Improving standard solvers convex reformulation for constrained quadratic 0-1 programs: the QCR method

Alain Billionnet[1], Sourour Elloumi[2], Marie-Christine Plateau[2]

December 23, 2006

[1]Laboratoire CEDRIC, Institut d'Informatique d'Entreprise,
18 allée Jean Rostand, F-91025 Evry
billionnet@ensiie.fr
[2]Laboratoire CEDRIC, Conservatoire National des Arts et Métiers,
292 rue Saint Martin, F-75141 Paris
{elloumi,mc.plateau}@cnam.fr

Abstract

Let (QP) be a 0-1 quadratic program which consists in minimizing a quadratic function subject to linear equality constraints. In this paper, we present QCR, a general method to reformulate (QP) into an equivalent 0-1 program with a convex quadratic objective function. The reformulated problem can then be efficiently solved by a classical branch-and-bound algorithm, based on continuous relaxation. This idea is already present in the literature and used in standard solvers such as CPLEX. Our objective in this work was to find a convex reformulation whose continuous relaxation bound is, moreover, as tight as possible. From this point of view, we show that QCR is optimal in a certain sense. State-of-the-art reformulation methods mainly operate a perturbation of the diagonal terms and are valid for any $\{0,1\}$ vector. The innovation of QCR comes from the fact that the reformulation also uses the equality constraints and is valid on the feasible solution domain only. Hence, the superiority of QCR holds by construction. However, reformulation by QCR requires the solution of a semidefinite program which can be costly from the running time point of view. We carry out a computational experience on three different combinatorial optimization problems showing that the costly computational time of reformulation by QCR can however result in a drastically more efficient branch-and-bound phase. Moreover, our new approach is competitive with very specific methods applied to particular optimization problems.

Keyword: Quadratic 0-1 programming, Convex quadratic programming, Semidefinite programming, Densest k-subgraph, Graph bisection, Task allocation, Experiments.

1 Introduction

Consider the following linearly-constrained zero-one quadratic program:

$$(QP) \ : \quad \text{Min} \quad \left\{ g(x) = x^t Q x + c^t x : Ax = b, \ x \in \{0,1\}^n \right\}$$

where c is an n real vector, b is an m real vector, Q is a symmetric $n \times n$ real matrix and A is an $m \times n$ matrix. Without loss of generality, we assume that all diagonal terms of Q are equal to 0.

Quadratic zero-one programming with linear constraints is a general model that allows to formulate numerous important problems in combinatorial optimization including, for example: quadratic assignment [19], graph partitioning [33], task allocation [5] and densest k-subgraph [31].

Various heuristics and exact methods have been proposed to solve (QP). Due to the non-convexity of the objective function, (QP) is often reformulated before searching for its optimal solution. So, several methods have been developed to solve it exactly through 0-1 linear reformulations (see, for example, [1], [2], [20], [22], [39]) or 0-1 convex quadratic reformulations (see, for example, [7], [12], [24], [34], [37]). This paper is concerned with the latter type of reformulation. Although 0-1 linear reformulations of (QP) are the most common approaches, other methods have been proposed. Let us cite, for example, algebraic and dynamic programming methods ([16], [25]), reformulation to a continuous concave minimization problem ([29]) and enumerative methods based on different types of relaxations such as lagrangian relaxation, semidefinite relaxation or convex quadratic relaxation ([3], [9], [11], [13], [18], [21], [23], [27], [35]).

In this paper we reformulate (QP) by an equivalent zero-one quadratic program with a convex objective function. Consequently, we can solve the transformed problem using general-purpose optimization software which implement branch-and-bound algorithms with a bounding procedure based on the optimal value of the continuous relaxation. We will show how to find the best convex reformulation of (QP), in a certain sense, by semidefinite programming.

The paper is organized as follows. In Section 2 we present QCR, a reformulation of (QP) by a zero-one quadratic program with a convex objective function. Section 3 reports computational experiments on the solution of the densest k-subgraph problem, the graph bisection problem and a task allocation problem. More precisely, we apply QCR on the one hand and the default preprocessing of CPLEX on the other hand in order to compare the efficiency of the two convexifications. Moreover, for the graph bisection problem, we compare QCR with a specific branch-and-bound algorithm, developed by Karisch, Rendl and Clausen [30]. Section 4 gives a conclusion.

2 The QCR method

Let $X = \{x : Ax = b,\ x \in \{0,1\}^n\}$ be the set of feasible solutions of problem (QP) and $\overline{X} = \{x : Ax = b,\ x \in [0,1]^n\}$ be the set of feasible solutions of the continuous relaxation of (QP).
The general term of matrix A is denoted by a_{ij} and the general term of Q by q_{ij}.

The objective function of (QP) is not convex. Consider the following zero-one quadratic problem equivalent to (QP) and depending on two parameters $\alpha \in \mathbb{R}^{m\times n}$ and $u \in \mathbb{R}^n$:

$$(QP_{\alpha,u}) \ : \quad \text{Min} \quad \{g_{\alpha,u}(x) : Ax = b,\ x \in \{0,1\}^n\}$$

where

$$\begin{aligned} g_{\alpha,u}(x) \ &= g(x) + \sum_{k=1}^{m}\left(\sum_{i=1}^{n}\alpha_{ki}x_i\right)\left(\sum_{j=1}^{n}a_{kj}x_j - b_k\right) + \sum_{i=1}^{n}u_i\left(x_i^2 - x_i\right) \\ &= x^tQ_\alpha x + c_\alpha^t x + \sum_{i=1}^{n}u_i\left(x_i^2 - x_i\right) \\ &= x^tQ_{\alpha,u}x + c_{\alpha,u}^t x \end{aligned}$$

and
$Q_\alpha = Q + \frac{1}{2}\left(\alpha^tA + A^t\alpha\right),\ \ Q_{\alpha,u} = Q_\alpha + Diag(u),$
$c_\alpha = c - \alpha^t b,\ \ c_{\alpha,u} = c_\alpha - u.$

$Diag(u)$ is a square n-matrix with the elements of u on the main diagonal.

It is easy to verify that for all $x \in X$, function $g_{\alpha,u}(x)$ is equal to $g(x)$. We are interested by the reformulations of $g(x)$ into $g_{\alpha,u}(x)$ if $g_{\alpha,u}(x)$ is convex over $\mathbb{R}^n$. This is always possible. Take α equal to the null matrix and $u = -\lambda e$ where λ is the smallest eigenvalue of matrix Q and e the n-vector of all ones. This amounts to the eigenvalue method introduced in [24].

