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

Alain Billionnet, Sourour Elloumi

# 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. Furthermore, we report computational results for the max-cut problem.

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

[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. Three recent overviews of these approaches are presented in (Beasley 1998), (Helmberg and Rendl 1998), and (Hansen et al. 2000). 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). More recently, second order cone programming has been used for solving $(P)$. See (Kim and Kojima 2001) and (Muramatsu and Suzuki 2003).

Beside these exact methods, several heuristic approaches have been proposed for solving large instances of (P). The reader can find a brief review and many references on the subject in (Merz and Katayama (to appear)).

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 positive semidefinite ($(Q - diag(u)) \succeq 0$). 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 $(P)$.

Sections 4 and 5 report the computational results together with comparisons with existing methods. 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 $\lambda_{min}$ 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 $\beta^*$ 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 $\beta^*$ 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 2001) 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 $\beta^*$ is equal to the optimal value of the semidefinite program:*

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

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

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

*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 $\beta^*$, based on the resolution of a semidefinite program. Let us consider again the instance of Example 1 to show that $\beta^*$ 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 for the Unconstrained Quadratic 0-1 Problem

We will first describe the machine and software we use all along the computational results in the current section and in Section 5 . Then we consider a large set of instances, randomly generated by the generator in (Pardalos and Rodgers 1990). We apply the two algorithms of Sections 2 and 3 to these instances and we compare our results to a general linearization method and to methods in the literature. Finally, we provide computational results for a set of 45 benchmak instances introduced in (Glover and al. 1998) and available in OR-Library (Beasley 1990).

**Machine and software**

Our experiments are carried out on a portable PC with a Pentium IV of 1600 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 $\lambda_{min}$. 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.

## 4.1 Results and comparisons for randomly generated 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 2. 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 three hours of CPU time.

**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 $\lambda_{min}$. 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 three hours of CPU time. For the instances solved within three hours, Columns 6, 7, and 8 (resp. 10) report the min, average, and max CPU time needed (resp. number of nodes explored) by the branch-and-bound algorithm to prove optimality. For the instances which could not be solved, the MIQP solver stops with a final gap, i.e. $\frac{bb-opt}{opt}$ where $bb$ is the lower bound computed by the branch-and-bound algorithm after three hours. Column 9 gives this average final gap. For example, for the $n = 100$ and $d = 1$ row, 4 instances over 10 could be solved to optimality within an average time of 4080 seconds and after 3736585 nodes have been explored. For the 6 unsolved instances, the final gap is equal to 2.6% 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 (3h) | | CPU2 | | final Gap (%) | nodes |
| | | av. | av. | | min | av. | max | av. | av. |
| *50 | 0.4 | 0.1 | 15.3 | 10 | 0.3 | 4.3 | 16.8 | 0 | 17865 |
| *50 | 0.6 | 0.2 | 13.7 | 10 | 0.1 | 3.3 | 10 | 0 | 13007 |
| 50 | 1.0 | 0.2 | 14.5 | 10 | 0.2 | 5.2 | 19.4 | 0 | 16952 |
| *60 | 0.2 | 0.1 | 16.6 | 10 | 0.1 | 45.4 | 228 | 0 | 185911 |
| *60 | 0.4 | 0.2 | 15.4 | 10 | 1.1 | 35.7 | 247 | 0 | 111388 |
| *70 | 0.3 | 0.2 | 13.8 | 10 | 0.9 | 209 | 1580 | 0 | 455052 |
| 70 | 0.8 | 0.3 | 15.1 | 10 | 12.6 | 104 | 208.5 | 0 | 224661 |
| *80 | 0.2 | 0.2 | 14.5 | 10 | 72 | 681 | 3447 | 0 | 1298802 |
| 100 | 1.0 | 0.7 | 15.3 | 4 | 2661 | 4080 | 7412 | 2.6 | 3736585 |
| 120 | 0.3 | 0.4 | 15.8 | 0 | | - | | 4.4 | - |
| 120 | 0.8 | 0.8 | 16.2 | 0 | | - | | 5.2 | - |
| 150 | 0.3 | 0.6 | 16.7 | 0 | | - | | NC | - |
| 150 | 0.8 | 1.2 | 16.2 | 0 | | - | | NC | - |
| 200 | 0.3 | 1.0 | 16.8 | 0 | | - | | NC | - |
| 200 | 0.8 | 2.4 | 15.5 | 0 | | - | | NC | - |

