overline{x}_i = 1$ can be applied), it follows that the roof-dual bound can be computed by solving a maximum flow problem in a graph with $2n+2$ vertices and at most $2(m+n)$ arcs.

Proposition 3 *If at some iteration of the Ford-Fulkerson algorithm, there exists no (augmenting) path using the arc $s \to y_k$ for some k, there will be no such path in the future iterations too.*

Proof:

Assume by contradiction that this is not true. Then at some iteration, there is no path from s to t using the arc $s \to y_k$, but a path $P : s \to y_{\ell_1} \to \cdots \to y_{\ell_p} \to t$ (and its mirror $\overline{P}$) has been found. And at the next iteration there exists a path starting with the arc $s \to y_k$.

Assume that this path is $P' : s \to y_k = y_{\ell'_1} \to y_{\ell'_2} \to \cdots \to y_{\ell'_q} \to t$. Since this path didn't exist at the previous iteration, some of the capacities on these arcs have been raised from 0 to a positive value after update of the posiform using the flow passing through the path P (if it was the path $\overline{P}$, simply invert the role of P and $\overline{P}$). This means that some arc of P' appeared in the opposite direction in P. Assume that the first such arc in P' is $y_{\ell'_i} \to y_{\ell'_{i+1}}$. Let $y_{\ell'_{i+1}} \to y_{\ell'_i} \to y_{\ell_r} \to y_{\ell_{r+1}} \to \cdots \to y_{\ell_p} \to t$ be the portion of the path P starting at this reverse arc. But then the path $s \to y_k \to \cdots \to y_{\ell'_i} \to y_{\ell_r} \to y_{\ell_{r+1}} \to \cdots \to y_{\ell_p} \to t$ existed at the previous iteration with strictly positive capacity on each arc. Since it starts by the arc $s \to y_k$, this contradicts our assumption. ∎

The procedure we now propose in order to compute the roof-dual lower bound scans iteratively the literals as a possible second vertex (after s) of an (augmenting) path. If a path is found, we compute the maximum flow that can pass through it and through its mirror, and we update the posiform and the associated graph. We then repeat this in the updated graph, and iterate until no path can be found. Proposition 3 tells us that when looking for such a path we can start the scanning of the literals at the literal that was considered last at the previous iteration.

The procedure **roofDualBound()** is described formally in Figure 2. A tolerance ε is used to construct the graph: the graph $G_{\psi,0,\varepsilon}$ is obtained from $G_{\psi,0}$ by removing the arcs whose capacity is less than ε.

In line 5, we use a breadth-first search to look for the shortest path. Since the path does not use an arc more than once (otherwise the path would not be the shortest one), the coefficients in ψ_P are 0, 1 or 2 (this latter case happens when the two arcs associated to a term are used: see Example 1). Hence the sum of the coefficients of the linear terms of the successive posiforms decrease by at list $\frac{\varepsilon}{2}$ at each iteration.

In line 10, $\overline{\psi}_P$ denotes the posiform obtained from ψ_P by complementing all literals.

We now illustrate this procedure on an example.

Example 1 *Assume that* $\varepsilon = 0$ *and consider the posiform* $\psi(x,\overline{x}) = 12x_2 + 4\overline{x}_2\overline{x}_3 + 8x_3\overline{x}_4 + 4x_1x_4 + 16\overline{x}_1x_3$. *The graph* $G_{\psi,0,0}$ *is given in Figure 3. The numbers into brackets are the capacities of the arcs. A path from* s *to* t *is* $P_1 : s \to x_2 \to \overline{x}_3 \to \overline{x}_4 \to x_1 \to x_3 \to \overline{x}_2 \to t$. *The posiform associated to it is* $\psi_{P_1}(x,\overline{x}) = \overline{x}_2\overline{x}_3 + x_3\overline{x}_4 + x_4x_1 + \overline{x}_1x_3 + \overline{x}_3\overline{x}_2 = 2\overline{x}_2\overline{x}_3 + x_3\overline{x}_4 + x_1x_4 + \overline{x}_1x_3$. *The greatest* δ *such that* $\psi(x,\overline{x}) - \delta\,(\psi_{P_1}(x,\overline{x}) + x_2 + x_2)$ *is a posiform is* $\overline{\delta} = 2$. *The new posiform* ψ' *is*

$$\begin{aligned}\psi'(x,\overline{x}) &= \psi(x,\overline{x}) - 2\,(x_2 + x_2 + \psi_{P_1}(x,\overline{x})) + 2\overline{\psi}_{P_1}(x,\overline{x}) \\ &= 8x_2 + 6x_3\overline{x}_4 + 2x_1x_4 + 14\overline{x}_1x_3 + 4x_2x_3 + 2\overline{x}_3x_4 + 2\overline{x}_1\overline{x}_4 + 2x_1\overline{x}_3\end{aligned}$$

