Using a Mixed Integer Quadratic Programming Solver for the Unconstrained Quadratic 0-1 Problem

Alain Billionnet • Sourour Elloumi[1]

Laboratoire CEDRIC, Institut d'Informatique d'Entreprise, 18 allée Jean Rostand, F-91025 Evry

Laboratoire CEDRIC, Conservatoire National des Arts et Métiers, 292 rue Saint Martin, F-75141 Paris

billionnet@iie.cnam.fr• elloumi@cnam.fr

Abstract

In this paper, we consider problem (P) of minimizing a quadratic function $q(x) = x^tQx + c^tx$ of binary variables. Our main idea is to use the recent Mixed Integer Quadratic Programming (MIQP) solvers. But, for this, we have to first convexify the objective function $q(x)$. A classical trick is to raise up the diagonal entries of Q by a vector u until $(Q+diag(u))$ is positive semidefinite. Then, using the fact that $x_i^2 = x_i$, we can obtain an equivalent convex objective function, which can then be handled by an MIQP solver. Hence, computing a suitable vector u constitutes a preprocessing phase in this exact solution method. We devise two different preprocessing methods. The first one is straightforward and consists in computing the smallest eigenvalue of Q. In the second method, vector u is obtained once a classical SDP relaxation of (P) is solved.

We carry out computational tests using the generator of (Pardalos and Rodgers, 1990) and we compare our two solution methods to several other exact solution methods.

Keywords: Integer Programming, Quadratic 0-1 optimization, Convex Quadratic Relaxation, Semidefinite Positive Relaxation, Experiments

[1]corresponding author

1 Introduction

Consider the quadratic function

$$q(x) = x^t Q x + c^t x$$

and the 0-1 quadratic problem

$$(P): \min \{q(x) : x \in \{0,1\}^n\} \tag{1}$$

where Q is an $n \times n$ real matrix, and $c \in \mathbb{R}^n$. Without loss of generality, we can suppose that Q is symmetric and also that the diagonal terms of Q are equal to 0. If this is not the case, Q can be converted to the symmetric form $(Q + Q^t)/2$ and the linear terms $q_{ii}x_i$ can be substituted for the diagonal terms $q_{ii}x_i^2$ because $x_i^2 = x_i$ for $x_i \in \{0,1\}$. Problem (P) is NP-hard and has numerous applications. Consult, for example, (Hansen et al. 2000) and (Boros and Hammer 2001). Furthermore, a recent application of (P) in the medical field is reported in (Iasemidis et al. 2001).

Various approaches have been used to solve (P) exactly. Two recent overviews of these approaches are presented in (Hansen et al. 2000) and (Helmberg and Rendl 1998). The decomposition method described in (Chardaire and Sutter 1995) is probably the most efficient approach in terms of the size of the instances it can solve (100 variables, all densities). One of the possible techniques is to relax (P) to a tractable nonlinear continuous problem in order to obtain lower bounds. As mentioned in (Poljak and Wolkowicz 1995), this approach was used for different Combinatorial Optimization problems. In addition to the references cited by Poljak and Wolkowicz, we have the following ones, which concern unconstrained quadratic 0-1 programming: McBride and Yormak (1980) and Helmberg and Rendl (1998).

For any vector $u \in \mathbb{R}^n$, let us define the perturbed function $q_u(x)$ in the following way

$$q_u(x) = x^t(Q - diag(u))x + (c + u)^t x \tag{2}$$

where $diag(u)$ is the diagonal matrix obtained from vector u. One can easily observe that $q_u(x)$ can also be written as $q(x) + \sum_{i=1}^n u_i(x_i - x_i^2)$, and that $q_u(x) = q(x)$ for all $x \in \{0,1\}^n$. Therefore, one can solve problem (P) by solving the equivalent problem

$$(P_u) : \min \{q_u(x) : x \in \{0,1\}^n\} \tag{3}$$

Moreover, we define the lower bound $\beta(u)$ of the optimal value of problem (P) as the optimal value obtained by the continuous relaxation of problem (P_u)