* : these instances have already been considered in (Pardalos and Rodgers 1990)
- : no instance could be solved to optimality within 3h

**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 2 reports the results for the instances of Table 1, and 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 2, 107 instances over 150 are solved within three hours, i.e., our improved method could solve 23 instances that the first method could not solve within three hours. Moreover, the new method is much faster. For example, for the instances with $n$=70, and density=0.8, the 10 instances were solved by both of the methods. The new method took 1.7+5.4=7.1 seconds while the first method took 0.3+104=104.3 seconds 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 eigen-

value by Scilab took about 0.7 seconds in average, versus 4.5 seconds for solving the SDP. Hence, for those instances, we spent about 4 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 6 of the ($n$=120, density=0.8) instances are solved within three hours of CPU time, the gap between the best node and the best known solution after three hours is about 1% for the 4 remaining instances.

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

| | | Preproc | MIQP solver | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $n$ | density | CPU1 | Gap (%) | solved (3h) | CPU2 | | | final Gap (%) | nodes |
| | | av. | av. | | min | av. | max | av. | av. |
| *50 | 0.4 | 1 | 6.1 | 10 | 0.1 | 0.3 | 0.8 | 0 | 910 |
| *50 | 0.6 | 1 | 5.8 | 10 | 0.1 | 0.3 | 1 | 0 | 1034 |
| 50 | 1.0 | 0.7 | 6.6 | 10 | 0.1 | 0.5 | 1.7 | 0 | 1367 |
| *60 | 0.2 | 2.4 | 5.5 | 10 | 0.1 | 0.5 | 1.5 | 0 | 1768 |
| *60 | 0.4 | 1.2 | 9.0 | 10 | 0.2 | 1.4 | 7.7 | 0 | 4366 |
| *70 | 0.3 | 1.7 | 6.1 | 10 | 0.1 | 2.8 | 11 | 0 | 8841 |
| 70 | 0.8 | 1.7 | 7.3 | 10 | 1 | 5.4 | 10.6 | 0 | 10899 |
| *80 | 0.2 | 4.7 | 5.9 | 10 | 0.6 | 5.4 | 28.7 | 0 | 16046 |
| 100 | 1.0 | 4.5 | 7.6 | 10 | 27.4 | 372.7 | 1671 | 0 | 370358 |
| 120 | 0.3 | 5.5 | 7.1 | 10 | 168 | 1263.6 | 4667 | 0 | 1405507 |
| 120 | 0.8 | 6.8 | 8.7 | 6 | 322 | 3909.5 | 9898 | 1.3 | 2703558 |
| 150 | 0.3 | 9.8 | 8.4 | 1 | | 6789 | | 2.5 | 3964619 |
| 150 | 0.8 | 12.2 | 8.8 | 0 | | - | | 2.9 | - |
| 200 | 0.3 | 17.7 | 8.9 | 0 | | - | | 4.9 | - |
| 200 | 0.8 | 22.5 | 8.5 | 0 | | - | | 4.8 | - |

* : these instances have already been considered in (Pardalos and Rodgers 1990)
- : no instance could be solved to optimality within 3h