The transformation of $g(x)$ into a convex function over $\mathbb{R}^n$ allows to solve $(QP_{\alpha,u})$ by a branch-and-bound algorithm based on continuous relaxation. It is well known that the behavior of such an algorithm is very dependent upon the bound at the root of the search tree. This bound is equal to the optimum value of the continuous relaxation of $(QP_{\alpha,u})$ that can be solved in polynomial time. So we are going to determine $\alpha \in \mathbb{R}^{n\times m}$ and $u \in \mathbb{R}^n$ such that $g_{\alpha,u}(x)$ is convex and the value of the continuous relaxation of $(QP_{\alpha,u})$ is as tight as possible. More precisely we have to solve the following problem:

$$(C(QP)) : \operatorname*{Max}_{\substack{\alpha\in\mathbb{R}^{n\times m}, u\in\mathbb{R}^n \\ Q_{\alpha,u}\succeq 0}} \ \operatorname*{Min}_{x\in\overline{X}} g_{\alpha,u}(x)$$

In the following theorem, we show that problem $(C(QP))$ is equivalent to the SDP-dual of a semidefinite relaxation $(SDQP)$ of problem (QP). Therefore, an optimal solution (α^*, u^*) of $(C(QP))$ can be obtained by solving $(SDQP)$, which can be done in polynomial time. For instance, Renegar [38] develops an interior point method for semidefinite programming, and shows that it can solve semidefinite programs to a prescribed accuracy in a polynomial number of arithmetic operations.

Theorem 1. *The optimum value of $(C(QP))$ is equal to the optimum value of the following semidefinite program $(SDQP)$, which is a semidefinite re-*

laxation of (QP)*:*

$$
(SDQP)\begin{cases}
Min \quad c^t x + \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij}X_{ij} & \\
s.t. \quad X_{ii} = x_i & i = 1,\dots,n \quad (1)\\
\quad -b_k x_i + \sum_{j=1}^{n} a_{kj}X_{ij} = 0 & i = 1,\dots,n;\ k = 1,\dots,m \quad (2)\\
\quad Ax = b & \\
\quad \begin{pmatrix} 1 & x^t \\ x & X \end{pmatrix} \succeq 0 & \\
\quad x \in \mathbb{R}^n,\ X \in \mathbf{S_n} &
\end{cases}
$$

where S_n *represents the set of* $n \times n$ *real symmetric matrices.*
The optimal values u_i^* *of* u_i $(i = 1\dots,n)$ *are given by the optimal values of the dual variables associated with constraints* (1) *and the optimal values* α_{ik}^* *of* α_{ik} $(i = 1,\dots,n; k = 1,\dots,m)$ *are given by the optimal values of the dual variables associated with constraints* (2)*.*

Proof. Consider the quadratic function $z_{\alpha,u,\beta}(x) = g_{\alpha,u}(x) + \beta^t(Ax - b)$ depending on the three multidimensional parameters α, u, β. Let $z_{\alpha,u,\beta}(x) = x^t Q_{\alpha,u} x + c^t_{\alpha,u,\beta} x - \beta^t b$ with $c^t_{\alpha,u,\beta} = c^t_{\alpha,u} + \beta^t A$. By observing that $x_i^2 \le x_i$ is equivalent to $0 \le x_i \le 1$, our convexification problem can be written as $(D1)$:

$$
(D1): \underset{\substack{\alpha\in\mathbb{R}^{n\times m}, u\in\mathbb{R}^n \\ Q_{\alpha,u}\succeq 0}}{\text{Max}} \quad \underset{x\in\mathbb{R}^n}{\text{Min}} \ \left\{ g_{\alpha,u}(x) : Ax = b, x_i^2 \le x_i (i = 1,\dots,n) \right\}
$$

$g_{\alpha,u}(x)$ is a convex function; the constraints $Ax = b$ and $x_i^2 \le x_i$ define a convex set. Therefore, by Lagrangian duality, $(D1)$ is equivalent to $(D2)$:

$$
(D2): \underset{\substack{\alpha\in\mathbb{R}^{n\times m}, u\in\mathbb{R}^n, \beta\in\mathbb{R}^m, \lambda\in\mathbb{R}^n_+ \\ Q_{\alpha,u}\succeq 0}}{\text{Max}} \quad \underset{x\in\mathbb{R}^n}{\text{Min}} \ \left\{ g_{\alpha,u}(x) + \sum_{i=1}^{n} \lambda_i (x_i^2 - x_i) + \beta^t(Ax - b) \right\}
$$

$(D2)$ is also equivalent to $(D3)$:

$$(D3): \underset{\substack{\alpha\in\mathbb{R}^{n\times m},u\in\mathbb{R}^n,\beta\in\mathbb{R}^m\\ Q_{\alpha,u}\succeq 0}}{\text{Max}} \quad \underset{x\in\mathbb{R}^n}{\text{Min}} \left\{g_{\alpha,u}(x)+\beta^t(Ax-b)\right\}$$

$$= \underset{\substack{\alpha\in\mathbb{R}^{n\times m},u\in\mathbb{R}^n,\beta\in\mathbb{R}^m\\ Q_{\alpha,u}\succeq 0}}{\text{Max}} \quad \underset{x\in\mathbb{R}^n}{\text{Min}} \; z_{\alpha,u,\beta}(x)$$

Indeed, if $(\alpha^*, u^*, \beta^*, \lambda^*)$ is an optimal solution of $(D2)$, $(\alpha^*, u^* = u^*+\lambda^*, \beta^*)$ is a feasible solution of $(D3)$ with the same value. Moreover, the optimal value of $(D2)$ is obviously greater than or equal to the optimal value of $(D3)$.

It is well known that a necessary condition for the quadratic function $z_{\alpha,u,\beta}(x)$ to have a minimum not equal to $-\infty$ is that matrix $Q_{\alpha,u}$ is positive semidefinite. Hence, $(D3)$ is equivalent to $(D4)$:

$$(D4): \underset{\alpha\in\mathbb{R}^{n\times m},u\in\mathbb{R}^n,\beta\in\mathbb{R}^m}{\text{Max}} \quad \underset{x\in\mathbb{R}^n}{\text{Min}} \quad z_{\alpha,u,\beta}(x)$$

This last problem is the Lagrangian dual obtained from (QP') by relaxing all the constraints:

$$(QP'): Min\left\{g(x): Ax=b, x_i^2=x_i (i=1,\dots,n), x_i\left(\sum_{j=1}^n a_{kj}x_j-b_k\right)(i=1,\dots,n; k=1,\dots,m)\right\}$$

Observe that (QP') is equivalent to the initial problem (QP). Following Lemaréchal and Oustry ([32], Corollary 4.2), the dual of (QP') is equivalent to the SDP problem $(D5)$:

$$(D5)\begin{cases} \text{Max} \quad r \\ \text{s.t.} \quad \begin{pmatrix} -\beta^t b - r & \frac{1}{2}c_{\alpha,u,\beta}^t \\ \frac{1}{2}c_{\alpha,u,\beta} & Q_{\alpha,u}\end{pmatrix} \succeq 0 \\ \qquad\quad r\in\mathbb{R}, \alpha\in\mathbb{R}^{m\times n}, u\in\mathbb{R}^n, \beta\in\mathbb{R}^m\end{cases}$$

Following again Lemaréchal and Oustry ([32], Theorem 4.4), if we apply SDP duality to problem $(D5)$, we get $(SDQP)$. Note that there is no duality gap since the feasible domain of $(SDQP)$ is nonempty (if (QP) admits a feasible solution).