0 INPUT: posiform $\psi(y) = \sum_{r=1}^{2n} c_r y_r + \sum_{r=1}^{2n} \sum_{s=r+1}^{2n} w_{rs} y_r y_s$,
lower bound ψ_0, tolerance $\varepsilon \geq 0$
1 Construct the graph $G_{\psi,0,\varepsilon}$
2 $r = 1$
3 **while** $(r \leq 2n)$ **do**
4 **if** $(c_r > \varepsilon)$ **then**
5 Look for a shortest path $P = (y_{r_1} = y_r, y_{r_2}, \ldots, y_{r_p} = \overline{y}_s)$ joining y_r to $\overline{y}_s$ for some s such that $c_s > \varepsilon$.
7 **if** (found) **then**
8 Construct the posiform ψ_P: $\psi_P(y) = \sum_{\ell=1}^{p-1} \overline{y}_{r_\ell} y_{r_{\ell+1}}$
9 Let $\overline{\delta}$ be the greatest δ such that $\psi - \delta(\psi_P + y_r + y_s)$ is a posiform
10 $\psi \leftarrow \psi - \overline{\delta}(\psi_P + y_r + y_s) + \overline{\delta}\,\overline{\psi}_P$
11 $\psi_0 \leftarrow \psi_0 + \overline{\delta}$
12 Update the graph $G_{\psi,0,\varepsilon}$
13 **else**
14 $r \leftarrow r + 1$
15 **end if**
16 **end if**
17 **end while**

Figure 2: Procedure **roofDualBound**$(\psi, \psi_0, \varepsilon)$

The graph $G_{\psi',0,0}$ is given in Figure 4. There is no path from s to t thus 2 is the roof dual bound for the $0-1$ minimum of the quadratic function corresponding to the posiform ψ.

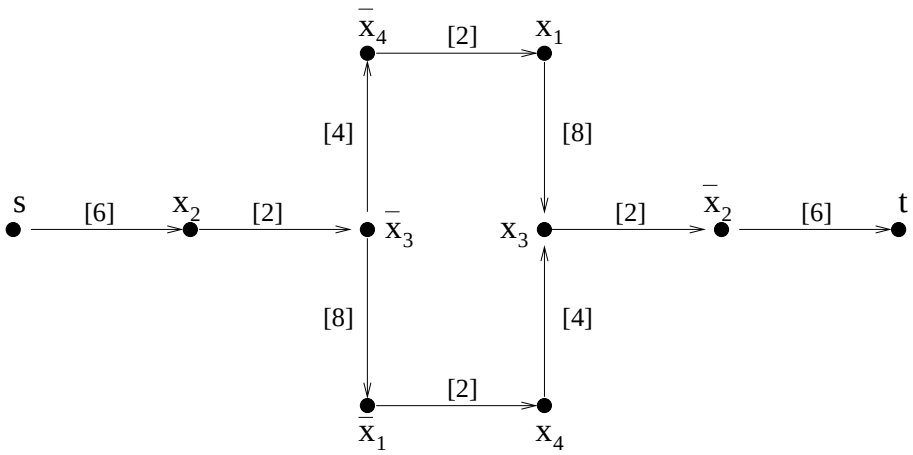

Figure 3: Graph $G_{\psi,0,0}$

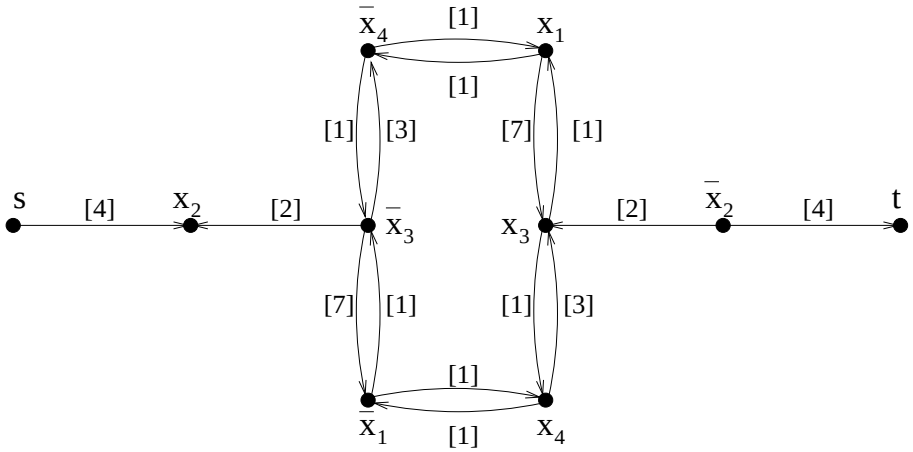

Figure 4: Graph $G_{\psi',0,0}$

In the algorithm to be given in Section 4, we compute the best possible posiform, i.e., the roof-dual bound, only at the first node of the search tree. At the subsequent nodes, we update the posiform by exploiting equalities $x_i + \overline{x}_i = 1$, i.e. we consider only paths of length 2 from s to t (this situation will appear when the posiform contains terms x_i and $\overline{x}_i x_j$ with a positive coefficient and we fix x_j to 1).

3 Variable fixation test

3.1 Fixation of variables by the "persistency" result

When the posiform associated to the roof-dual bound has a non-null linear part, all literals with a non-null coefficient in this linear part must have value 0 in an optimal solution. This is the so-called persistency result: see [23],[8].
Note that by Proposition 3, y_k could be fixed to 0 as soon as we establish that there is no augmenting path starting with the arc $s \rightarrow y_k$, and the coefficient of y_k is positive.
Note however that this fixation test is applicable only at the root of the branching tree since we will not solve exactly the roof-dual at optimality at the other nodes.