**Comparison with the solution of $(P)$ by linearization**

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

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

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

| $n$ | density | Gap (%) | solved (3h) | CPU | | | final Gap (%) | nodes |
|---|---|---|---|---|---|---|---|---|
| | | av. | | min | av. | max | av. | av. |
| *50 | 0.4 | 55.3 | 10 | 0.7 | 4.4 | 16.7 | 0 | 2297 |
| *50 | 0.6 | 88.4 | 10 | 1.8 | 90.1 | 250 | 0 | 34781 |
| 50 | 1.0 | 146.2 | 7 | 1130 | 3420 | 5555 | 14.3 | 393864 |
| *60 | 0.2 | 19.0 | 10 | 0.1 | 0.4 | 0.8 | 0 | 86 |
| *60 | 0.4 | 75.0 | 10 | 22 | 122.8 | 581 | 0 | 49104 |
| *70 | 0.3 | 54.2 | 10 | 6 | 60 | 208 | 0 | 21666 |
| 70 | 0.8 | 169.5 | 0 | | - | | 60.2 | - |
| *80 | 0.2 | 36.6 | 10 | 1.6 | 16.3 | 88 | 0 | 6031 |
| 100 | 1.0 | 266.6 | 0 | | - | | 187.3 | - |
| 120 | 0.3 | 112.2 | 0 | | - | | NC | - |
| 120 | 0.8 | 267.8 | 0 | | - | | NC | - |
| 150 | 0.3 | 145.6 | 0 | | - | | NC | - |
| 150 | 0.8 | 302.4 | 0 | | - | | NC | - |
| 200 | 0.3 | 186.1 | 0 | | - | | NC | - |
| 200 | 0.8 | 354.7 | 0 | | - | | NC | - |

* : these instances have already been considered in (Pardalos and Rodgers 1990)
- : no instance could be solved to optimality within 3h

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 three hours. By a brief comparison of Tables 1, 2, and 3, we can conclude that the LP-based method outperforms the one based on the solution of $(P_{\lambda_{min}e})$ for sparse instances, but the method based on the solution of $(P_{u^*})$ largely outperforms the LP-based method for all instances.

**Comparison to enumerative methods**

Here, we focus on a comparison of our approach to two enumerative methods. A well known enumerative method is the one proposed by Pardalos and Rodgers (1990). Hansen et al. (2000) improve this method by first transforming the objective function into a posiform.

The main feature of Pardalos and Rodgers algorithm consists in using first order boolean derivatives in order to progressively fix variables in a branch and bound process. Consider the quadratic function of binary variables $q(x) = \sum_i c_i x_i + \sum_i \sum_{j \neq i} c_{ij} x_i x_j$ and let $\Delta_i(x) = c_i + \sum_{j \neq i} c_{ij} x_j$ . The following property allows to fix some variables: if $\Delta_i(x) > 0$ for all $x$ then $x_i = 0$ in all optimal solutions, if $\Delta_i(x) < 0$ for all $x$ then $x_i = 1$ in all optimal solutions,

and if $\Delta_i(x) = 0$ for all $x$ then there exist an optimal solution with $x_i = 0$ and an optimal solution with $x_i = 1$.

In Hansen et al.'s algorithm, at the root of the search tree, thanks to a max-flow algorithm, $q(x)$ is written as $q(x) = c^* + \phi(x, \bar{x})$, $\phi(x, \bar{x})$ being a quadratic posiform and $c^*$ being a constant as large as possible. The lower bound $c^*$ belongs to the family of roof dual bounds (Hammer et al. 1984) and is precisely equal to the LP-bound reported in our Table 3. Recall that $\phi(x, \bar{x})$, a function of the literals $x_i$ and $\bar{x}_i = 1 - x_i$, is a quadratic posiform if $\phi(x, \bar{x}) = \sum_{k=1}^{m} a_k T_k$ with $a_k > 0$ for all $k$ and $T_k$ is a literal or a product of two literals. Then some variables are fixed by using the following so-called persistency property: all minimizing solutions of $q(x)$ satisfy $T_k = 0$ for all $k$ such that $T_k$ is a linear term (Hammer et al. 1984), (Billionnet and Sutter 1992). Then, the fixation of variables to 0 or 1 by branching or reduction tests can create new linear terms in the posiform that can be used by relation of the form $x_i + \bar{x}_i = 1$ to improve the lower bound. Conversely, in a classical way, the lower bound is used to fix some variables. Finally, fixation of variables are carried out by using the partial derivatives as in Pardalos and Rodgers algorithm.

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, and Table 4 reports their results.