$\square$

Remark

We can observe that, since $\sum_{k=1}^{m}\left(\sum_{i=1}^{n}\alpha^*_{ki}x_i\right)\left(\sum_{j=1}^{n}a_{kj}x_j - b_k\right)$ is null for all $x \in \overline{X}$, then:

$$\underset{x\in\overline{X}}{\text{Min}} \;\; g_{\alpha^*,u^*}(x) = \underset{x\in\overline{X}}{\text{Min}} \;\; g_{O,u^*}(x)$$

where O is the $m \times n$-null matrix.
However, $g_{O,u^*}(x)$ is not necessary convex over $\mathbb{R}^n$ and recall that our main objective is to convexify $g(x)$ over $\mathbb{R}^n$ in order to make the continuous relaxation easy to solve.

Example

Consider the following linearly constrained 0-1 quadratic programming problem whose optimal value is -80, obtained for $x_2 = x_3 = x_5 = 1$ and $x_1 = x_4 = 0$:

$$\begin{array}{lll}(E): & \text{Min} & \phi(x) = -9x_1 - 7x_2 + 2x_3 + 23x_4 + 12x_5 - 48x_1x_2 + 4x_1x_3 + 36x_1x_4 \\ & & -24x_1x_5 - 7x_2x_3 + 36x_2x_4 - 84x_2x_5 + 40x_3x_4 + 4x_3x_5 - 88x_4x_5 \\ & \text{s.t.} & \\ & & x_1 + x_2 + x_4 + x_5 = 2 \\ & & x_1, x_2, x_3, x_4, x_5 \in \{0,1\}\end{array}$$

The SDP relaxation of our example (E) is:

$$\begin{array}{lllll}(SDE): & \text{Min} & -9x_1 - 7x_2 + 2x_3 + 23x_4 + 12x_5 - 48X_{12} + 4X_{13} + 36X_{14} & & \\ & & -24X_{15} - 7X_{23} + 36X_{24} - 84X_{25} + 40X_{34} + 4X_{35} - 88X_{45} & & \\ & \text{s.t.} & & & \\ & & x_1 + x_2 + x_4 + x_5 = 2 & & \\ & & X_{1i} + X_{2i} + X_{4i} + X_{5i} = 2x_i & i = 1, \dots, 5 & \leftarrow \alpha^*_i \\ & & X_{ii} = x_i & i = 1, \dots, 5 & \leftarrow u^*_i \\ & & \begin{pmatrix} 1 & x^t \\ x & X \end{pmatrix} \succeq 0 & & \\ & & x \in \mathbb{R}^5, \; X \in S_5 & & \end{array}$$

Parameters u^* and α^* that allow us to build the new problem are obtained from the solution of (SDE). The optimal solution value of (SDE) equals

-88.02. It is therefore the optimal solution value of the continuous relaxation of the QCR-reformulated problem (E_{α^*,u^*}):

$$
(E_{\alpha^*,u^*}):\quad \begin{array}{ll}
\text{Min} & \phi(x) + (14x_1 + 18.6x_2 - 1.4x_3 + 0.12x_4 + 29.26x_5) \\
& (x_1 + x_2 + x_4 + x_5 - 2) + 24.6(x_1^2 - x_1) \\
& +3.38(x_2^2 - x_2) + 17.38(x_3^2 - x_3) + 111.46(x_4^2 - x_4) \\
& -7.4(x_5^2 - x_5) \\
\text{s.t.} & \\
& x_1 + x_2 + x_4 + x_5 = 2 \\
& \\
& x \in \{0,1\}^5
\end{array}
$$

Note that the bound obtained by the preprocessing of $CPLEX$ is -113.68.

3 Computational results

In this section, we present different applications of QCR: the densest k-subgraph problem (Section 3.1), the graph bisection problem (Section 3.2) and finally a task allocation problem (Section 3.3).

All the experiments have been carried out on a Pentium IV 2.2 GHz computer with 1 Go of RAM. For each instance, execution time limit is set to 1h.

We choose to solve semidefinite programs using SB ([26], [28]), a software applying the spectral bundle method on eigenvalue optimization problems, developed by Helmberg and Rendl [28]. (QP_{α^*,u^*}) is modeled with $AMPL$ and then solved by $CPLEX9$ [15].

For convenience, we use 'QCR+CPLEX' to refer to CPLEX with QCR preprocessing and 'CPLEX' to refer to the direct execution of CPLEX.

3.1 Computational results for the densest k-subgraph problem

Given an undirected graph $G = (V, U)$ with n nodes $\{v_1, ..., v_n\}$ and a positive integer k in $\{3, ..., n-2\}$, the densest k-subgraph problem consists in selecting a node subset $S \subseteq V$ of cardinality k and such that the subgraph of G induced by S contains as many edges as possible.

The densest k-subgraph problem can be formulated as the following linearly constrained 0-1 quadratic optimization problem (DS):

$$(DS): \ \text{Max}\left\{\sum_{i<j}\delta_{ij}x_ix_j : \sum_{j=1}^{n}x_j = k, \ x\in\{0,1\}^n\right\}$$

where the binary coefficient $\delta_{ij} = 1$ if and only if $[v_i, v_j]$ is an edge of G. The binary variable x_i is equal to 1 if and only if vertex v_i is in the k-subgraph. (DS) is also known under the name of *k-cluster problem* [14]. It can be also considered as a special case of the *k-dispersion-sum problem* [36].

The densest k-subgraph problem can be rewritten as follows:

$$(DS'): \ \text{Min}\left\{f(x) = x^tMx : \sum_{j=1}^{n}x_j = k, \ x\in\{0,1\}^n\right\}$$

where the general term of M is $m_{ij} = -\frac{1}{2}\delta_{ij}, \ \forall i,j$. Problems (DS) and (DS') are equivalent and their optimal values are opposite.

Now, we can apply the QCR approach to problem (DS').