3.2 Fixation of variables using the lower bound

The fixation of variables to 0 or to 1 by branching or by reduction tests can create new linear terms in the posiform, that can be used via relations of the form $x_i + \overline{x}_i = 1$ to improve the lower bound. Conversely the lower bound resulting from the posiform can be used to fix some variables. Let $\overline{f}$ be the value of the best known solution, and let i be the index of a candidate variable for fixing. Let q_i and $\overline{q}_i$ be respectively the coefficient of x_i and $\overline{x}_i$ in the posiform ψ, so that

$$\psi(x, \overline{x}) = \psi_0 + q_i x_i + \overline{q}_i \overline{x}_i + \dots$$

where the unexplicited part is a posiform not containing x_i and $\overline{x}_i$ in its linear part. The constant ψ_0 is the current lower bound. We can assume that $q_i\overline{q}_i = 0$, otherwise we could improve ψ_0 using the equality $x_i + \overline{x}_i = 1$. Now the fixation of x_i to 1 would increase the lower bound by q_i. If $\psi_0 + q_i \geq \overline{f}$, it would not be possible to find an optimal solution with value strictly less than $\overline{f}$, hence x_i can be fixed to 0.
Similarly the fixation of x_i to 0 would increase the lower bound by $\overline{q}_i$. If $\psi_0 + \overline{q}_i \geq \overline{f}$, this fixation cannot yield a strictly better solution, hence x_i can be fixed to 1.

3.3 Fixation of variables using the range of the partial derivatives

Another way to fix variables is to consider the partial derivatives of f. This is already done in Pardalos and Rodgers algorithm [40]. We recall it for completeness. Let Δ_i be the partial derivative of f with respect to x_i. To simplify the notations, we will assume from now on that the lower triangular part of the matrix A (which is not used in the definition of (1)) is filled with the elements of the upper triangular part, that is $a_{ji} = a_{ij}$ for all $j > i$. Moreover, we define $a_{ij}^- = \min\{a_{ij}, 0\}$ and $a_{ij}^+ = \max\{a_{ij}, 0\}$. We have

$$\Delta_i(x) = a_{ii} + \sum_{j=1, j\neq i}^{n} a_{ij}x_j. \tag{10}$$

Observe that $\Delta_i(x)$ does not depend upon x_i. It is easy to show the following result (see, e.g., Hammer and Hansen [22]):

Proposition 4
a) If $\Delta_i(x) > 0$ for all x, then $x_i = 0$ in all optimal solutions.
b) If $\Delta_i(x) < 0$ for all x, then $x_i = 1$ in all optimal solutions.
c) If $\Delta_i(x) = 0$ for all x, then there exist an optimal solution with $x_i = 0$ and an optimal solution with $x_i = 1$.

Observe that $\underline{\Delta}_i \leq \Delta_i(x) \leq \overline{\Delta}_i$ with

$$\underline{\Delta}_i = a_{ii} + \sum_{j=1, j\neq i}^{n} a_{ij}^- \tag{11}$$

$$\overline{\Delta}_i = a_{ii} + \sum_{j=1, j\neq i}^{n} a_{ij}^+. \tag{12}$$

It follows from Proposition 4 that x_i can be fixed to 0 if $\underline{\Delta}_i \geq 0$ and to 1 if $\overline{\Delta}_i \leq 0$.

4 Algorithm

Given an upper triangular matrix $A = \{a_{ij}\}_{1\leq i\leq j\leq n}$, the algorithm we propose finds an optimal solution of problem (1).

The following variables are used in the description of the algorithm:

a_{ij}, for $j \geq i$, entry of the matrix A; for $j < i$, we assume that $a_{ij} = a_{ji}$.
$a_{ij}^+ = \max\{a_{ij}, 0\}$, $a_{ij}^- = \min\{a_{ij}, 0\}$.
$\overline{x}$ and $\overline{f} = f(\overline{x})$, the best known minimizer and its value.
lev, the current level in the branch and bound tree.
p, the permutation vector. $p_1, \ldots, p_{lev}$ are the indices of the fixed variables and $p_{lev+1}, \ldots, p_n$ the indices of the free variables.
x, the current solution.
ψ, the current posiform.
ψ_0, the current lower bound.
$\underline{\Delta}_i$, a lower bound on the partial derivative of f with respect to x_i.
$\overline{\Delta}_i$, an upper bound on the partial derivative of f with respect to x_i.
stack, a stack to hold the subproblems when branching is necessary.
update, a logical variable that indicates if the bounds on the partial derivatives can be updated, or if they have to be recomputed from scratch.

1 $(\overline{f}, \overline{x}) \leftarrow$ best known minimum and minimizer
2 $p_i \leftarrow i$ and $x_i \leftarrow 0$, $i = 1, \ldots, n$
3 **push**($\{-1, -, -\}$, stack)
4 $lev \leftarrow -1$; $update \leftarrow false$
5 Calculate a posiform $\psi(x, \overline{x}) = \sum_{i=1}^{n} (q_i x_i + \overline{q}_i \overline{x}_i) + \sum_{i=1}^{n} \sum_{j=i+1}^{n} (q_{ij} x_i x_j + \overline{q}_{ij} \overline{x}_i \overline{x}_j + q'_{ij} x_i \overline{x}_j)$
and a constant ψ_0 such that $f = \psi_0 + \psi$ as explained in Section 2.1.
6 **while** $(true)$ **do**
7 **if** $((\psi_0 \geq \overline{f})$ **or** $(lev = n))$ **then**
8 **if** $(\psi_0 < \overline{f})$ **then** $(\overline{f}, \overline{x}) \leftarrow (\psi_0, x)$ **end if**
9 **pop**($\{lev, \psi, \psi_0\}$, stack)
10 **if** $(lev = -1)$ **then** stop **end if**
11 $x_{p_{lev}} \leftarrow 1 - x_{p_{lev}}$; $update \leftarrow false$
12 Update the linear part of the posiform ψ and the lower bound ψ_0
13 **else**
14 **if** $(\psi_0 + max\{q_{p_i}, \overline{q}_{p_i}\} \geq \overline{f}$ for some $i, i = lev + 1, \ldots, n)$ **then**
15 **if** $(\psi_0 + q_{p_i} \geq \overline{f})$ **then** $x_{p_i} \leftarrow 0$ **else** $x_{p_i} \leftarrow 1$ **end if**
16 $update \leftarrow false$
17 **else**
18 **if** $(update = true)$ **then**
19 **if** $(x_{p_{lev}} = 1)$ **then**
20 $\underline{\Delta}_{p_i} \leftarrow \underline{\Delta}_{p_i} + a^+_{p_i p_{lev}}$; $\overline{\Delta}_{p_i} \leftarrow \overline{\Delta}_{p_i} + a^-_{p_i p_{lev}}$, $i = lev + 1, \ldots, n$
21 **else**
22 $\underline{\Delta}_{p_i} \leftarrow \underline{\Delta}_{p_i} - a^-_{p_i p_{lev}}$; $\overline{\Delta}_{p_i} \leftarrow \overline{\Delta}_{p_i} - a^+_{p_i p_{lev}}$, $i = lev + 1, \ldots, n$
23 **end if**
24 **else**