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)$^{(1)}$ | HJM (2000)$^{(1)}$ | our$^{(2)}$ |
| 50 | 0.4 | 238 | 55 | 6.1 | 317.8 | 46.4 | 1.3 |
| 50 | 0.6 | 299 | 88 | 5.8 | - | 636.6 | 1.3 |
| 60 | 0.2 | 176 | 18 | 5.5 | 34.2 | 2.1 | 2.9 |
| 60 | 0.4 | 276 | 75 | 9.0 | - | 3525 | 2.6 |
| 70 | 0.3 | 232 | 53 | 6.1 | - | 6190 | 4.5 |
| 80 | 0.2 | 204 | 36 | 5.9 | - | 5011 | 10.1 |

(1): on a Sun Sparc SS20/514 MP with 128 MB of RAM
(2): on a PC with a Pentium IV of 1600 MHz and 1024 MB of RAM

Our method has an average gap equal to about 7%, smaller by a factor 8 compared to HJM(2000) and by a factor 37 compared to PR(1990). This makes us think that it would still need significantly less CPU time than the two others if a more accurate comparison was carried on by using the same machine.

## 4.2 Results for Glover et al. Instances

Here, we consider a set of 45 instances that are available in OR-Library (Beasley 1990). All these instances were generated by the Pardalos and Rodgers generator and are divided into 5 classes having different characteristics ($n$, density, and ranges of diagonal and off-diagonal coefficients). These classes are referred to as problem sets a, b, c, d, and e. Problems from classes a, b, and c were already introduced in (Pardalos and Rodgers 1990), the others were generated in (Glover et al. 1998).

We first run our method of Section 3 on the 25 instances from classes a, b, and c. Each of these instances are solved within a limit of two hours of CPU time (the largest one having $n = 125$ and $d = 1$). But, we also observed that all these instances are much faster solved by the trivial linearization method described above in Section 4.1. The total CPU time needed by the linearization method for these 25 instances is 79 seconds ! The largest CPU time needed by a single instance is 21 seconds and the largest number of branch and bound nodes is 3942. In view of these results and of the results reported in (Glover and al. 1998) about the exact solution of these instances by the Pardalos and Rodgers software, we can now consider that the linearization method is largely the best to solve these instances.

Then, we consider the 10 instances of class d. For all of them, $n = 100$, the linear (or diagonal) coefficients $c_i$ are in the range [-75, 75], 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. Table 5 reports the results of our method applied to these instances, where Column opt provides the optimal solution value. It shows that each of them can be solved within 10 minutes of CPU time. To the best of our knowledge, for 6 instances over 10, they are solved to optimality for the first time. We can also indicate that, when applied to these instances, the linearization method performs very well for instance 1d but fails in solving the remaining instances within a limit of 1 hour of CPU time.

Finally, we consider the instances from classes e ($n = 200$) and f ($n = 500$). These instances are globally beyond the practical capability of our method. However, the preprocessing phase by semidefinite programming remains tractable within about 7 minutes for

the largest instance ($n = 500$, $d = 1$) and the associated gap varies from 5.6% to 10.7%. Nevertheless, none of these instances could be solved within the limit of 1 hour of CPU time.

Table 5: Results through the solution of $(P_{u^*})$ for d-instances of Glover et al. ($n = 100$)

| | | | Preproc | MIQP solver | | |
|---|---|---|---|---|---|---|
| instance | density | opt | CPU1 | Gap (%) | CPU2 | nodes |
| 1d** | 0.1 | 6333* | 8.4 | 4.1 | 4 | 9320 |
| 2d | 0.2 | 6579* | 3.7 | 10.0 | 188 | 367962 |
| 3d | 0.3 | 9261* | 3.6 | 7.6 | 114 | 216551 |
| 4d | 0.4 | 10727* | 4.8 | 8.1 | 137 | 224352 |
| 5d | 0.5 | 11626 | 3.5 | 8.7 | 545 | 786445 |
| 6d | 0.6 | 14207 | 4.1 | 7.2 | 80 | 108890 |
| 7d | 0.7 | 14476 | 4.6 | 8.3 | 490 | 598217 |
| 8d | 0.8 | 16352 | 5.3 | 6.1 | 39 | 46885 |
| 9d | 0.9 | 15656 | 5.1 | 8.7 | 372 | 411085 |
| 10d | 1.0 | 19102 | 6.0 | 6.9 | 221 | 241924 |

* : these values are already announced to be optimal in (Boros et al. 2005)
** : instance 1d is solved within 1.7 seconds and 59 nodes by the linearization method

# 5 Computational results for the max-cut problem

A very large literature concerns max-cut, a central combinatorial optimization problem. Our objective here is to apply our method to max-cut. We make almost no adaptation of our general method described in Section 3. Our computational experiment gives a first idea of what can be obtained by our approach on a particularly-structured problem.