$$\beta(u) = \min \{q_u(x) \ : x \in [0,1]^n\} \tag{4}$$

In this paper, we will focus on vectors u such that matrix $(Q - diag(u))$ is semidefinite positive ($(Q - diag(u)) \succeq 0$). In this case, computing the lower bound $\beta(u)$ amounts to solving a convex quadratic problem, which can be done in polynomial time (see Kozlov et al. 1979). Our motivation comes from the fact that the new versions of optimization software are now able to solve quadratic integer programs, that have a convex quadratic objective function and a set of linear constraints. This new tool is called an MIQP solver. Hence, whenever $(Q - diag(u)) \succeq 0$, these new versions can handle problem (P_u) directly. This gives us a family of exact solution methods for problem (P), which differ by the preprocessing phase, i.e. the method used to find an appropriate vector u.

In Section 2, we present a simple way to find a first vector u with $(Q - diag(u)) \succeq 0$, based on the computation of the smallest eigenvalue of matrix Q. This constitutes a first exact solution method for problem (P).

In Section 3, we consider the problem of finding the best vector u^*, i.e. that maximizes $\beta(u)$ under the constraint $(Q - diag(u)) \succeq 0$. In addition, we use the known relationship between $\beta(u^*)$ and semidefinite programming. This relationship provides us with a practical method for computing u^*, based on the resolution of an SDP relaxation of problem (P). This constitutes a second exact solution method for problem (P), through the resolution of problem (P_{u^*}). SDP relaxations are known to provide tighter bounds than quadratic convex relaxations but more effort is required to compute them. In our second method, the strength of an SDP relaxation of (P) is captured in a quadratic convex relaxation of (P).

Section 4 reports the computational results together. Then, comparisons with existing methods are reported in Section 5. Section 6 is a conclusion.

2 A first choice of the perturbed function q_u

Let $\lambda_{min} \in \mathbb{R}$ be the smallest eigenvalue of matrix Q. If at least one term of Q is nonzero then λ_{min} is a real negative number. Consider the perturbed function q_u where $u = \lambda_{min} e$ and e is the vector of all ones

$$q_{\lambda_{min} e}(x) = x^t(Q - diag(\lambda_{min} e))x + (c + \lambda_{min} e)^t x$$

Matrix $(Q - diag(\lambda_{min}e))$ is positive semidefinite and then function $q_{\lambda_{min}e}$ is convex. So, problem $(P_{\lambda_{min}e})$ can be solved by an MIQP solver. Recall that problems $(P_{\lambda_{min}e})$ and (P) are equivalent in the sense that $\forall\ x \in \{0,1\}^n$, $q_{\lambda_{min}e}(x) = q(x)$.

The idea of transforming (P) into $(P_{\lambda_{min}e})$ has already been considered by Hammer and Rubin (1970). However, to the authors' knowledge, no computational results associated to this approach have been reported in the literature.

Example 1 *Consider the example where* $Q = \begin{pmatrix} 0 & 26 & 44 & -73 \\ 26 & 0 & -45 & 11 \\ 44 & -45 & 0 & 84 \\ -73 & 11 & 84 & 0 \end{pmatrix}$ *and* $c = \begin{pmatrix} -119 \\ 27 \\ -187 \\ -2 \end{pmatrix}$. *The smallest eigenvalue of matrix* Q *is* $\lambda_{min} = -149.8$ *and the bound* $\beta(\lambda_{min}e)$ *is equal to* -302.25. *The optimal value is -267.*

3 An optimal perturbed function q_u^*

We have seen in the introduction that any vector u satisfying $(Q - diag(u)) \succeq 0$ gives a new problem (P_u), defined by (3), and equivalent to our initial problem (P), defined by (1). We also recall the definition $\beta(u) = \min\{q_u(x)\ : x \in [0,1]^n\}$ already given by (4).