25 $\quad\quad \underline{\Delta}_{p_i} \leftarrow a_{p_ip_i} + \sum_{j=1}^{lev} a_{p_ip_j}x_{p_j} + \sum_{j=lev+1, j\neq i}^{n} a^{-}_{p_ip_j}, \quad i = lev+1, \ldots, n$

26 $\quad\quad \overline{\Delta}_{p_i} \leftarrow a_{p_ip_i} + \sum_{j=1}^{lev} a_{p_ip_j}x_{p_j} + \sum_{j=lev+1, j\neq i}^{n} a^{+}_{p_ip_j}, \quad i = lev+1, \ldots, n$

27 **end if**
28 $update \leftarrow true$
29 **if** ($\underline{\Delta}_{p_i} \geq 0$ or $\overline{\Delta}_{p_i} \leq 0$ for some $i, i = lev+1, \ldots, n$) **then**
30 **if** ($\underline{\Delta}_{p_i} \geq 0$) **then** $x_{p_i} \leftarrow 0$ **else** $x_{p_i} \leftarrow 1$ **end if**
31 **else**
32 $i \leftarrow j$ where $\min(-\underline{\Delta}_{p_j}, \overline{\Delta}_{p_j}) = \max_k \{\min(-\underline{\Delta}_{p_k}, \overline{\Delta}_{p_k}), k = lev+1, \ldots, n\}$
33 $x_{p_i} \leftarrow 0$ or 1 depending on the value that increases ψ_0 least
34 **push**($\{lev+1, \psi, \psi_0\}$, stack)
35 **end if**
36 **end if**
37 $lev \leftarrow lev + 1$
38 $p_{lev} \leftrightarrow p_i$
39 Update the linear part of the posiform ψ and the lower bound ψ_0
40 **end if**
41 **end while**

In line 5 we calculate the roof-dual lower bound and the associated posiform by using the procedure **roofDualBound** (see Section 2.1). If the lower bound is greater than or equal to the value of the best known solution, we are done. Otherwise we jump to line 14. We check for the existence of a variable for which the fixation to 0 or 1 would give a lower bound greater than or equal to the value of the best known solution. If such a variable is found, we fix it to the opposite value. Otherwise in lines 18 through 28, we recompute the bounds on the partial derivatives of the free variables: if $update = true$, i.e., if only one variable has changed its value since the last computation, we use the update procedure of Pardalos and Rodgers [40]; otherwise the recomputation is done from scratch. We then check if a variable can be fixed by using the partial derivatives (see Section 3.3). If no such variable exists, we have to branch.
In line 32, the selection criterion corresponds to choosing a free variable which is the farthest from being fixed by the test on the partial derivative. If several variables have the same value for this criterion, we choose the variable that maximizes $\max\{-\underline{\Delta}_{p_k}, \overline{\Delta}_{p_k}\}$. Additional ties are broken by taking the variable of smallest index. In line 33, we evaluate the lower bound in the case where the selected variable is fixed to 0 and where it is fixed to 1, and we fix the variable to the value that gives the smallest lower bound. The current subproblem is stacked in line 34 if the lower bound of the alternate subproblem is not greater than the value of the best known solution. For each subproblem, we store in the stack its level, the linear part of the posiform and the lower bound; the quadratic part of the posiform is never modified after line 5, thus need not be stored at each branching. In line 37-38, we increase lev and update the permutation vector in line 31. In line 39, the update of the posiform and of the lower bound is done as follows:

if $x_{p_i} = 1$ **then**
  for each term $q_{p_ip_j}x_{p_i}x_{p_j}$ of ψ add $q_{p_ip_j}$ to q_{p_j} **end for**
  for each term $q'_{p_ip_j}x_{p_i}\overline{x}_{p_j}$ of ψ add $q'_{p_ip_j}$ to $\overline{q}_{p_j}$ **end for**
else
  for each term $q'_{p_jp_i}x_{p_j}\overline{x}_{p_i}$ of ψ add $q'_{p_jp_i}$ to q_{p_j} **end for**
  for each term $\overline{q}_{p_ip_j}\overline{x}_{p_i}\overline{x}_{p_j}$ of ψ add $\overline{q}_{p_ip_j}$ to $\overline{q}_{p_j}$ **end for**
end if
for $j = lev + 1$ **to** n **do**
  $m \leftarrow \min\{q_{p_j}, \overline{q}_{p_j}\}$
  $\psi_0 \leftarrow \psi_0 + m, \quad q_{p_j} \leftarrow q_{p_j} - m, \quad \overline{q}_{p_j} \leftarrow \overline{q}_{p_j} - m$
end for

If in line 7 the lower bound is greater than or equal to the upper bound, or if all variables have been fixed, we try to update the best minimizer (note that if all variables have been fixed, ψ_0 is equal to the value of the solution). In line 9, we retrieve a subproblem of the stack. If we get the end-of-stack marker -1, the algorithm terminates. Otherwise we change the value of $x_{p_{lev}}$, update the posiform and the lower bound, and return to line 7.