According to the general method presented in Section 2, the reformulation of (DS') is:

$$(DS'_{\alpha,u}): \text{Min} \ \left\{f_{\alpha,u}(x) \ : \sum_{j=1}^{n}x_j = k, \ x\in\{0,1\}^n\right\}$$

where

$$f_{\alpha,u}(x) = x^tMx + \sum_{i=1}^{n}u_i(x_i^2 - x_i) + \sum_{i=1}^{n}\alpha_ix_i\left(\sum_{j=1}^{n}x_j - k\right)$$

.

Since problem (DS') has just one constraint, α is a vector with n components. The best parameters α^* and u^* are then computed by the SDP relaxation associated to (DS'), that we call $(SDDS')$ in the following.

We consider randomly generated instances of the densest k-subgraph problem. We take different graph sizes ($n = 40, 80, 100$), different densities ($d = 25\%, 50\%, 75\%$) and different k values ($k = \frac{n}{4}, \frac{n}{2}, \frac{3n}{4}$). For each couple (k, d), there are 5 instances (used in [4] for $n = 40$ and in [37] for $n = 80$). They are generated as follows: for a given density d and any pair of indexes (i, j) such that $i < j$, we generate a random number ρ from $[0, 1]$. If $\rho > d$ then δ_{ij} is set to 0, otherwise, δ_{ij} is set to 1.

All the results are reported in Table 1.

Legend of Table 1:

- d, density of the graph
- CPU, the average value of the CPU time, for five instances, required by CPLEX to solve (DS') or required by SB and CPLEX to solve $(SDDS')$ and (DS'_{α^*,u^*}) respectively. Note that a number $i < 5$ in brackets corresponds to the number of instances out of 5 solved within 1h. In this case, the corresponding CPU time is the average over these i instances.
- gap, the average value of the gap at the root node, defined as $\frac{bound-opt}{opt} * 100$ where opt is the value of the optimal or the best known solution and $bound$ is the optimal value of the continuous relaxation at the root of the branch-and-bound algorithm.

Table 1: Average results for randomly generated instances of k-cluster

			QCR+CPLEX		CPLEX	
n	k	$d(\%)$	CPU	$gap(\%)$	CPU	$gap(\%)$
40	$\lfloor \frac{n}{4} \rfloor$	25	0.16”	9.63	1’5”	91.4
		50	0.36”	9.32	-	154.53
		75	3.96”	12.28	-	257.40
	$\lfloor \frac{n}{2} \rfloor$	25	0.08”	3.06	5’41”	33.1
		50	0.22”	2.5	-	43.45
		75	0.10”	1.5	-	71.27
	$\lfloor \frac{3n}{4} \rfloor$	25	0.12”	1.08	13.88”	10.6
		50	0.04”	0.78	8’32” (4)	17.7
		75	0.03”	0.48	-	23.6
80	$\lfloor \frac{n}{4} \rfloor$	25	2’6”	9.16	-	114.4
		50	5’26”	8	-	170.7
		75	22’3”	6.44	-	225.3
	$\lfloor \frac{n}{2} \rfloor$	25	19.6”	2.96	-	42.9
		50	18.54”	1.8	-	62.1
		75	2’57”	1.32	-	78.2
	$\lfloor \frac{3n}{4} \rfloor$	25	0.92”	0.86	-	14.9
		50	2.09”	0.56	-	20.9
		75	3.6”	0.42	-	26.2
100	$\lfloor \frac{n}{4} \rfloor$	25	24’32”	9.1	-	124.1
		50	32’58”	7.88	-	180
		75	-	5.6	-	229.9
	$\lfloor \frac{n}{2} \rfloor$	25	7’54”	2.56	-	47.3
		50	6’32” (1)	2.18	-	67
		75	9’8”	1	-	79.3
	$\lfloor \frac{3n}{4} \rfloor$	25	9”	0.8	-	16.1
		50	50.81”	0.64	-	22.8
		75	14.32”	0.34	-	26.7

-: none of the five corresponding instances could be solved within 1h
(i): i instances out of 5 were solved within 1h

The running times corresponding to the computation of (α^*, u^*) by the semidefinite programming solver SB are very small, always less than one second. The computation of the bound, i.e. the solution of the continuous relaxation of (QP_{α^*,u^*}) is very fast too.

First, consider results of graphs with $n = 40$ where 'QCR+CPLEX' and 'CPLEX' are compared: on average, the gap of 'QCR+CPLEX' is about 20 times smaller than the gap of 'CPLEX'. The CPU time is hence drastically improved. Moreover, the optimal solution is always found by 'QCR+CPLEX' in less than one second of CPU time, except for $k = \lfloor \frac{n}{4} \rfloor$ and $d = 75\%$. In this case, the required CPU time is less than 5 seconds. Note that 'CPLEX' solves only 19 out of 45 instances with one hour of CPU time whereas 'QCR+CPLEX' solves all instances.

For $n = 80$ as for $n = 40$, the best results with 'QCR+CPLEX' are obtained for $k = \lfloor \frac{3n}{4} \rfloor$. In this case, the optimal solution is obtained within 2 seconds on average.

For $n = 100$ and $k = \lfloor \frac{3n}{4} \rfloor$, all the instances are solved with 'QCR+CPLEX' in less than one hour of CPU time (more precisely within 3'). It is also the case for $k = \lfloor \frac{n}{2} \rfloor$ when the density is 25% or 75%. In these experimental conditions, the most difficult instances for the QCR approach seem to be those with $k = \lfloor \frac{n}{4} \rfloor$. Note that, in this case, even if the optimal solution is rarely found in less than one hour of CPU time, the gap at the root of the search tree is relatively small, almost always less than 10%. 'CPLEX' is not efficient since the average gap is about 30 times larger than the one of 'QCR+CPLEX'.

All these results show that QCR largely outperforms the default preprocessing of CPLEX for this problem.

Instances with 40 and 80 nodes have already been used in [37]: eigenvalue methods have been applied to solve (QP). Note that for all instances, QCR improves results presented in [37] (i.e the bound value and therefore the CPU time): the gap obtained with QCR is about 2 times smaller. In addition, the 'QCR+CPLEX' approach allows to solve all instances with $n = 80$ and $k = \lfloor \frac{n}{4} \rfloor$ whereas, with the reformulation studied in [37], only 3 instances out of 15 are solved in less than one hour.

Moreover, we can solve larger problems than recent publications which use the same instances but different approaches. In [36], the reported results concern weighted instances with no more than 60 nodes. The integer linear programming approach presented in [4] and based on six different formulations doesn't allow to solve instances with 80 nodes, except for a density of 75%.

