Convex Quadratic Programming for Exact Solution of 0-1 Quadratic Programs

Alain Billionnet[1], Sourour Elloumi[2], Marie-Christine Plateau[3]

10th June 2005

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

Abstract

Let (QP) be a 0-1 quadratic program which consists in minimizing a quadratic function subject to linear constraints. In this paper, we present a general method to solve (QP) by reformulation of the problem into an equivalent 0-1 program with a convex quadratic objective function, followed by the use of a standard mixed integer quadratic programming solver. Our convexification method, which is the best in a certain sense, uses the equality constraints of (QP) and requires the solution of a semidefinite program. We apply it to the densest k-subgraph problem and report experimental results showing that, for this graph problem, the approach outperforms existing methods.

Keyword : Quadratic 0-1 programming, Convex quadratic programming, semidefinite programming, densest k-subgraph, Experiments.

1 Introduction

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

$$(QP) \ : \quad \text{Min} \quad \left\{ g(x) = x^t Q x + c^t x : Ax = b, \ A'x \leq b', \ x \in \{0,1\}^n \right\}$$

where c is an n real vector, b is an m real vector, b' is a p real vector, Q is a symmetric $n \times n$ real matrix, A is an $m \times n$ matrix and A' is a $p \times n$ 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, graph partitioning, task allocation, capital budgeting and densest k-subgraph.

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, [2], [3], [13], [15], [30]) or 0-1 convex quadratic reformulations (see, for example, [6], [9], [17], [25], [28]). 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 ([11], [18]), reformulation to a continuous concave minimization problem ([22]) and enumerative methods based on different types of relaxations as lagrangian relaxation, semidefinite relaxation or convex quadratic relaxation ([4], [7], [8], [10], [12], [14], [16], [20], [26]).

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 a 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 a reformulation of (QP) by a zero-one quadratic program with a convex objective function. We situate the approach within other related methods in Section 3. Section 4 presents the solution of a small instance of problem (QP) by these different methods. Section 5 reports computational experiments on the solution of the densest k-subgraph problem, a particular case of (QP). Section 6 is a conclusion.

2 Our Approach

Let $X = \{x : Ax = b,\ A'x \leq b',\ x \in \{0,1\}^n\}$ be the set of feasible solutions of problem (QP) and $\overline{X} = \{x : Ax = b,\ A'x \leq b',\ x \in [0,1]^n\}$ 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}. Let e be the n-vector of 1 and O the $m \times n$ null-matrix .

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 $u \in \mathbb{R}^n$:

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

where

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

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 $x \in \{0,1\}^n$ such that $Ax = b$, the 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. This is always possible. Take $\alpha = O$ and $u = -\lambda e$ where λ is the smallest eigenvalue of matrix Q. Obviously $g_{\alpha,u}(x)$ is convex. This particular value of (α, u) is denoted (α^1, u^1) in Section 3. The transformation of $g(x)$ into a convex function allows to solve $(QP_{\alpha,u})$ by a Mixed Integer Quadratic Programming (MIQP) solver such as CPLEX [1]. It is well known that the behavior of the associated Branch & Bound 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 $(QP_{\alpha,u})$. 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 maximal. More precisely we have to solve the following problem :

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

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 $(SDQP)$.

Theorem 1. *The optimum value of* $(C(QP))$ *is equal to the optimum value of the following semidefinite program*

$$(SDQP)\begin{cases} Min \quad c^tx + \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij}X_{ij} & \\ s.t. \quad X_{ii} = x_i & i = 1,\ldots,n \quad (1) \\ -b_kx_i + \sum_{j=1}^{n} a_{kj}X_{ij} = 0 & i = 1,\ldots,n;\; k = 1,\ldots,m \quad (2) \\ Ax = b & \\ A'x \le b' & \\ \begin{pmatrix} 1 & x^t \\ x & X \end{pmatrix} \succeq 0 & \\ x \in \mathbb{R}^n,\; X \in \mathbb{R}^{n\times n} & \end{cases}$$

The optimal values u_i^* *of* u_i $(i = 1\ldots,n)$ *are given by the optimal values of the dual variables associated with constraints* (1) *and the optimal values* α_{ik}^* *of* α_{ik} $(i = 1,\ldots,n; k = 1,\ldots,m)$ *are given by the optimal values of the dual variables associated with constraints* (2).