5 Computational experiments

We compare our algorithm with that one of Pardalos and Rodgers [40], which is similar to ours, except that it uses the simpler lower bound $\sum_{i=1}^{n}\sum_{j=i}^{n} a_{ij}^{-}$ and does not try to exploit it to fix variables.

We have done 2 series of experiments. In each case we measured the total CPU time in seconds CPU, the number of iterations *iter* (number of passages through the external **while** loop), the total number of subproblems stored in the stack *nbSubp*, and the gap between the initial lower bound and the optimal solution in percentage of the value of the optimal solution $\%gap$.

In the first experience, we have selected 13 pairs (N, d) where N is the number of variables and d the density, and for each pair we have generated 10 instances using the test problem generator of Pardalos and Rodgers [40] (with seeds 1, 2, ..., 10). The pairs (N, d) correspond to those of Table 5.4 of Pardalos and Rodgers [40] with a reported CPU time greater than 10s. We run our program and the one of Pardalos and Rodgers on a *Sun-Sparc SS20/514MP* with 128 MRam for two choices of the initial solution: the solution $(0, 0, \ldots, 0)$ and the optimal one (supposed to be known). Note that these choices can simulate the extreme cases arising when using the programs in combination with heuristics: a very bad heuristic and a very good heuristic. Table 1 reports the minimum, the average and the maximum of each indicator for the two programs with initial solution 0. Table 2 gives the same results but with the optimal solution as initial solution. Pardalos and Rodgers algorithm stops when the number of iterations exceeds $134,217,728$. When this occurred, we indicated the number of instances for which the maximal number of iterations was attained. When this number exceeds 5, we do not provide the CPU time for Pardalos and Rodgers algorithm. We also give the number of instances for which this

maximal number of iterations is attained when they are solved with our program, although our program does not stop.
In the second experience, we considered the 7 difficult instances reported in Pardalos and Rodgers [40] for which they give the seed (which allows an exact replication of the instances). Table 3 and Table 4 give respectively the results when the programs are initialized with the 0 solution and with the optimal solution. Note that because of the definition of these instances, it is likely that our program will perform better than Pardalos and Rodgers one.

We observe that our algorithm is in average faster by a factor 3 through 18 (series $(60, .2)$, Table 2) compared to Pardalos and Rodgers's one. Because of the difference of programming language (*Fortran* versus *C*), this must be taken with some precaution. However the superiority of our algorithm is also illustrated by the indicators *iter* and *nbSubp*. In particular the number of iterations is reduced by a factor 10 to 67 (series $(70, .2)$, Table 2). Since the two algorithms differ essentially by the lower bound, it is natural to think that the lower bounds may explain these differences in the results. Indeed, the last indicator $\%gap$ shows that the lower bound used by our algorithm is far better than Pardalos and Rodgers's one at the root node. This certainly helps to cut off many branches in the search tree.
The impact of the lower bound is also visible when comparing Tables 1 and 2. Whereas the knowledge of the optimal solution does not help Pardalos and Rodgers's algorithm (for some of the series, their algorithm is even slightly faster when started with the 0 solution), our algorithm is always faster when the optimal solution is available, and in some cases the decrease of the CPU time is significant (e.g. 53.9% for $(60, .3)$, 60.8% for $(70, .2)$). Note that for the Pardalos and Rodgers algorithm, we find CPU time greater than those reported in Table 5.4 of [40]. This can be explained by differences in the performance of the computers (Pardalos and Rodgers used a IBM 3090), by differences in the set of instances chosen or possibly by the use of a variant optimized to solve sparse problems (we used the variant of Pardalos and Rodgers algorithm for dense problems in all cases).

As expected, we solve the "difficult" problems faster than does Pardalos and Rodgers's algorithm.