3.2 Computational results for the graph bisection problem

Let $G = (V, E)$ be an undirected graph with n nodes $\{v_1, \ldots, v_n\}$ and a set of weighted edges E. The graph bisection problem consists in dividing the nodes of G into two sets V_1 and V_2 such that $\mid V_1 \mid = p$ and $\mid V_2 \mid = n - p$ and such that the total weight of edges that have end-points in different sets is minimal. This problem can be formulated as the following linearly constrained zero-one quadratic problem:

$$(BP) \ : \ \text{Min} \ \left\{ g(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} \left(x_i(1-x_j) + (1-x_i)x_j \right) : \sum_{i=1}^{n} x_i = p, \ x \in \{0,1\}^n \right\}$$

where c_{ij} is the weight of edge $[v_i, v_j]$. The binary variable x_i is equal to 1 if and only if the node v_i is in V_1.

(BP) is equivalent to the following problem where, in the objective function, linear terms are separated from quadratic terms:

$$(BP) \ : \ \text{Min} \ \left\{ g(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} -2c_{ij}x_ix_j + \sum_{i=1}^{n} C_i x_i : \sum_{i=1}^{n} x_i = p, \ x \in \{0,1\}^n \right\}$$

where $C_i = \sum\limits_{j=1}^{i-1} c_{ji} + \sum\limits_{j=i+1}^{n} c_{ij}$. According to the general method presented in Section 2.1, the reformulation of (BP) is:

$$(BP_{\alpha,u}) \ : \ \text{Min} \ \left\{ g_{\alpha,u}(x) : \sum_{i=1}^{n} x_i = p, \ x \in \{0,1\}^n \right\}$$

with:

$$g_{\alpha,u}(x) = g(x) + \sum_{i=1}^{n} u_i \left(x_i^2 - x_i \right) + \sum_{i=1}^{n} \alpha_i x_i \left(\sum_{j=1}^{n} x_j - p \right)$$

The best parameters α^* and u^* are computed by the SDP relaxation associated to BP that we call $(SDBP)$ in the following.

Note that one constraint is added to (BP) when $p = \frac{n}{2}$ in order to accelerate the branch-and-bound algorithm, which consists in fixing to 1 the variable corresponding to a largest degree node.

3.2.1 Randomly generated unweighted graphs

We first apply QCR to randomly generated unweighted graphs. They are generated as follows: for a given density $d \in [0, 1]$ and a pair of indices (i, j) such that $i < j$, we generate a random number ρ from $[0, 1]$. If $\rho > d$ then c_{ij} is set to 0, otherwise, c_{ij} is set to 1.

Table 2 shows the results for four graph sizes ($n = 40, 80, 90, 100$), three densities ($d = 25\%, 50\%, 75\%$) and three p values ($p = \lfloor \frac{n}{8} \rfloor, \lfloor \frac{n}{4} \rfloor, \lfloor \frac{n}{2} \rfloor$). For each triplet (n, p, d), we solve 5 instances.

Legend of Table 2 :

- d, density of the graph.
- CPU, average value of the total CPU time required by CPLEX to solve (BP) or required by SB and CPLEX to solve $(SDBP)$ and (BP_{α^*,u^*}) respectively. Note that a number $i < 5$ in brackets corresponds to the number of instances out of 5 solved within 1h. In this case, the corresponding CPU time is the average over these i instances.
- gap, the average value of the gap at the root node, defined as $\frac{bound-opt}{opt} * 100$ where opt is the value of the optimal or the best known solution and $bound$ is the optimal value of the continuous relaxation at the root of the branch-and-bound algorithm.

Table 2: Average results for randomly generated unweighted instances for the graph bipartition problem

			QCR+CPLEX		CPLEX	
n	p	$d(\%)$	*CPU*	*gap*(%)	*CPU*	*gap*(%)
40	$\lfloor\frac{n}{8}\rfloor$	25	0.04”	12.45	1.17”	100
		50	0.07”	7.22	8.2”	100
		75	0.08”	4.32	53.4”	100
	$\lfloor\frac{n}{4}\rfloor$	25	0.15”	10.2	2’6”	100
		50	0.25”	6.33	-	100
		75	0.36”	3.36	-	100
	$\lfloor\frac{n}{2}\rfloor$	25	0.54”	8.05	13’9” (3)	89.37
		50	0.64”	3.08	-	93
		75	0.53”	2.02	-	93.7
80	$\lfloor\frac{n}{8}\rfloor$	25	5.7”	8.48	-	100
		50	3.8”	4.98	-	100
		75	14.8”	3.26	-	100
	$\lfloor\frac{n}{4}\rfloor$	25	2’18”	8.4	-	100
		50	4’04”	4.54	-	100
		75	5’09”	2.6	-	100
	$\lfloor\frac{n}{2}\rfloor$	25	24’21”	6.52	-	95.3
		50	10’51”	3.26	-	96.5
		75	6’49”	1.62	-	96.9
90	$\lfloor\frac{n}{8}\rfloor$	25	41.2”	10.04	-	100
		50	25.3”	5.14	-	100
		75	29.6”	3.08	-	100
	$\lfloor\frac{n}{4}\rfloor$	25	19’31” (4)	8.76	-	100
		50	13’40” (4)	3.48	-	100
		75	9’46” (3)	2.56	-	100
	$\lfloor\frac{n}{2}\rfloor$	25	36’1” (3)	5.92	-	96.2
		50	9’48” (1)	3.14	-	97
		75	23’18” (3)	1.6	-	97.4
100	$\lfloor\frac{n}{8}\rfloor$	25	1’22”	9.54	-	100
		50	2’9”	5.36	-	100
		75	3’4”	3	-	100
	$\lfloor\frac{n}{4}\rfloor$	25	27’13” (1)	7.9	-	100
		50	26’34” (1)	4.4	-	100
		75	22’40” (2)	2	-	100
	$\lfloor\frac{n}{2}\rfloor$	25	-	5.92	-	96.7
		50	-	3	-	97.5
		75	-	1.7	-	97.7

-: none of the five corresponding instances could be solved within 1h
(i): i instances out of 5 were solved within 1h

For Table 2, the running times corresponding to the computation of (α^*, u^*) by semidefinite programming are very small: they are always much smaller than the running time of the branch-and-bound phase.

We give below some additional comments on table 2.

$\underline{n = 40}$
'QCR+CPLEX': For each instance, the optimal value is computed within about 2 seconds.
'CPLEX': The gap is always equal to 100% for $p \neq \lfloor \frac{n}{2} \rfloor$. 'CPLEX' allows to solve only 23 instances out of 45 within 1h. Note that the lower bound becomes positive when $p = \lfloor \frac{n}{2} \rfloor$ but the gap associated remains much higher than the QCR one.