As mentioned above, any problem (P_u) can be submitted for resolution by the branch-and-bound algorithm of the MIQP solver. But, it is well known that the behavior of a branch-and-bound algorithm is very dependent upon the bound at the root of the search tree. Hence, we would say that formulation (P_{u_1}) is better than formulation (P_{u_2}) if its continuous relaxation amounts to a better bound or, in other words, if $\beta(u_1)$ is larger than $\beta(u_2)$. Furthermore, one can make an "optimal" choice by computing a vector u^* that maximizes $\beta(u)$, under the constraint $(Q - diag(u)) \succeq 0$. More precisely, let us define u^* and the associated lower bound β^* as

$$\beta^* = \beta(u^*) = \max_{\substack{u \in \mathbb{R}^n \\ (Q - diag(u)) \succeq 0}} \beta(u) \tag{5}$$

It is already known that β^* can be computed as the optimal value of a semidefinite programming relaxation of problem (P). This result appears, for example, in (Poljak, Rendl, and Wolkowicz 1995), but we can consider it was already proved in (Körner and Richter 1982), (Shor 1987), and (Körner 1988). Besides, (Lemaréchal and Oustry 1999) give an interpretation of this result through lagrangian duality. In order to recall this result inside our context and notations, we state the following proposition.

Proposition 1 *Bound β^* is equal to the optimal value of the semidefinite program:*

$$\begin{array}{lll} (SDP) & : max & r \\ & s.t.: & \\ & & \left[\begin{array}{cc} -r & \frac{1}{2}(c+u)^t \\ \frac{1}{2}(c+u) & Q - diag(u) \end{array} \right] \succeq 0 \\ & & r \in \mathbb{R} \qquad u \in \mathbb{R}^n \end{array}$$

and is also equal to the optimal value of its dual:

$$\begin{array}{lll} (DSDP) & : min & c^t x + \sum_{i=1}^n \sum_{j=1}^n Q_{ij} X_{ij} \\ & s.t.: & X_{ii} = x_i \qquad\qquad i = 1, ..., n \\ & & \left[\begin{array}{cc} 1 & x^t \\ x & X \end{array} \right] \succeq 0 \\ & & x \in \mathbb{R}^n \qquad X \in \mathbb{R}^{n \times n} \end{array}$$

Moreover, if (r^*, u^*) is an optimal solution of (SDP), then $(Q - diag(u^*)) \succeq 0$ holds and $\beta(u^*) = \beta^*$. ■

Proposition 1 gives a practical method for computing the best vector u^* and the lower bound β^*, based on the resolution of a semidefinite program. Let us consider again the instance of Example 1 to show that β^* can be strictly better than $\beta(\lambda_{min} e)$. The optimal vector one obtains by solving the SDP is $u^* = (-174.85, -74.62, -177.62, -145.71)$, and the corresponding lower bound is $\beta^* = \beta(u^*) = -290.50$, which is indeed better than $\beta(\lambda_{min} e) = -302.25$. Recall that, in this example, the optimal value is equal to -267.

The SDP relaxation we consider is not new. It has already been considered by several authors. It was shown in (Goemans and Williamson 1994) that, for the max-cut problem with non-negative weights -a particular case of problem (P)-, this SDP relaxation leads to a bound with an error of at most 13.8%. Furthermore, in (Helmberg and Rendl 1998), this SDP relaxation has been both used in a cutting-plane approach and embedded in a branch-and-bound framework. We provide an alternative use of this SDP relaxation, as a preprocessing phase of an MIQP resolution.

4 Computational results

Considered Instances

We use the Sparse Problem Generator of (Pardalos and Rodgers 1990) which we implemented in $C++$. We restrict our parameters to the following profile, already used by several authors:

- the linear (or diagonal) coefficients c_i are in the range [-100, 100],
- the off-diagonal coefficients of the symmetric matrix Q are in the range [-50, 50], and,
- matrix Q has a density equal to d, a fraction of 1. This is a parameter of the generator. Density is here meant to be the probability that a nonzero will occur for any off-diagonal entry.

For any considered pair (n, d) where n is the number of variables and d, the density of matrix Q, we generate 10 instances with seeds 1, 2, ..., 10. Then, we choose 15 pairs (n, d), 6 of which have been used by (Pardalos and Rodgers 1990) in their Table 5.4 which was intended to show the limits of their method. These instances have also been used in (Hansen et al. 2000). The remaining 9 pairs are intended to show the performances and limits of our new solution methods, as shown in Tables 1 and 4. For several instances, we could compute the optimal value by our solution methods. But, for the large-sized instances, we only have the value of the best integer solution computed by our second method through the MIQP solver, after one hour of CPU time.