		CPU		iter		nbSubp		%gap	
		Our	P&R	Our	P&R	Our	P&R	Our	P&R
(40, .5)	min	0.1	1.0	854	46,296	102	2,747	15.88	152.10
	av.	4.3	15.8	61,305	725,855	10,752	66,213	54.28	233.26
	max	12.7	41.3	184,520	1,920,716	33,132	216,827	105.87	348.51
(40, .6)	min	0.2	5.1	2,999	241,055	458	12,980	25.23	167.50
	av.	11.4	63.9	151,033	2,889,590	31,047	287,004	67.31	264.15
	max	48.0	311.4	624,234	14,638,030	143,253	1,569,259	113.73	369.87
(40, .7)	min	1.4	15.0	18,524	681,486	2,859	39,398	40.80	200.40
	av.	16.5	112.0	204,771	5,035,010	42,428	430,616	80.23	287.33
	max	37.8	192.5	470,838	8,779,315	99,166	750,565	120.81	385.13
(40, .8)	min	5.2	38.5	56,387	1,746,277	12,927	124,753	59.65	233.47
	av.	32.2	231.1	412,056	10,276,146	88,316	909,046	91.53	305.60
	max	95.9	621.2	1,266,204	28,415,315	289,066	2,470,731	140.26	423.83
(50, .4)	min	0.9	13.3	7,953	494,753	1,215	26,020	23.10	167.99
	av.	46.4	317.8	451,566	11,598,976	79,025	1,022,244	55.27	238.89
	max	111.5	1,156.2	1,169,908	41,986,719	196,661	4,274,258	103.42	349.58
(50, .5)	min	3.1	110.2	27,069	4,202,352	4,512	199,570	32.30	187.85
	av.	179.7	1,317.9	1,805,529	48,757,860	347,952	4,257,387	73.75	277.33
	max	749.8	3,519.9	7,715,622	134,217,728[1]	1,558,677	13,620,698	151.63	453.93
(50, .6)	min	4.1	136.9	40,258	4,662,069	5,694	264,974	37.28	201.81
	av.	636.6	-	6,295,069	115,022,936	1,328,707	8,655,762	88.41	299.38
	max	1,897.8	-	18,265,029	134,217,728[7]	4,013,542	11,967,955	139.70	417.02
(60, .2)	min	0.1	0.1	191	3,222	11	93	0.00	94.92
	av.	2.1	34.2	17,937	1,032,758	2,271	113,849	18.98	176.91
	max	6.9	86.8	53,003	2,903,489	7,929	355,192	33.77	236.49
(60, .3)	min	17.7	95.7	126,366	3,266,648	20,842	303,587	28.15	185.00
	av.	280.5	1,001.2	2,283,561	27,927,142	381,564	2,373,907	48.45	227.79
	max	1,114.0	3,613.9	9,241,160	90,769,808	1,600,938	7,835,595	72.78	287.13
(60, .4)	min	93	366	684,846	10,994,496	96,255	570,032	37.03	194.34
	av.	3,525	-	25,861,568	121,895,404	5,112,314	8,745,809	75.02	276.04
	max	11,235	-	79,383,980	134,217,728[9]	16,585,561	12,317,275	143.55	434.79
(70, .2)	min	1.6	26.7	10,339	755,511	981	65,985	12.48	151.27
	av.	75.5	509.0	528,415	12,522,681	64,324	968,048	23.94	183.10
	max	425.6	1,071.0	2,905,466	25,501,284	376,156	2,356,260	40.53	250.89
(70, .3)	min	176	-	1,115,456	134,217,728	165,858	6,253,042	32.03	184.40
	av.	6,190	-	38,001,519	134,217,728	6,652,011	8,586,765	53.77	232.27
	max	24,106	-	136,431,618[1]	134,217,728[10]	25,270,672	11,784,946	90.00	319.44
(80, .2)	min	166	1,151	891,807	28,459,486	108,740	1,609,056	21.63	176.37
	av.	5,011	-	26,693,048	123,641,903	3,925,128	10,214,359	36.63	204.83
	max	33,986	-	174,510,384[1]	134,217,728[9]	26,728,276	13,948,144	74.29	296.38

Table 1: Comparison when starting with the 0 solution

		CPU		iter		nbSubp		%gap	
		Our	P&R	Our	P&R	Our	P&R	Our	P&R
(40, .5)	min	0.1	1.0	624	46,296	79	2,747	15.88	152.10
	av.	3.3	16.0	45,025	712,718	8,091	65,109	54.28	233.26
	max	12.1	42.3	167,520	1,900,491	30,361	214,417	105.87	348.51
(40, .6)	min	0.3	5.2	2,440	238,325	368	12,866	25.23	167.50
	av.	5.5	62.7	64,907	2,659,226	13,436	266,360	67.31	264.15
	max	17.5	290.6	190,559	12,377,285	45,191	1,365,940	113.73	369.87
(40, .7)	min	1.2	15.3	14,076	681,395	2,505	39,391	40.80	200.40
	av.	10.8	114.8	138,030	5,000,618	29,163	428,834	80.23	287.33
	max	30.6	196.3	408,656	8,657,868	89,030	745,348	120.81	385.13
(40, .8)	min	4.1	39.5	45,652	1,746,271	9,445	124,530	59.65	233.47
	av.	17.9	238.7	218,248	10,252,982	47,557	907,998	91.53	305.60
	max	47.1	637.6	570,068	28,313,814	132,575	2,466,483	140.26	423.83
(50, .4)	min	0.9	13.5	7,827	488,852	1,232	25,777	23.10	167.99
	av.	39.7	390.7	378,457	11,476,864	67,639	1,010,210	55.27	238.89
	max	109.3	1,837.6	1,069,731	41,546,929	181,653	4,225,311	103.42	349.58
(50, .5)	min	3.0	111.7	26,496	4,189,691	4,518	196,760	32.30	187.85
	av.	116.9	1,447.3	1,122,773	47,446,608	222,693	4,157,571	73.75	277.33
	max	506.6	3,882.5	4,940,892	134,217,728[1]	1,035,107	13,774,397	151.63	453.93
(50, .6)	min	2.4	153.9	21,880	4,494,473	3,156	261,175	37.28	201.81
	av.	404.4	3,181.8	3,769,637	114,660,987	804,470	8,649,012	88.41	299.38
	max	1,456.7	3,954.9	14,825,140	134,217,728[7]	3,200,303	11,989,631	139.70	417.02
(60, .2)	min	0.0	0.1	0	3,222	0	93	0.00	94.92
	av.	1.8	34.0	15,157	966,948	1,974	104,003	18.98	176.91
	max	6.1	79.6	52,248	2,378,332	7,985	276,857	33.77	236.49
(60, .3)	min	14.0	102.2	97,814	3,247,081	16,723	303,545	28.15	185.00
	av.	129.2	946.0	1,029,407	27,363,931	171,414	2,319,955	48.45	227.79
	max	322.5	2,976.7	2,581,059	85,491,073	456,492	7,341,833	72.78	287.13
(60, .4)	min	25.6	374.9	174,494	10,969,565	26,242	568,115	37.03	194.34
	av.	1,966.9	-	14,705,176	121,892,911	2,943,967	8,715,476	75.02	276.04
	max	6,838.1	-	55,390,179	134,217,728[9]	10,915,488	12,209,508	143.55	434.79
(70, .2)	min	1.5	27.6	9,668	752,448	925	65,524	12.48	151.27
	av.	29.6	511.4	182,265	12,240,103	22,073	941,269	23.94	183.10
	max	95.6	1,102.3	468,908	25,114,147	57,292	2,311,691	40.53	250.89
(70, .3)	min	171	-	1,090,284	134,217,728	164,260	6,252,966	32.03	184.40
	av.	4,780	-	28,866,598	134,217,728	5,126,085	8,524,648	53.77	232.27
	max	21,990	-	135,157,532[1]	134,217,728[10]	25,000,203	11,802,821	90.00	319.44
(80, .2)	min	66	1,377	360,033	28,438,527	44,716	1,606,638	21.63	176.37
	av.	2,574	-	13,993,004	123,639,807	2,047,042	10,071,427	36.63	204.83
	max	16,976	-	89,864,555	134,217,728[9]	13,794,692	13,237,930	74.29	296.13