Let $G = (V, E)$ be a weighted undirected graph. The max-cut problem consists in finding a partition $(S,\ V \setminus S)$ of the set of vertices $V$ such that the total weight of the crossing edges is maximum. Many equivalent formulations exist for max-cut. We consider the following one, which matches our general problem $(P)$:

$$(MC) : \min\ \{mc(x)\ : x \in \{0,1\}^n\}$$

with

$$mc(x) = \ -\sum_{i=1}^{n}(\sum_{j\neq i}^{n} w_{ij})x_i + \sum_{i=1}^{n}\sum_{j\neq i} w_{ij}x_ix_j$$

where $w_{ij} = w_{ji}$ is the weight of edge $(i, j)$, $x_i$ is the decision variable equal to 1 if and only if vertex $i$ is in $S$, and $mc(x)$ is precisely the cut weight multiplied by -2.

Following the general method described in Section 3, we first solve the SDP relaxation $(DSDP_{MC})$ of $(MC)$:

$$\begin{array}{rll} (DSDP_{MC}): \min & -\sum_{i=1}^n (\sum_{j\neq i}^n w_{ij}) x_i + \sum_{i=1}^n \sum_{j\neq i} w_{ij} X_{ij} & \\ \text{s.t.:} & X_{ii} = x_i & i = 1, ..., n \\ & \begin{bmatrix} 1 & x^t \\ x & X \end{bmatrix} \succeq 0 & \\ & x \in \mathbb{R}^n \quad X \in \mathbb{R}^{n\times n} & \end{array}$$

and we get the optimal convexified function $mc'(x)$, which is equivalent to $mc(x)$ over $\{0,1\}^n$. Then, we solve the following quadratic 0-1 problem :

$$(MC'): \min \ \{mc'(x) \ : \ x_{i_0} = 1, \ x \in \{0,1\}^n\}$$

The variable fixation $x_{i_0} = 1$ is the only adaptation we introduce in our general method described in Section 3. It is a classical trick to break the symmetry of the max-cut solutions. We choose a vertex $i_0$ for which $\sum_{j\neq i} w_{ij}$ is maximum. We observed that this simple trick divides by about 4 the number of nodes while solving $(MC')$ by the MIQP solver. Observe that, for the results of Tables 6 and 7, we let the MIQP solver of CPLEX use the default branching rules for a general mixed integer problem. This turns out, for the max-cut problem, to select a node $i$ and branch on the fact that node $i$ is either in subset $S$ or not.

Following (Helmberg and Rendl 1998), we generate two types of test problems. The first, called $G_{.5}$, consists of unweighted graphs with edge probability 1/2. The second type, $G_{-1/0/1}$ is a weighted (complete) graph with edge weights chosen uniformly from $\{-1/0/1\}$.

Table 6: Results for the $G_{.5}$ instances through the solution of $(P_{u^*})$

| | Preproc | MIQP solver | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $n$ | CPU1 | Gap (%) | solved (3h) | CPU2 | | | final Gap (%) | nodes |
| | av. | av. | | min | av. | max | av. | av. |
| 40 | 0.9 | 2.5 | 10 | 0.2 | 1.0 | 3.2 | 0 | 1894 |
| 50 | 1.1 | 2.6 | 10 | 2.4 | 7.3 | 17.2 | 0 | 11239 |
| 70 | 2.8 | 2.4 | 10 | 7.2 | 103 | 128.4 | 0 | 242225 |
| 80 | 4.0 | 2.2 | 10 | 35.5 | 481 | 1934.4 | 0 | 876623 |
| 90 | 5.9 | 2.4 | 9 | 582 | 6044 | 10572 | 0.3 | 6317736 |
| 100 | 7.8 | 2.2 | 2 | 7071 | 7848 | 8624 | 0.4 | 7915644 |