Machine and software

Our experiments are carried out on a PC with a Pentium III of 1000 Mhz and 1024 MB of RAM.

Implementing our first exact solution method described in Section 2 is particularly easy. The preprocessing phase consists of computing the smallest eigenvalue λ_{min}. We do this using Scilab (Gomez 1999).

For our second exact solution method described in Section 3, the preprocessing phase consists of computing the best vector u^* as part of an optimal dual solution (r^*, u^*) for problem $(DSDP)$. In order to solve $(DSDP)$, we use SDP_S, the modeling language for semidefinite programming designed by Delaporte et al. (2002). SDP_S works with SBmethod, an SDP solver that implements the spectral bundle method of Helmberg and Rendl (2000).

Then, for our two solution methods, the preprocessing phase is followed by the exact solution phase. For this, we use the modeling language AMPL (Fourer et al. 1994) together with the MIQP solver of CPLEX 8.1 (ILOG 2002). Observe that only the formulation $(P_{\lambda_{min}e})$ or (P_{u^*}) is provided to the solver. For example, we do not provide a starting feasible solution although this could be done easily.

Computational results concerning the solution of P through the solution of $(P_{\lambda_{min}e})$

In Table 1, column 1 gives the size n. Column 2 indicates the density of matrix Q. Column 3 gives the average CPU time in seconds for preprocessing, i.e. for computing the smallest eigenvalue λ_{min}. Column 4 gives the average gap at the root, which is also the error associated to lower bound $\beta(\lambda_{min}e)$, i.e. $\frac{\beta(\lambda_{min}e)-opt}{opt}$ where opt is either the optimal value or the value of the best known integer solution obtained as described above. Column 5 gives the number of instances (over 10) for which the MIQP solver was able to solve the quadratic MIP problem within one hour of CPU time. Furthermore, for the instances solved within one hour, Column 6 (resp. 8) reports the average CPU time needed (resp. number of nodes explored) to prove optimality. For the instances which could not be solved, Column 7 gives the average final gap obtained by the MIQP after one hour. For example, for the $n = 80$ and $d = 0.2$ row, 8 instances over 10 could be solved to optimality within an average time of 578.2 seconds and after 629145 nodes have been explored. For the 2 unsolved instances, the final gap is equal to 1% in average.

For all the instances of Table 1, the preprocessing phase is very fast, and the gap at the root is rather independent of the size and of the density. These results are already satisfactory if compared to literature and we will show in the following that they are much improved by the results of our second solution method.

Table 1: Results through the solution of $(P_{\lambda_{min}e})$

		Preproc	MIQP solver				
n	density	CPU1	Gap (%)	solved (1h)	CPU2	final Gap (%)	nodes
*50	0.4	0.1	15.3	10	10.4	0	17865
*50	0.6	0.2	13.7	10	8.5	0	13007
50	1.0	0.2	14.5	10	14.3	0	16952
*60	0.2	0.1	16.6	10	96.6	0	185911
*60	0.4	0.2	15.4	10	91.6	0	111388
*70	0.3	0.2	13.8	9	131.4	0.4	176058
70	0.8	0.3	15.1	10	299.3	0	224661
*80	0.2	0.2	14.5	8	578.2	1.0	629145
100	1.0	0.7	15.3	0	-	4.1	-
120	0.3	0.4	15.8	0	-	6.3	-
120	0.8	0.8	16.2	0	-	7.4	-
150	0.3	0.6	16.7	0	-	9.9	-
150	0.8	1.2	16.2	0	-	9.9	-
200	0.3	1.0	16.8	0	-	12.4	-
200	0.8	2.4	15.5	0	-	11.4	-

* : these instances have already been considered in (Pardalos and Rodgers 1990)
- : no instance could be solved to optimality within 1h

Computational results concerning the solution of P through the solution of (P_{u^*}) Here, we consider again the same instances. We apply the two-phase solution method described in Section 3. Table 4 reports the results for the instances of Table 1, and nearly the same meaning is associated to each column. When compared to the results of Table 1, we can observe that the gap at the root of the branch-and-bound tree is approximatively divided by 2. This mainly explains the fact that in Table 4, for $n \leq 120$, 98 instances over 110 are solved within one hour, i.e., our improved method could solve 21 instances that the first method could not solve within one hour. Moreover, the new method is much faster. For example, for the the instances with n=60, and density=0.4, the 10 instances were solved by both of the methods. The new method took 2.2+3.7=5.9 seconds while the first method took 0.2+91.6=91.8 in average, i.e. the new method is about 15 times faster for those instances.