$\underline{n = 80}$
'QCR+CPLEX': For $p = \lfloor \frac{n}{8} \rfloor$, the CPU time is very small (always less than 37 seconds). For $p = \lfloor \frac{n}{4} \rfloor$, each instance is solved in less than 11 minutes. Worst results are obtained for $p = \lfloor \frac{n}{2} \rfloor$ and $d = 25\%$. However, in this case, all instances are solved within 1 hour.
'CPLEX': This direct approach is still inefficient since the lower bound is null when $p \neq \lfloor \frac{n}{2} \rfloor$ and when $p = \lfloor \frac{n}{2} \rfloor$, the gap associated is about 95%.

$\underline{n = 90}$
'QCR+CPLEX': For $p = \lfloor \frac{n}{8} \rfloor$, the worst CPU time is about 2 minutes. For $p = \lfloor \frac{n}{4} \rfloor$, 11 instances out of 15 are solved in less than 1 hour. However, one instance (with $d = 75\%$) requires more than 6 hours. For $p = \lfloor \frac{n}{2} \rfloor$, 7 instances out of 15 are solved in less than 1 hour. The worst CPU time equals 9 hours when $d = 50\%$.
'CPLEX': 'CPLEX' does not allow to solve any instance within 1h and the gap associated is either equal to 100% or about 97%.

$\underline{n = 100}$
'QCR+CPLEX': For $p = \lfloor \frac{n}{8} \rfloor$, the worst CPU time is about 6 minutes. For $p = \lfloor \frac{n}{4} \rfloor$, the worst CPU time, equal to 12h30', is obtained for $d = 50\%$. However, the average of the CPU time for all the 15 instances and for this value of p is 3h16'. For $p = \lfloor \frac{n}{2} \rfloor$, the worst CPU time is equal to about 50 hours for one instance with $d = 50\%$. On average, all the 15 instances are solved in 24 hours.
'CPLEX': 'CPLEX' does not allow to solve any instance within 1h and the gap associated is either equal to 100% or about 97%.

As a general remark, we can observe that, for the QCR approach,whatever the graph sizes are, the best gap values are obtained for largest densities. The average gap is equal to 9% for $d = 25\%$, 4% for $d = 50\%$ and 3% for $d = 75\%$.

Moreover, the longest running times are obtained for the graph equicut problem (i.e. $p = \lfloor \frac{n}{2} \rfloor$): intuitively, we can think that the partition size p increases the difficulty of the problem when it tends towards the value $\frac{n}{2}$ since the number of feasible solutions is equal to the number of combinations of p objects taken among n.

Finally, QCR always outperfoms the default preprocessing of CPLEX9.

3.2.2 Comparison with Karisch, Rendl and Clausen (KRC) method

In this section, we compare our method with the one of Karisch, Rendl and Clausen (KRC) who designed an exact solution method for the graph bisection problem [30]. Their approach is a specific branch-and-bound algorithm based on semidefinite programming and polyhedral relaxations.

Comparison with the KRC method based on Brunetta, Conforti and Rinaldi instances
Karisch, Rendl and Clausen tested their approach on a library created by Brunetta, Conforti and Rinaldi (BCR) [10]. We choose to solve instances from it (*ftp://ftp.math.unipd.it/pub.Misc.equicut*) and compare our results with the ones of KRC. The instances correspond to $p = \lfloor \frac{n}{2} \rfloor$ and are divided into 4 classes: *Random Instances*, *Toroidal Grid Instances*, *Mixed grid Instances*, *Instances with Negative Weights*. We only present results for *Toroidal Grid Instances*. For more information, all results and more complete details are available in [8].

Note that our computer is about 15 times faster than the one of Karisch, Rendl and Clausen whose experiments were performed on a HP 9000/735. So, in following Tables, we report CPU times required by the KRC method divided by 15.

Legend of Table 3:

- *opt*, value of the optimal solution
- CPU_{QCR}, total CPU time required by the 'QCR+CPLEX' method
- $gap_{QCR} = \frac{opt-bound}{bound}$ where *bound* is the optimal value of the continuous relaxation of the QCR reformulated problem
- CPU_{KRC}, total CPU time required by the KRC method on a HP 9000/735 (divided by 15).

Table 3: Equicut of toroidal grid instances from the BCR-library

problem	n	$d(\%)$	*opt*	CPU_{KRC}	CPU_{QCR}	$gap_{QCR}(\%)$
4x5t	20	21	28	0.06”	0.01”	12.6
6x5t	30	14	31	0.2”	0.02”	26.9
8x5t	40	10	33	0.4”	0.09”	38.5
12x2t	42	10	9	0.34”	0.05”	66.5
23x2t	46	9	9	8.3”	0.2”	78
4x12t	48	9	24	1.13”	0.14”	54.2
5x10t	50	8	33	0.4”	0.25”	50
10x6t	60	7	42	23.33”	2.22”	53.5
7x10t	70	6	45	38.13”	3.69”	54.2
10x8t	80	5	43	1’3”	2.29”	47.7
	Average value			**13.53”**	**0.9”**	**48.21**

We solve all instances to optimality. For all instances, the CPU time required to find the optimal solution is drastically improved compared with the results found by KRC: CPU_{QCR} can be up to 40 times smaller than CPU_{KRC} taking into account the difference of computers. Our method is 15 times faster and in spite of rather large gap, due to the small densities, the CPU time is very small.

Comparison with the KRC results obtained on their randomly generated instances

Finally, we apply our method on random instances, generated by Karisch et al. [30]. The graphs are unweighted random graphs with uniform edge probability $\frac{1}{2}$. Instances got a number of nodes varying from 36 to 84, and the p values are $\lfloor\frac{n}{2}\rfloor$, $\lfloor\frac{3n}{4}\rfloor$, $\lfloor\frac{7n}{12}\rfloor$ and $\lfloor\frac{13n}{24}+\frac{1}{2}\rfloor$. Table 4 presents the results.

Table 4: Randomly generated unweighted instances

graph	n	p	*opt*	CPU_{KRC}	CPU_{QCR}	$gap_{QCR}(\%)$
ex36a	36	$\lfloor \frac{n}{2} \rfloor$	117	3”	0.22”	
ex60a	60		367	20”	11.39”	3.4
ex84a	84		742	2h02’55”	40’06”	3.5
ex36a	36	$\lfloor \frac{3n}{4} \rfloor$	85	31”	0.13”	
ex60a	60		268	9’56”	8.95”	5.96
ex84a	84		548	8’20”	12’24”	4.9
ex36a	36	$\lfloor \frac{7n}{12} \rfloor$	112	0.67”	0.21”	
ex60a	60		351	22.53”	3.56”	2.9
ex84a	84		721	37’7”	2h17’24”	4.08
ex36a	36	$\lfloor \frac{13n}{24} + \frac{1}{2} \rfloor$	114	0.2”	0.13”	4.5
ex60a	60		360	21.2”	8.72”	3.1
ex84a	84		735	20’15”	1h52’48”	3.8
	Average value			**16’41”**	**25’16”**	

As Karisch et al. noted, randomly generated instances of this type constitute the most difficult classes. Let us remark that the p values are, for each instance, very close to $\frac{n}{2}$ (considered as the most difficult case by several authors). All instances are solved to optimality. For largest graphs, the solution time for the different densities are very small for $n = 36$ (less than 1”) and $n = 60$ (less than 1’39”). For $n = 84$, several minutes and even hours are required (between 12’ and 2h17’). Note that, for all instances, the gap is rather small.