Proof. $q_{\alpha,u}(x)$ being a convex function and $\{x : Ax = b,\; A'x \le b'\}$ a convex set, by Lagrangian duality $(C(QP))$ is equivalent to

$$(LD) : \underset{\substack{\alpha\in\mathbb{R}^{n\times m}, u\in\mathbb{R}^n \\ Q_{\alpha,u}\succeq 0}}{Max} \quad \underset{\substack{\beta^t\in\mathbb{R}^m \\ \beta'^t\in\mathbb{R}^{p+}}}{Max} \quad \underset{x\in[0,1]^n}{Min} \; \left(c_{\alpha,u}^tx + x^tQ_{\alpha,u}x + \beta^t(Ax-b) + \beta'^t(A'x-b')\right)$$

For given values of parameters $\alpha \in \mathbb{R}^{n\times m}$, $\beta \in \mathbb{R}^n$ and $\beta' \in \mathbb{R}^{p+}$, consider the following problem :

$$(L_{\alpha,\beta,\beta'}) : \underset{\substack{u\in\mathbb{R}^n \\ Q_{\alpha,u}\succeq 0}}{Max} \quad \underset{x\in[0,1]^n}{Min} \quad \left(c^t_{\alpha,u}x + x^tQ_{\alpha,u}x + \beta^t(Ax-b) + \beta'^t(A'x-b')\right)$$

It is known that problem $(L_{\alpha,\beta,\beta'})$ has the same optimal value as the semidefinite program $(P_{\alpha,\beta,\beta'})$ (see, for example [23], [24], [29], [31]) and the optimal values of u_i for problem $(L_{\alpha,\beta,\beta'})$ are given by the optimal values of the dual variables associated with constraints (3).

$$(P_{\alpha,\beta,\beta'}) \begin{cases} \text{Min} \quad c^t_\alpha x + \sum\limits_{i=1}^n \sum\limits_{j=1}^n Q_{\alpha_{ij}} X_{ij} + \beta^t(Ax-b) + \beta'^t(A'x-b') \\ \text{s.t.} \quad X_{ii} = x_i \qquad i = 1,\ldots,n \qquad (3) \\ \qquad \begin{pmatrix} 1 & x^t \\ x & X \end{pmatrix} \succeq 0 \\ \qquad x \in \mathbb{R}^n, \; X \in \mathbb{R}^{n\times n} \end{cases}$$

where $Q_{\alpha_{ij}}$ is the (i,j) term of matrix Q_α.

Hence, Problem (LD) consists in solving $\underset{\alpha\in\mathbb{R}^{n\times m},\beta\in\mathbb{R}^n,\beta'\in\mathbb{R}^{p+}}{Max} v(P_{\alpha,\beta,\beta'})$ where $v(P_{\alpha,\beta,\beta'})$ is the optimal value of $(P_{\alpha,\beta,\beta'})$. The objective function of $(P_{\alpha,\beta,\beta'})$ is linear in x and X, and its feasible solution set is convex. Therefore, again by Lagrangian duality, $(C(QP))$ has the same optimal value as $(SDQP)$. ■ □

So, *Algorithm 1* is a solution algorithm for (QP):

Algorithm 1:

1. Solve the semidefinite program $(SDQP)$. Let α^* and u^* be respectively the optimal values of the dual variables associated to constraints (2) and (1) of $(SDQP)$.

2. Solve the quadratic problem (QP_{α^*,u^*}) whose continuous relaxation is convex.

3 Related methods

The reformulation of (QP) into an equivalent quadratic 0-1 problem with a convex objective function has already been considered in the literature.

Hammer and Rubin [17] propose to reformulate (QP) into:

$$\operatorname*{Min}_{x \in X} \left(x^t Q x + c^t x - \lambda_{\min}(Q) \sum_{i=1}^{n} (x_i^2 - x_i) \right)$$

where $\lambda_{\min}(Q)$ is the smallest eigenvalue of Q. This method can be viewed as solving (QP_{α^1,u^1}) with $\alpha^1 = O$ and u^1 is determined by solving $(C(QP))$ subject to $u_i = u$ $(i = 1, \ldots, n)$ and $\alpha = O$. In this case, it is easy to compute u^1 without using semidefinite programming since $u^1 = -\lambda_{\min}(Q)e$. Computational results with this method, that we call the eigenvalue method, are reported in [6] for the unconstrained binary quadratic problem.