We can also observe that, as expected, the preprocessing by solving an SDP takes quite a long time. For the (n=100, density=1.0) instances, the computation of the smallest eigenvalue by Scilab took about 0.7 seconds in average, versus 10.3 seconds for solving the SDP. Hence, for those instances, we spent about 10 seconds of additional time to move the gap on

the relaxation at the root from 15.3% to 7.6%, and this is globally very profitable. Let us point out that, although only two of the (n=120, density=0.8) instances are solved within one hour of CPU time, the gap between the best node and the best known solution after one hour is less than 3% for the 8 remaining instances.

Table 2: Results through the solution of (P_{u^*})

		Preproc	MIQP solver				
n	density	CPU1	Gap (%)	solved (1h)	CPU2	final Gap (%)	nodes
*50	0.4	1.7	6.1	10	0.6	0	916
*50	0.6	1.9	5.8	10	0.8	0	1104
50	1.0	1.8	6.6	10	1.3	0	1333
*60	0.2	4.0	5.5	10	1.1	0	1700
*60	0.4	2.2	9.0	10	3.7	0	4542
*70	0.3	3.1	6.1	10	7.4	0	9093
70	0.8	4.5	7.3	10	15.1	0	23044
*80	0.2	8.1	5.9	10	13.8	0	15771
100	1.0	10.3	7.6	9	725.0	0.9	212414
120	0.3	13.2	7.1	7	1472.6	1.2	757873
120	0.8	14.1	8.7	2	2374.3	2.5	579616
150	0.3	29.3	8.4	0	-	3.6	-
150	0.8	29.4	8.9	0	-	4.4	-
200	0.3	61.9	9.0	0	-	5.9	-
200	0.8	75.0	8.7	0	-	6.0	-

* : these instances have already been considered in (Pardalos and Rodgers 1990)
- : no instance could be solved to optimality within 1h

5 Comparison with other methods

Comparison with the solution of (P) by linearization

Consider problem (LP) which is equivalent to (P):

$$
\begin{array}{lll}
(LP): \min & \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2 * Q_{ij} y_{ij} & \\
\text{s.t.:} & y_{ij} \leq x_i & i < j,\ Q_{ij} < 0 \\
& y_{ij} \leq x_j & i < j,\ Q_{ij} < 0 \\
& y_{ij} \geq x_i + x_j - 1 & i < j,\ Q_{ij} > 0 \\
& 0 \leq y_{ij} \leq 1 & i < j,\ Q_{ij} \neq 0 \\
& x_i \in \{0, 1\} &
\end{array}
$$

Table 3: Solution of (P) through (LP) and a MILP solver

n	density	Gap (%)	solved (1h)	CPU	final Gap (%)	nodes
*50	0.4	55.3	10	14.1	0	2297
*50	0.6	88.4	10	313	0	34781
50	1.0	146.2	0	-	32.7	-
*60	0.2	19.0	10	1.2	0	86
*60	0.4	75.0	10	397.2	0	49104
*70	0.3	54.2	10	209.3	0	21666
70	0.8	169.5	0	-	116.2	-
*80	0.2	36.6	10	55.0	0	6031
100	1.0	266.6	0	-	264.2	-
120	0.3	112.2	0	-	85.2	-
120	0.8	267.8	0	-	278.4	-
150	0.3	145.7	0	-	139.2	-
150	0.8	302.4	0	-	330.6	-
200	0.3	186.1	0	-	202.8	-
200	0.8	354.7	0	-	432.4	-

* : these instances have already been considered in (Pardalos and Rodgers 1990)
- : no instance could be solved to optimality within 1h

As already observed by other authors, the integrality gap associated to problem (LP) and reported in Column 3 of Table 3 is large for this class of instances. Moreover, when n is large or Q is not sparse, only a few nodes can be explored within one hour. For example, with $n = 200$ and density 0.8, the average number of branch-and bound nodes explored by the linearization method is equal to 660, while it is equal to 324000 by our second solution method. Let us also observe that the final gap obtained by the MILP resolution after one hour, and reported in Column 6 of Table 3 can be larger than the integrality gap, reported in Column 3. This is due to the fact that the integrality gap is computed by using the best known solutions while Column 6 reports the actual gaps obtained by the MILP solver.