Table 7: Results for the $G_{-1/0/1}$ instances through the solution of $(P_{u^*})$

| | Preproc | MIQP solver | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $n$ | CPU1 | Gap (%) | solved (3h) | CPU2 | | | final Gap (%) | nodes |
| | av. | av. | | min | av. | max | av. | av. |
| 40 | 0.5 | 14.5 | 10 | 0.1 | 0.3 | 0.6 | 0 | 1150 |
| 50 | 0.6 | 15.5 | 10 | 0.2 | 1.9 | 6.8 | 0 | 6897 |
| 70 | 1.1 | 16.2 | 10 | 32.6 | 73 | 190.6 | 0 | 154730 |
| 80 | 1.3 | 17.9 | 10 | 199 | 1016 | 2952 | 0 | 1419577 |
| 90 | 1.9 | 18.0 | 6 | 270 | 5208 | 10321 | 3.1 | 4787156 |
| 100 | 2.3 | 16.6 | 0 | | - | | 2.1 | - |

- : no instance could be solved to optimality within 3h

To explore the limits of our method for the max-cut problem, we run it on a 450 nodes graph from the $G_{.5}$ class. The SDP-based preprocessing phase takes 208 seconds of CPU time. In three hours of the MIQP solver phase, about 320000 nodes are explored. The deviation of the lower bound from the best cut, given after three hours, moves only from 1.56% to 1.51%.

Helmberg and Rendl (1998) consider the same SDP relaxation ($DSDP_{MC}$) and combine it with a cutting plane technique. This bounding process is embedded within a branch-and-bound algorithm. Our method seems to be competitive with the Helmberg and Rendl's from the computational time point of view. For example, with the 80 nodes $G_{.5}$ instances of Table 6, Helmberg and Rendl's method needs about one hour and 46 minutes (average on 2 instances) of CPU time on a HP9000/715 workstation while our's needs about 8 minutes, on the above Pentium IV PC. But their branch-and-bound algorithm performs 154 nodes when our's performs 876623 nodes.

In (Helmberg and Rendl 1998), the branching rule is carefully discussed and selected among five different rules. All these rules use the so-called edge-branching. Edge branching is commonly used for the max-cut problem. It consists in selecting an edge $(i, j)$ and branch on either $(i, j)$ contributes in the cut or not. It is well-known that, in both cases, the two vertices $i$ and $j$ can be contracted into a single vertex. In our approach, an implementation of this branching scheme can simply be done by adding constraint $x_i = x_j$ to the quadratic program of one node, and constraint $x_i + x_j = 1$ to the other node's. Hence, the convexity of the quadratic objective function is again preserved. An implementation of this specialized branch-and-bound algorithm would probably improve the results of Tables 6 and 7. But it

makes us loose the ability to use a general-purpose algorithm in the solution process.

# 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 MIQP solver of 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] J. Beasley. Heuristic algorithms for the unconstrained binary quadratic programming problem, 1998. Tech. Rep., Management School, Imperial College, London, UK, 1998.

[2] J.E. Beasley. Or-library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

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

[4] E. Boros, P. L. Hammer, and G. Tavares. The pseudo-boolean optimization website, 2005. http://rutcor.rutgers.edu/~pbo/index.htm.

[5] 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/.

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

[7] 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.

[8] F. Glover, G.A. Kochenberger, and B. Alidaee. Adaptive memory tabu search for binary quadratic programs. *Management Science*, pages 336–345, 1998.

[9] 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.

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

[11] 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.

[12] 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.

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

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

[15] 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.

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

[17] S. Kim and M. Kojima. Second order cone programming relaxation of nonconvex quadratic optimization problems. *Optimization Methods and Software*, 15:201–224, 2001.

[18] 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.

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

[20] 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.

[21] C. Lemaréchal and F. Oustry. SDP relaxations in combinatorial optimization from a Lagrangian point of view. In N. Hadjisavvas and P. M. Pardalos, editors, *Advances in Convex Analysis and Global Optimization*, pages 119–134. Kluwer, 2001.

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

[23] P. Merz and K. Katayama. Memetic algorithms for the unconstrained binary quadratic programming problem. *BioSystems*, 78(1-3):99–118, 2004.

[24] M. Muramatsu and T. Suzuki. A new second order cone programming relaxation for max-cut problems. *Journal of Operations Research Society of Japan*, 46:164–177, 2003.

[25] 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.

[26] 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.

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

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