In [6], another exact solution approach is proposed for the unconstrained binary quadratic problem. This approach can be viewed as the application of our *Algorithm 1* to a particular case of problem (QP). Computational results reported in this reference show that this last method drastically improves the eigenvalue method. It was already suggested in [6] to extend the approach to the constrained case, without using the constraints in the reformulation process. This idea can be interpreted as the particular reformulation (QP_{α^2,u^2}) of (QP) where $\alpha^2 = O$ and u^2 being determined by solving $(C(QP))$ subject to $\alpha = O$, i.e. the semidefinite program $(SDQP)$ without constraints (2). Note that u^2 is equal to the optimal value of the dual variable associated with constraints $X_{ii} = x_i$.

In [28], the general linearly-constrained 0-1 quadratic programming problem is considered. The eigenvalue method is studied and a new method is proposed. The main idea is the same that previous ones: it is a transformation of the objective function in order to convexify it. For that, function $g(x)$ is reformulated into $g'_{\beta,\lambda}(x) = x^t Q x + c^t x + \beta(Ax-b)^t(Ax-b) + \lambda \sum_{i=1}^{n} (x_i^2 - x_i)$. It is proved that an optimal pair (β^*, λ^*) can be obtained with $\lambda^* = -\lambda_{\min}(Q + \beta^* A^t A)$ and it is shown how to compute the pair $(\beta^*, \lambda_{\min}(Q + \beta^* A^t A))$ by semidefinite programming (see [28]). Problem (QP) is therefore reformulated as follows :

$$(QP'_{\beta^*,\lambda^*}) : \quad \operatorname*{Min}_{x \in X} \left\{ x^t P x + \beta^* \sum_{k=1}^{m} \left(\sum_{j=1}^{n} a_{kj} x_j - b_k \right)^2 + \lambda^* \sum_{i=1}^{n} \left(x_i^2 - x_i \right) \right\}$$

Computational results and comparisons regarding this last method with the one of Hammer and Rubin are reported for the densest k-subgraph problem in [28]. We show in the appendix of the paper that the bound found by this method is lower than the one obtained by resolving (QP_{α^*,u^*}).

In this paper, in order to estimate the efficiency of *Algorithm 1* (Section 2), we choose to compare it with the solution of (QP) through the reformulation (QP_{α^2,u^2}). More precisely, we compare the CPU times and the quality of the bound at the root of the search tree. Note that the reformulation of (QP) into (QP_{α^2,u^2}) does not explicitely introduce the equality constraints of (QP) into the objective function contrary to *Algorithm 1*.

Before presenting computational results for the densest k-subgraph problem in Section 5, a small instance of (QP) is studied. The four previous reformulations (QP_{α^1,u^1}), (QP_{α^2,u^2}), $(QP'_{\beta^*,\lambda^*})$, (QP_{α^*,u^*}) are applied to it.

4 Example

Given the following linearly constrained 0-1 quadratic programming problem whose optimal value is -2:

$$(\Pi) \begin{cases} \text{Min} & x^tPx \\ s.t. & Ax = b \\ & x \in \{0,1\}^5 \end{cases}$$

where $P = \begin{pmatrix} 0 & -0.5 & -0.5 & -0.5 & -0.5 \\ -0.5 & 0 & 0 & -0.5 & 0 \\ -0.5 & 0 & 0 & 0 & 0 \\ -0.5 & -0.5 & 0 & 0 & -0.5 \\ -0.5 & 0 & 0 & -0.5 & 0 \end{pmatrix}$,

$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 9 & 0 & 9 \end{pmatrix}$, $b = \begin{pmatrix} 3 \\ 11 \end{pmatrix}$.

Let S be the feasible solution set of (Π).
In order to solve (Π) exactly, we use the three reformulations mentioned in Section 3 and our new formulation presented in Section 2.