Comparison to enumerative methods

Table 4 presents a comparison between the method of Pardalos and Rodgers (1990), the one of Hansen et al. (2000) and ours. Each line of this table corresponds to average results for 10 instances. These instances are exactly the same as the instances pointed out by $*$ in Tables 1-3. Hansen et al. run their program and the one of Pardalos and Rodgers on a Sun Sparc SS20/514 MP with 128 MB of RAM, with 0 as initial solution.

Table 4: Comparison between Pardalos and Rodgers (PR), Hansen et al. (HJM), and our approach

		Gap (%)			CPU (sec.)		
n	density	PR (1990)	HJM (2000)	our	PR (1990)	HJM (2000)	our
50	0.4	238	55	6.1	317.8	46.4	2.3
50	0.6	299	88	5.8	-	636.6	2.7
60	0.2	176	18	5.5	34.2	2.1	5.1
60	0.4	276	75	9.0	-	3525	5.9
70	0.3	232	53	6.1	-	6190	10.5
80	0.2	204	36	5.9	-	5011	21.9

We observe that, except for instances with 60 variables and density 0.2, our algorithm is, in average, faster by a factor 20 to about 600 compared to the one of Hansen et al. We also can observe that, for all the considered instances, our method gives an average gap equal to about 7%, i.e. smaller by a factor 8 compared to the one of Hansen et al.

Comparison to Chardaire and Sutter's decomposition method

Table 5 presents a comparison between the method in (Chardaire and Sutter, 1995) and our solution method described in Section 3. Each row in this table corresponds to average results for 10 instances. We generated these instances as described above, i.e. in a similar way to Chardaire and Sutter, but we are not sure to be considering exactly the same instances as them. Moreover, Column 5 of Table 5 reports the CPU times obtained by Chardaire and Sutter on a HP 720 workstation. An important observation one can do is that the gap obtained by Chardaire and Sutter at the root of their branch-and-bound is much smaller than our. We can remark that their method is very specific to problem (P) and implements a variety of sophisticated algorithms. Our method uses only two general-purpose solvers and is easy to implement. Moreover, our method can be easily adapted to extensions of problem (P), for example by adding linear constraints.

Table 5: Comparison to Chardaire and Sutter's decomposition method

		Gap (%)		CPU (sec.)	
n	density	CS(1995)	our	CS(1995)	our
40	0.1	0	2.6	15.6	0.7
40	0.5	0	5.7	6.2	1.4
40	1.0	0	6.5	18.7	1.1
50	0.1	0	2.7	41.2	2.9
50	0.5	0	5.4	23.1	2.5
50	1.0	0	6.6	62.0	3.1
75	0.1	1.0	4.4	208.2	8.7
75	0.5	0.2	6.9	491.8	22.4
75	1.0	0.5	8.2	766.5	66.9
100	0.1	2.6	4.9	764.8	45.8
100	0.5	2.4	7.1	1902.5	$436.7^{(1)}$
100	1.0	2.7	7.6	2890.4	$735.3^{(1)}$

$^{(1)}$: One instance over ten could not be solved within one hour of CPU time

6 Conclusion

In this paper, we have proposed two methods to state the problem of minimizing a quadratic 0-1 function $q(x)$ as the problem of minimizing an equivalent 0-1 function $q'(x)$ whose continuous relaxation is convex. The first method is fast and easy to implement since it only requires eigenvalues computation. The second one, which requires the solution of a positive semidefinite program, takes more computation time and is more sophisticated. The experiments show that the relative gap between the optimum of the continuous relaxation of $q'(x)$ and the optimum of $q(x)$ is about twice smaller in the second method than in the first one. In both approaches, the computation of the optimum of the 0-1 function $q'(x)$ (and therefore the optimum of $q(x)$) is entirely carried out by the general-purpose solver CPLEX. At each node of the search tree, the computation of the lower bound consists in minimizing a convex quadratic function over $[0,1]^n$, which is particularly fast. The experiments show that these two methods allow to minimize quadratic 0-1 functions efficiently in terms of computation time and the size of the problems considered, in comparison to existing methods. Finally, the approaches that we propose can be easily generalized to linearly constrained quadratic 0-1 optimization problems.