Now, if we compare our results with the ones of KRC, our method is not always better. For instance, when $n = 84$ and $p = \lfloor \frac{7n}{12} \rfloor$, the optimal value is found 3.7 times faster with their approach than with the QCR one. But we can say that Karisch et al. apply a specific method to the graph bisection problem whereas QCR is a very general approach which can be applied to many combinatorial optimization problems.

3.3 Computational results for the task allocation problem

Let $P = \{P_1, \ldots, P_n\}$ be a set of non identical processors of a distributed system and $T = \{T_1, \ldots, T_m\}$ be a set of tasks that must be run over this distributed system. Some of these tasks have to communicate.

The task allocation problem can be formulated as the following linearly constrained 0-1 quadratic optimization problem (TA):

$$(TA) \ : Min \ \ \left\{ l(x) = \sum_{i=1}^{m}\sum_{k=1}^{n} e_{ik}x_{ik} + \sum_{i=1}^{m-1}\sum_{j=i+1}^{m}\sum_{k=1}^{n}\sum_{\ell=1}^{n} c_{ikjl}x_{ik}x_{jl} \ : \right.$$
$$\left. \sum_{k=1}^{n} x_{ik} = 1 \ (i = 1, \ldots, m), x \in \{0,1\}^{m\times n} \right\}$$

where the real coefficient e_{ik} represents the execution cost of task T_i when it is assigned to processor P_k and the real coefficient c_{ikjl} represents the communication cost betweem two tasks T_i and T_j (respectively assigned to processors P_k and P_ℓ). The binary variable x_{ik} is equal to 1 if and only if the task T_i is assigned to processor P_k.

According to the general method presented in Section 2.1, the reformulation of (TA) is:

$$(TA_{\alpha,u}) \ : \ \ \text{Min} \ \ \left\{ l_{\alpha,u}(x) : \sum_{k=1}^{n} x_{ik} = 1, \ (i = 1, \ldots, m), x \in \{0,1\}^{m\times n} \right\}$$

with:

$$l_{\alpha,u}(x) = l(x) + \sum_{i=1}^{m}\sum_{k=1}^{n} u_{ik}\left(x_{ik}^2 - x_{ik}\right) + \sum_{i=1}^{m}\left(\sum_{j=1}^{m}\sum_{l=1}^{n}\alpha_{ijl}x_{jl}\right)\left(\sum_{k=1}^{n} x_{ik} - 1\right)$$

Since problem (TA) has m constraints, α is a $mn \times m$-matrix. Then, the best parameters α^* and u^* are computed by the SDP relaxation associated to (TA) that we call $(SDTA)$ in the following. For comparison of 'CPLEX' and 'QCR+CPLEX', we consider randomly generated instances: we choose the parameters m and n who define the problem dimension and then the coefficients e_{ik} and c_{ikjl} are randomly selected from an interval $[-50, 50]$. These instances have been already used in [6].

Table 5 presents results for all instances that we have considered: 3 values of n $(3, 4, 5)$ and 4 values of m $(10, 15, 18, 20)$. For each pair (m, n), average values (over 10 instances) of computation time and gap are reported.

Legend of Table 5:

- CPU, average value of the total CPU time required by CPLEX to solve (TA) or required by SB and CPLEX to solve $(SDTA)$ and (TA_{α^*,u^*}) respectively. Note that a number i in brackets corresponds to the number of instances out of 10 solved within 1h. In this case, the corresponding CPU time is the average of these i instances.
- *gap*, the average value of the gap at the root node, defined as $\frac{bound-opt}{opt} * 100$ where *opt* is the value of the optimal or the best known solution and *bound* is the optimal value of the continuous relaxation at the root of the branch-and-bound algorithm.
- $nodes$, number of nodes in the search tree

Table 5: Average results for the task allocation problem

		QCR+CPLEX		CPLEX	
m	n	CPU	$gap(\%)$	CPU	$gap(\%)$
15	5	21"	30	4'16"	63
18	4	27"	22	4'34"	60
20	4	35"	24	21'4"	62
20	5	9'59"	34	-	67
25	4	9'14"	24	-	59
25	5	48'27"(1)	35	-	67
25	6	-	46	-	76

-: none of the ten corresponding instances could be solved within 1h
(i): i instances out of 10 were solved within 1h

For all instances, QCR drastically improves the default preprocessing of CPLEX. More complete comparisons are presented in [17]. In this reference, our 'QCR+CPLEX' method is compared with a sophisticated linear programming based method. For the $m = 20$ and $n = 5$ instances, it is observed that the average gap (resp. CPU time) of the linear programming based method is 84% (resp. 36'46") versus 34% (resp. 9'59").

4 Conclusion

In this paper, we have considered the problem (QP) of minimizing a quadratic 0-1 function $g(x)$ subject to linear equality constraints $\sum_{j=1}^{n} a_{kj}x_j = b_k \ \forall k = 1, \ldots, m$. We have proposed a reformulation of this problem into an equivalent 0-1 program with a convex quadratic objective function $g_{\alpha,u}(x)$ depending on two parameters α and u and obtained by adding to $g(x)$ the two following functions, null on the feasible solution set, $g_1(x) = \sum_{i=1}^{n} u_i \left(x_i^2 - x_i\right)$ and $g_2(x) = \sum_{k=1}^{m} \left(\sum_{i=1}^{n} \alpha_{ki} x_i\right) \left(\sum_{j=1}^{n} a_{kj} x_j - b_k\right)$. The n-vector u and the $m \times n$-matrix α are determined by semidefinite programming in order to make $g_{\alpha,u}(x)$ convex and to maximize its value over the relaxed domain $\overline{X}$. This reformulation, that we call QCR (Quadratic Convex Reformulation), can be viewed as a preprocessing of (QP) in order to solve it by a general-purpose (MIQP) solver. We applied QCR to three difficult combinatorial optimization problems: the densest k-subgraph, the graph bisection and a task allocation problem. For all the considered instances of these problems, (QCR) largely outperforms the method consisting to directly submit the quadratic 0-1 problem to a (MIQP) solver such as CPLEX.