Table 2: Comparison with Pardalos and Rodgers's algorithm when starting with the optimum

Pbl		CPU		iter		nbSubp		%gap	
N	d	Our	P&R	Our	P&R	Our	P&R	Our	P&R
40	0.8	9.5	126.1	114,153	5,167,735	21,879	423,545	65.56	247.64
50	0.6	119.1	2,145.0	1,229,037	76,666,401	217,003	4,455,187	63.79	250.52
60	0.4	108.9	4,546.4	805,210	134,217,728*	135,240	6,718,524	44.04	210.45
70	0.3	110.0	1,433.5	593,058	36,817,775	88,639	1,745,120	36.61	186.25
80	0.2	213.9	1,192.4	1,339,697	28,459,486	171,816	1,609,056	21,63	176.37
90	0.1	0.4	12.0	2,304	272,549	131	31,867	1.47	129.44
100	0.1	0.4	23.3	114	456,148	14	57,087	0.00	120.77

Table 3: Comparison on the difficult problems of Pardalos and Rodgers when starting with the 0 solution

Pbl		CPU		iter		nbSubp		%gap	
N	d	Our	P&R	Our	P&R	Our	P&R	Our	P&R
40	0.8	4.1	123.3	45,652	5,167,612	9,445	423,544	65.56	247.64
50	0.6	94.7	2,094.6	767,047	76,328,231	136,680	4,448,102	63.79	250.52
60	0.4	98.0	4,433.3	708,828	134,217,728*	120,429	6,721,441	44.04	210.45
70	0.3	108.7	1,397.7	626,225	36,817,760	93,345	1,745,119	36.61	186.25
80	0.2	64.0	1,154.6	360,033	28,438,527	44,716	1,606,638	21.63	176.37
90	0.1	0.3	11.2	327	264,981	12	30,528	1.47	129.44
100	0.1	0.4	17.3	0	338,168	0	37,917	0.00	120.77

Table 4: Comparison on the difficult problems of Pardalos and Rodgers when starting with the optimum

References

[1] W. E. Adams, A. Billionnet, and A. Sutter. Unconstrained $0-1$ optimization and Lagrangean relaxation. *Discrete Applied Mathematics*, 29:131–142, 1990.

[2] W. P. Adams and P. M. Dearing. On the equivalence between roof duality and Lagrangean duality for unconstrained $0-1$ quadratic programming problems. *Discrete Applied Mathematics*, 48(1):1–20, 1994.

[3] W. P. Adams and H. D. Sherali. A tight linearization and an algorithm for zero-one quadratic programming problems. *Management Science*, 32(10):1274–1290, 1986.

[4] T. M. Alkhamis, M. Hasan, and M. A. Ahmed. Simulated annealing for the unconstrained quadratic pseudo-Boolean function. *European Journal of Operational Research*, 108(3):641–652, 1998.

[5] K. Allemand, T. M. Liebling, and A. Lodi. A genetic algorithm for quadratic 0-1 programming. Technical Report OR-97-12, DEIS-Universita di Bologna, 1997.

[6] F. Barahona, M. Jünger, and G. Reinelt. Experiments in quadratic 0-1 programming. *Mathematical Programming*, 44A(2):127–137, 1989.

[7] D. P. Bertsekas. *Network optimization: continuous and discrete models.* Athena Scientific, Belmont (Massachusetts), 1998.

[8] A. Billionnet and A. Sutter. Persistency in quadratic $0-1$ optimization. *Mathematical Programming*, 54(1):115–119, 1992.

[9] A. Billionnet and A. Sutter. Minimization of a quadratic pseudo-Boolean function. *European Journal of Operational Research*, 78(1):106–115, 1994.

[10] E. Boros, Y. Crama, and P. L. Hammer. Upper-bounds for quadratic 0-1 maximization. *Operations Research Letters*, 9(2):73–79, March 1990.

[11] E. Boros and P. L. Hammer. A max-flow approach to improved roof duality in quadratic $0-1$ minimization. Technical Report 15-89, RUTCOR, Rutgers University, 1989.