Reformulation into (Π_{α^1,u^1}), the eigenvalue method :

$$(\Pi_{\alpha^1,u^1}) : \min_{x\in S} \left(x^tPx + u^1 \sum_{i=1}^{5} (x_i^2 - x_i) \right)$$

Here, $\alpha^1 = 0$ and $u^1 = -\lambda_{\min}(P) = 1.34$.

$$(\Pi_{\alpha^1,u^1}) : \quad \underset{x\in S}{\text{Min}} \left(x^t P x + 1.34 \sum_{i=1}^{5} (x_i^2 - x_i) \right)$$

and the value of the continuous relaxation is -3.978

Reformulation into (Π_{α^2,u^2}) :

$$(\Pi_{\alpha^2,u^2}) : \quad \underset{x\in S}{Min} \left(x^t P x + \sum_{i=1}^{5} u_i^2 (x_i^2 - x_i) \right)$$

with $\alpha^2 = O$ and we find $u^2 = \begin{pmatrix} 1.98 \\ 0.90 \\ 0.64 \\ 1.62 \\ 0.90 \end{pmatrix}$ and the value of the continuous relaxation is -3.79.

Reformulation into $(\Pi'_{\beta^*,\lambda^*})$:

$$(\Pi'_{\beta^*,\lambda^*}) : \quad \underset{x\in S}{\text{Min}} \left\{ x^t P x + \beta^* \sum_{k=1}^{m} \left(\sum_{j=1}^{n} a_{kj} x_j - b_k \right)^2 + \lambda^* \sum_{i=1}^{n} \left(x_i^2 - x_i \right) \right\}$$

We find $\beta^* = 219.37$, $\lambda^* = 0.12$ $(= -\lambda_{\min}(P + 219.37 A^t A))$ and the value of the continuous relaxation is -2.92.

Reformulation into (Π_{α^*,u^*}), the method of this paper :

$$(\Pi_{\alpha^*,u^*}) : \quad \underset{x\in S}{\text{Min}} \left\{ x^t P x + \sum_{i=1}^{5} \sum_{k=1}^{2} \alpha_{ki}^* x_i \left(\sum_{j=1}^{5} a_{kj} x_j - b_k \right) + \sum_{i=1}^{5} u_i^* \left(x_i^2 - x_i \right) \right\}$$

We find $\alpha^* = \begin{pmatrix} 0.9 & 0.86 & -0.62 & 1.64 & 0.36 \\ -0.14 & -0.08 & 0.62 & -0.21 & -0.39 \end{pmatrix}$, $u^* = \begin{pmatrix} -0.1 \\ 0.23 \\ -4 \\ -0.84 \\ 4 \end{pmatrix}$ and the value of the continuous relaxation $= -2.41$.

The following table sums up the results obtained by the four reformulations :

	bound	GAP
(Π_{α^1,u^1})	-3.978	98%
(Π_{α^2,u^2})	-3.79	46%
$(\Pi'_{\beta^*,\lambda^*})$	-2.92	89%
(Π_{α^*,u^*})	$-$ **2.41**	**20%**

where $GAP = \mid \frac{opt - bound}{opt} \mid$ and $opt = -2$.

5 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 belonging to $\{3, ..., n-2\}$, 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):

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

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.* It can be also considered as a special case of the *k-dispersion-sum problem* ([27]).

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

$$(DS'): \text{ Min} \left\{ f(x) = x^t M x : \sum_{j=1}^{n} x_j = k, \; x \in \{0,1\}^n \right\}$$

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 *Algorithm 1* to problem (DS'). This amounts to first solve the following semidefinite program:

$$
(SDDS')\left\{\begin{array}{lll}
\text{Min} & \sum_{i=1}^{n}\sum_{j=1}^{n} m_{ij}X_{ij} & \\
\text{s.t.} & X_{ii} = x_i \qquad i = 1,\dots,n & (4)\\
 & -kx_i + \sum_{j=1}^{n} X_{ij} = 0 \qquad i = 1,\dots,n & (5)\\
 & \sum_{j=1}^{n} x_j = k & \\
 & \begin{pmatrix} 1 & x^t \\ x & X \end{pmatrix} \succeq 0 & \\
 & x \in \mathbb{R}^n,\ X \in \mathbb{R}^{n\times n} &
\end{array}\right.
$$

and then take the values u^* and α^* of the optimal dual variables associated to (4) and (5) in order to build problem (DS'_{α^*,u^*}) that is equivalent to (DS').