Moreover, for the densest k-subgraph problem, QCR also outperforms the methods proposed in the literature (specific algorithms or linearizations). For the graph bisection problem, QCR is competitive with the best method - to our knowledge - proposed in the literature: the one of Karisch, Rendl and Clausen (KRC) that is a specific branch-and-bound algorithm based on semidefinite programming and polyhedral relaxations.

Finally, QCR is a general approach which can be applied to a lot of combinatorial optimization problems. We are currently trying to extend QCR to quadratic 0-1 programs involving linear inequality constraints.

References

[1] W.P. Adams, R. Forrester, and F. Glover. Comparisons and enhancement strategies for linearizing mixed 0-1 quadratic programs. *Discrete Optimization*, 1(2):99–120, 2004.

[2] W.P. Adams and H.D. Sherali. A tight linearization and an algorithm for 0-1 quadratic programming problems. *Management Science*, 32(10):1274–1290, 1986.

[3] K.M. Anstreicher and N.W. Brixius. A new bound for the quadratic assignment problem based on convex quadratic programming. *Mathematical Programming*, 89:341–357, 2001.

[4] A. Billionnet. Different formulations for solving the heaviest k-subgraph problem. *Information Systems and Operational Research*, 43(3):171–186, 2005.

[5] A. Billionnet, M.-C. Costa, and A. Sutter. An efficient algorithm for a task allocation problem. *Journal of the association for computing machinery*, 39:502–518, 1992.

[6] A. Billionnet and S. Elloumi. Best reduction of the quadratic semi-assignment problem. *Discrete Applied Mathematics*, 109:197–213, 2001.

[7] A. Billionnet and S. Elloumi. Using a mixed integer quadratic programming solver for the unconstrained quadratic 0-1 problem. *Mathematical Programming*, 109:55–68, 2007.

[8] A. Billionnet, S. Elloumi, and M.C. Plateau. Quadratic convex reformulation : a computational study of the graph bisection problem. *Technical Report CEDRIC*, http://cedric.cnam.fr/PUBLIS/RC1003.pdf, 2005.

[9] A. Billionnet and E. Soutif. An exact method based on lagrangian decomposition for the 0-1 quadratic knapsack problem. *European Journal of Operational Research*, 157(3):565–575, 2004.

[10] L. Brunetta, M. Conforti, and G. Rinaldi. A branch-and-cut algorithm for the equicut problem. *Mathematical Programming*, 78:243–263, 1997.

[11] P. Carraresi and F. Malucelli. A new lower bound for the quadratic assignment problem. *Operations Research*, 40(Suppl. No1):S22–S27, 1992.

[12] M.W. Carter. The indefinite zero-one quadratic problem. *Discrete Applied Mathematics*, 7:23–44, 1984.

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

[14] D.G. Corneil and Y. Perl. Clustering and domination in perfect graphs. *Discrete Applied Mathematics*, 9:27–39, 1984.

[15] Cplex. Ilog cplex 9.0 reference manual. *ILOG CPLEX Division, Gentilly, France*, http://www.ilog.com/products/cplex, 2004.

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

[17] S. Elloumi. Linear programming versus convex quadratic programming for the module allocation problem. *Technical Report CEDRIC 1100*, http://cedric.cnam.fr/PUBLIS/RC1100.pdf, 2005.

[18] A. Faye and F. Roupin. Partial lagrangian and semidefinite relaxations of quadratic programs. *Rapport technique CEDRIC N°673*, http://cedric.cnam.fr/PUBLIS/RC673.pdf, 2004.

[19] G. Finke, R.E. Burkard, and F. Rendl. Quadratic assignment problems. *Annals of Discrete Mathematics*, 31:61–82, 1987.

[20] R. Fortet. Applications de l'algèbre de boole en recherche opérationnelle. *Revue Française d'Automatique d'Informatique et de Recherche Opérationnelle*, 4:5–36, 1959.

[21] A.M. Frieze and J. Yadegar. On the quadratic assignment problem. *Discrete applied mathematics*, 5:89–98, 1983.

[22] F. Glover and E. Woolsey. Further reduction of 0-1 polynomial programming problems to 0-1 linear programming problems. *Operations Research*, 21:156–161, 1973.

[23] M.X. Goemans. Semidefinite programming in combinatorial optimization. *Mathematical Programming*, pages 143–161, 1997.

[24] P.L. Hammer and A.A. Rubin. Some remarks on quadratic programming with 0-1 variables. *RAIRO*, 3:67–79, 1970.

[25] P.L. Hammer and S. Rudeanu. Boolean methods in operations research. *Springer, Berlin*, 1968.

[26] C. Helmberg. A c++ implementation of the spectral bundle method. *Manual version 1.1.1*, http://www.zib.de/helmberg/SBmethod, 2000.

[27] C. Helmberg and F. Rendl. Solving quadratic (0,1)-problems by semidefinite programs and cutting planes. *Math. Programming*, 8(3):291–315, 1998.

[28] C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization*, pages 673–696, 2000.

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

[30] S.E. Karisch, F. Rendl, and J. Clausen. Solving graph bisection problems with semidefinite programming. *INFORMS Journal on Computing*, 12(3):177–191, 2000.

[31] J Krarup, D. Pisinger, and F. Plastria. Discrete location problems with push-pull objectives. *Discrete Applied Mathematics*, 123:365–378, 2002.

[32] C. Lemaréchal and F. Oustry. Semidefinite relaxations and lagrangian duality with application to combinatorial optimization. *RR-3710, INRIA Rhones-Alpes*, 1999.

[33] T. Lengauer. Combinatorial algorithms for integrated circuit layout. *Wiley,Chichester*, 1990.

[34] R.D. McBride and J.S. Yormark. An implicit enumeration algorithm for quadratic integer programming. *Management Science*, 26(3), 1980.

[35] P. Michelon and N. Maculan. Lagrangian decomposition for integer non linear programming with linear constraints. *Mathematical Programming*, 52(2):303–314, 1991.

[36] D. Pisinger. Upper bounds and exact algorithm for p-dispersion problems. *Computers and Operations Research*, 33(5):1380–1398, 2006.

[37] M.C. Plateau, A. Billionnet, and S. Elloumi. Eigenvalue methods for linearly constrained quadratic 0-1 problems with application to the densest k-subgraph problem. *In 6ème congrès ROADEF, Tours, 14-16 février, Presses Universitaires Francois Rabelais*, pages 55–66, 2005.

[38] J. Renegar. A mathematical view of interior-point methods in convex optimization. *MPS-SIAM Series on Optimization, SIAM, Philadelphia*, 2001.

[39] H.D. Sherali and W.P. Adams. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems.* Kluwer Academic Publishers, Norwell, MA, 1999.