References

[1] E. Boros and P. L. Hammer. Pseudo-boolean optimization. *Discrete Applied Mathematics*, 123:155–225, 2002.

[2] P. Chardaire and A. Sutter. A decomposition method for quadratic zero-one programming. *Management Science*, 41(4):704–712, 1995.

[3] G. Delaporte, S. Jouteau, and F. Roupin. Sdp_s : A tool to formulate and solve semidefinite relaxations for bivalent quadratic problems, 2002. http://semidef.free.fr/.

[4] C. Gomez (Ed.). *Engineering and Scientific Computing With Scilab*. Springer Verlag, 1999.

[5] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. The Scientific Press (now an imprint of Boyd & Fraser Publishing Co.), Danvers, MA, USA, 1993.

[6] M. X. Goemans and D. P. Williamson. .878-approximation for max cut and max 2sat. *in Proc. 26 th ACM Symp. Theor. Computing*, pages 422–431, 1994.

[7] P. L. Hammer and A. A. Rubin. Some remarks on quadratic programming with 0–1 variables. *Revue Francaise d'Informatique et de Recherche Operationnelle*, 4(3):67–79, 1970.

[8] P. Hansen, B. Jaumard, and C. Meyer. A simple enumerative algorithm for unconstrained 0-1 quadratic programming. Technical Report G-2000-59, Les Cahiers du GERAD, 2000.

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

[10] C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization*, 10(3):673–696, 2000.

[11] L. D. Iasemidis, P. M. Pardalos, J. C. Sackellares, and D.-S. Shiau. Quadratic binary programming and dynamical system approach to determine the predictability of epileptic seizures. *Journal of Combinatorial Optimization*, 5(1):9–26, 2001.

[12] ILOG. *ILOG CPLEX 8.0 Reference Manual*. ILOG CPLEX Division, Gentilly, France, 2002.

[13] F. Körner. A tight bound for the Boolean quadratic optimization problem and its use in a branch and bound algorithm. *Optimization*, 19(5):711–721, 1988.

[14] F. Körner and C. Richter. Zur effektiven Lösung von Booleschen, quadratischen Optimierungsproblemen. *Numerische Mathematik*, 40:99–109, 1982.

[15] M. K. Kozlov, S. P. Tarasov, and L. G. Khachiyan. Polynomial solvability of convex quadratic programming. *Doklady Akademii Nauk SSSR*, 248(5):1049–1051, 1979. See also, *Soviet Mathematics Doklady* volume 20, pages 1108–1111, 1979.

[16] C. Lemaréchal and F. Oustry. Semidefinite relaxations and lagrangian duality with application to combinatorial optimization. Technical Report RR-3710, INRIA Rhône-Alpes, 1999.

[17] R. McBride and J. Yormark. An implicit enumeration algorithm for quadratic integer programming. *Management Science*, 26:282–296, 1980.

[18] P.M. Pardalos and G.P. Rodgers. Computational aspects of a branch and bound algorithm for quadratic 0-1 programming. *Computing*, 45:131–144, 1990.

[19] S. Poljak, F. Rendl, and H. Wolkowicz. A recipe for semidefinite relaxation for (0,1)-quadratic programming. *Journal of Global Optimization*, 7:51–73, 1995.

[20] S. Poljak and H. Wolkowicz. Convex relaxations of $(0, 1)$-quadratic programming. *Mathematics of Operations Reseach*, 20:550–561, 1995.

[21] N.Z. Shor. Class of global minimum bounds of polynomial functions. *Cybernetics*, 23(6):731–734, 1987.