$$
(DS'_{\alpha^*,u^*}):\quad \text{Min}\ \{f_{\alpha^*,u^*}(x) = x^tMx + \sum_{i=1}^{n} u_i^*(x_i^2 - x_i) + \sum_{i=1}^{n} \alpha_i^* x_i \left(\sum_{j=1}^{n} x_j - k\right) : \sum_{j=1}^{n} x_j = k,\ x \in \{0,1\}^n\}.
$$

The next step is to submit (DS'_{α^*,u^*}) to an MIQP solver. We know that, at the root of the Branch & Bound algorithm, the bound obtained by continuous relaxation is equal to the optimal value of $(SDDS')$. For comparison purposes, we have also implemented *Algorithm 2* as an alternative to *Algorithm 1*. *Algorithm 2* is based on the reformulation (Q_{α^2,u^2}) described in Section 3. Here, α^2 is a null vector and u^2 can be computed by dropping constraints (5) in problem $(SDDS')$.

We choose to solve semidefinite programs using *SB* ([19], [21]), a software applying the spectral bundle method on eigenvalue optimization problems. All the experiments have been carried out on a Pentium IV 2.2 GHz computer with 1 Go of RAM. For each instance, the execution is stopped after 3600 seconds.

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 [5] for $n = 40$ and in [28] 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.

The following tables (1 to 9) present complete results for all graphs.

Legend of the tables :

- d, density of the graph
- *opt*, value of the optimal or best known solution
- CPU, total CPU time required by $CPLEX9$ to solve (QP_{α^*,u^*}) (*Algorithm 1*) or (QP_{α^2,u^2}) (*Algorithm 2*).
- *bound*, bound at the root of the Branch & Bound tree, i.e. optimal value of the continuous relaxation of (QP_{α^*,u^*}) or (QP_{α^2,u^2})
- $GAP = \frac{bound - opt}{opt} * 100$
- *nodes*, number of nodes in the search tree

Table 1: Comparison of Algorithm 1 and Algorithm 2 for $n = 40$ and $k = 10$

			Algo 1				*Algo 2*			
d	num	*opt*	*CPU*	*bound*	GAP	nodes	*CPU*	*bound*	*GAP*	nodes
0.25	1	29	0.04	30.63	5.6	43	15.33	49.9	72	88848
	2	30	0.07	32.08	6.9	150	106.16	56.41	88	579430
	3	27	0.36	30.42	12.6	1231	264.95	53.18	96	1361551
	4	27	0.16	29.95	10.9	508	59.06	49.41	83	337578
	5	26	0.18	29.12	12	656	76.22	47.92	84	431958
0.5	1	40	0.19	43.46	8.6	645	-	97.16	142.9	13733723
	2	41	0.15	44.38	8.2	459	-	105.99	158.5	18043481
	3	39	0.92	43.54	11.6	3765	-	103.67	165.8	18463871
	4	40	0.28	43.64	9.1	999	-	100.25	150.6	19319151
	5	40	0.23	43.65	9.1	820	-	97.96	144.9	16771867
0.75	1	45	2.58	50.69	12.6	12996	-	145.02	222.2	20629251
	2	45	5.25	50.77	12.8	31820	-	147.05	226.7	20569711
	3	45	4.15	50.36	11.9	21951	-	148.03	228.9	21253751
	4	45	3.39	50.27	11.7	18557	-	146.92	226.5	20678313
	5	45	4.41	50.58	12.4	27111	-	145.86	224.1	20828406