[12] P. Carraresi, F. Farinaccio, and F. Malucelli. Testing optimality for quadratic 0-1 problems. *Mathematical Programming*, 85A(2):407–421, 1999.

[13] Y. Crama, P. Hansen, and B. Jaumard. The basic algorithm for pseudo-Boolean programming revisited. *Discrete Applied Mathematics*, 29(2-3):171–185, 1990.

[14] O. du Merle, P. Hansen, B. Jaumard, and N. Mladenović. An interior point algorithm for minimum sum-of-squares clustering. *SIAM Journal on Scientific Computing*, 21:1485–1505, 2000.

[15] R. Fortet. L'algèbre de Boole et ses applications en recherche opérationnelle. *Cahiers du Centre d'Études de Recherche Opérationnelle*, 4:5–36, 1959.

[16] R. Fortet. Applications de l'algèbre de Boole en recherche opérationelle. *Rev. Française Informat. Recherche Opérationnelle*, 4:17–26, 1960.

[17] F. Glover, G. A. Kochenberger, and B. Alidaee. Adaptative memory tabu search for binary quadratic programs. *Management Science*, 44:336–345, 1998.

[18] F. Glover and R. E. D. Woolsey. Further reduction of zero-one polynomial programs to zero-one linear programming problems. *Operations Research*, 21:156–161, 1973.

[19] F. Glover and R. E. D. Woolsey. Note on converting the $0-1$ polynomial programming problem to a $0-1$ linear program. *Operations Research*, 22:180–181, 1974.

[20] D. Granot and F. Granot. On solving fractional $(0-1)$ programs by implicit enumeration. *INFOR*, 14:241–249, 1976.

[21] P. L. Hammer. Boolean elements in combinatorial optimization. *Annals of Discrete Mathematics*, 4:51–71, 1979.

[22] P. L. Hammer and P. Hansen. Logical relations in quadratic $0-1$ programming. *Revue Roumaine Math. Pures et Appl.*, 26(3):421–429, 1981.

[23] P. L. Hammer, P. Hansen, and B. Simeone. Roof duality, complementation and persistency in quadratic 0-1 optimization. *Mathematical Programming*, 28(2):121–155, 1984.

[24] P. L. Hammer and S. Rudeanu. *Boolean methods in operations research and related areas*. Springer, Berlin, New-York, 1968.

[25] P. L. Hammer and B. Simeone. Quadratic functions of binary variables. In *Combinatorial optimization, Proc. 3rd Sess., Como/Italy 1986*, volume 1403 of *Lect. Notes Math.*, pages 1–56, 1989.

[26] P. Hansen. Un algorithme pour les programmes non linéaires en variables zéro-un. *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences (Paris). Séries A et B*, 270:A1700–A1702, 1970.

[27] P. Hansen. Pénalités additives pour les programmes en variables zéro-un. *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences (Paris). Séries A et B*, 273:A175–A177, 1971.

[28] P. Hansen. Methods of nonlinear 0-1 programming. *Annals of Discrete Mathematics*, 5:53–69, 1979.

[29] C. Helmberg and F. Rendl. Solving quadratic $(0,1)$-problem by semidefinite programs and cutting planes. *Mathematical Programming*, 82(3):291–315, 1998.

[30] B. Jaumard, O. Marcotte, and C. Meyer. Estimation of the quality of cellular networks using column generation techniques. Les Cahiers du GERAD G-98-02, January 1998.

[31] B. Jaumard, O. Marcotte, C. Meyer, and T. Vovor. Comparison of column generation models for channel assignment in cellular networks. Les Cahiers du GERAD G-98-49, September 1998. To appear in Discrete Applied Mathematics.

[32] B. Kalantari and A. Bagchi. An algorithm for quadratic zero-one programs. *Naval Research Logistics*, 37(4):527–538, 1990.

[33] H. Konno. Maximizing a convex quadratic function over a hypercube. *Journal of the Operations Research Society of Japan*, 23(2):171–189, 1980.

[34] A. Lodi, K. Allemand, and T. M. Liebling. An evolutionary heuristic for quadratic 0-1 programming. *European Journal of Operational Research*, 119:662–670, 1999.

[35] S. H. Lu. An improved enumerative algorithm for solving quadratic zero-one programming. *European Journal of Operational Research*, 15:110–120, 1984.

[36] S. H. Lu and A. C. Williams. Roof duality for polynomial 0-1 optimization. *Mathematical Programming*, 37(3):357–360, 1987.

[37] P. Michelon and N. Maculan. Lagrangean methods for 0-1 quadratic problems. *Discrete Applied Mathematics*, 42(2-3):257–269, 1993.

[38] G. Palubeckis. Quadratic 0-1 optimization. *Lithuanian Academy of Sciences. Informatica*, 1(1):89–106, 188, 199, 1990.

[39] G. Palubeckis. A heuristic-based branch and bound algorithm for unconstrained quadratic zero-one programming. *Computing*, 54(4):283–301, 1995.

[40] P. M. Pardalos and G. P. Rodgers. Computational aspects of a branch and bound algorithm for quadratic zero-one programming. *Computing*, 45(2):131–144, 1990.

[41] N. V. Thoai. Global optimization techniques for solving the general quadratic integer programming problem. *Computational Optimization and Applications*, 10(2):149–163, 1998.

[42] X. Zhou, Y. Wang, X. Tian, and R. Guo. A tabu search algorithm for quadratic 0-1 programming problem. *Chin. Q. J. Math.*, 12(4):98–102, 1997.