- : the corresponding instance could not be solved within 3600 seconds

Table 2: Comparison of Algorithm 1 and Algorithm 2 for $n = 40$ and $k = 20$

				Algo 1				*Algo 2*		
d	num	*opt*	*CPU*	*bound*	GAP	nodes	*CPU*	*bound*	*GAP*	nodes
0.25	1	77	0.08	79.44	3.2	212	29.57	97.55	25.7	171754
	2	84	0.06	86.08	2.5	110	127.66	111.61	32.8	631749
	3	76	0.18	79.15	4.1	526	1173.93	105.36	38.6	4353901
	4	75	0.04	76.84	2.4	54	53.07	97.46	29.9	304227
	5	73	0.08	75.28	3.1	173	59.12	94.54	29.5	325885
0.5	1	130	0.03	131.48	1.1	32	-	193.17	48.66	18387481
	2	130	0.62	134.53	3.5	2500	-	211.21	62.4	21678192
	3	133	0.12	136.16	2.4	349	-	206.23	55	19876820
	4	128	0.21	131.75	2.9	728	-	199.38	55.8	20606526
	5	129	0.13	132.37	2.6	435	-	194.78	50.9	20366412
0.75	1	168	0.1	170.80	1.7	249	-	289.54	72.3	22372032
	2	173	0.06	175.15	1.2	111	-	293.43	69.6	22551071
	3	171	0.13	173.61	1.5	370	-	295.42	72.8	21920075
	4	171	0.09	173.43	1.4	251	-	293.23	71.4	22343940
	5	171	0.13	173.86	1.7	419	-	290.97	70	22470031

- : the corresponding instance could not be solved within 3600 seconds

Table 3: Comparison of Algorithm 1 and Algorithm 2 for $n = 40$ and $k = 30$

			Algo 1				*Algo 2*			
d	num	*opt*	*CPU*	*bound*	GAP	nodes	*CPU*	*bound*	*GAP*	nodes
0.25	1	134	0.03	135.11	0.8	3	0.15	142.52	6.3	954
	2	148	0.03	149.66	1.1	34	7.62	165.54	11.8	51514
	3	138	0.06	140.57	1.9	207	15.53	156.51	13.4	108802
	4	134	0.03	135.20	0.9	8	0.22	143.76	7.2	1438
	5	130	0	130.97	0.7	0	0.31	139.65	7.4	1953
0.5	1	247	0.03	248.34	0.5	22	145.47	287.90	16.5	714591
	2	263	0.03	264.86	0.7	43	1865.84	315.59	19.9	3444769
	3	261	0.05	263.48	0.9	92	243.79	307.69	17.8	1177113
	4	254	0.05	256.63	1	99	148.86	297.19	17	842617
	5	250	0.04	251.96	0.8	41	93.9	290.27	16.1	568437
0.75	1	349	0.03	350.67	0.5	33	-	433.54	24.2	18757433
	2	357	0.04	359.02	0.6	49	-	439.10	22.9	13360731
	3	358	0.03	359.39	0.4	23	-	442.1	23.5	16029881
	4	354	0.03	355.58	0.4	41	-	438.93	23.9	17658571
	5	355	0.04	356.93	0.5	41	-	435.36	22.6	8502981

- : the corresponding instance could not be solved within 3600 seconds

Table 4: Results of Algorithm 1 for $n = 80$ and $k = 20$

d	num	opt	CPU	$bound$	GAP	nodes
0.25	1	94	116.46	102.71	9.2	190016
	2	93	125.15	102.26	9.9	203159
	3	100	8.21	107.11	7.1	13739
	4	96	80.55	105.12	9.5	129227
	5	97	301.94	106.84	10.1	501668
0.5	1	149	82.25	160.34	7.6	129556
	2	148	110.66	158.92	7.3	181519
	3	144	984.63	157.99	9.7	1592333
	4	144	362.6	155.28	7.8	592656
	5	149	86.72	160.36	7.6	140435
0.75	1	182	1240.39	193.63	6.4	2112340
	2	182	3236.95	194.99	7.1	5291081
	3	184	1028.55	196.80	6.9	1662192
	4	183	905.86	194.36	6.2	1481003
	5	183	202.34	193.30	5.6	305878

Table 5: Results of Algorithm 1 for $n = 80$ and $k = 40$

d	num	opt	CPU	$bound$	GAP	nodes
0.25	1	280	12.21	287.09	3.9	20379
	2	273	51.77	281.32	3	82008
	3	285	9.48	292.17	2.5	15982
	4	284	12.27	291.85	2.8	18736
	5	292	12.17	299.52	2.6	20104
0.5	1	488	22.33	497.58	2	37078
	2	489	27.21	497.53	1.7	45882
	3	484	27.26	493.19	1.9	41950
	4	479	26.57	487.25	1.7	40501
	5	495	11.31	503.54	1.7	17308
0.75	1	671	10.09	677.87	1	15691
	2	668	51.42	676.83	1.3	82276
	3	665	596.7	676.63	1.7	924400
	4	669	67.51	677.37	1.2	107320
	5	657	158.21	666.33	1.4	251302

Table 6: Results of Algorithm 1 for $n = 80$ and $k = 60$

d	num	opt	CPU	$bound$	GAP	nodes
0.25	1	518	1.09	523.07	0.9	2080
	2	512	0.8	516.71	0.9	1462
	3	523	0.43	526.81	0.7	587
	4	528	1.88	533.57	1.1	3763
	5	543	0.38	546.7	0.7	605
0.5	1	968	1.26	974.26	6.5	2583
	2	983	0.31	987.11	0.4	357
	3	972	6.19	979.72	0.8	13097
	4	972	0.39	976.14	0.4	597
	5	989	2.29	996.28	0.7	5156
0.75	1	1413	7.02	1420.53	0.5	15638
	2	1410	1.36	1415.44	0.4	2728
	3	1417	7.94	1424.40	0.5	16644
	4	1416	0.61	1420.38	0.3	1040
	5	1394	1.06	1398.94	0.4	1912

Table 7: Results of Algorithm 1 for $n = 100$ and $k = 25$

d	num	opt	CPU	$bound$	GAP	nodes
0.25	1	139	-	151.58	9	3879506
	2	138	-	153.46	11.2	3977611
	3	142	1645.62	153.99	8	1713934
	4	140	945.8	152.02	8.6	979105
	5	145	1823.4	157.66	8.7	1948478
0.5	1	218	-	236.97	8.7	3874652
	2	221	-	241.23	9	3798921
	3	216	-	232.92	7.8	4005381
	4	221	1977.64	236.68	7	2026697
	5	221	-	236.28	6.9	3846491
0.75	1	282	-	297.81	5.6	3987021
	2	282	-	298.92	6	4200221
	3	285	-	301.41	5.7	3908591
	4	288	-	303.43	5.3	3983371
	5	286	-	301.38	5.4	4105875

- : the corresponding instance could not be solved within 3600 seconds

Table 8: Results of Algorithm 1 for $n = 100$ and $k = 50$

d	num	opt	CPU	$bound$	GAP	nodes
0.25	1	417	1276.47	428.54	2.8	1385156
	2	417	443.77	429.01	2.9	464713
	3	429	92.18	437.90	2.07	102571
	4	413	182.61	423.03	2.4	191163
	5	435	371.84	446.43	2.6	379615
0.5	1	729	-	745.07	2.2	3983441
	2	740	-	757.57	2.4	3916567
	3	729	391.51	740.64	1.6	396780
	4	729	-	747.09	2.5	3950406
	5	733	-	749.10	2.2	4002221
0.75	1	1029	225.7	1039.01	0.9	250625
	2	1035	942.16	1046.58	1.1	1006513
	3	1047	874.15	1058.77	1.1	888108
	4	1055	470.57	1066.04	1	512979
	5	1049	229.1	1058.75	0.9	244333

- : the corresponding instance could not be solved within 3600 seconds

Table 9: Results of Algorithm 1 for $n = 100$ and $k = 75$

d	num	opt	CPU	$bound$	GAP	nodes
0.25	1	796	2.03	801.97	0.7	2564
	2	787	4.3	793.48	0.8	6029
	3	809	3.49	815.01	0.7	5167
	4	776	29.53	783.88	1	43107
	5	828	5.64	834.69	0.8	8115
0.5	1	1483	162.79	1494.76	0.8	240649
	2	1518	42.77	1529.28	0.7	60137
	3	1476	6.48	1483.33	0.5	9226
	4	1487	34.4	1497.96	0.7	50457
	5	1495	7.4	1502.75	0.5	10512
0.75	1	2175	14.11	2182.72	0.35	22351
	2	2192	25.11	2200.64	0.36	36487
	3	2230	17.32	2237.96	0.36	25226
	4	2235	4.7	2242.07	0.3	7066
	5	2229	10.37	2235.86	0.3	14093

The running times corresponding to the computation of (α^*, u^*) (*Algorithm 1*) and of u^2 (*Algorithm 2*) by the semidefinite programming solver *SB* are not reported in the tables. They 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^*}) or (QP_{α^2,u^2}) is very fast too.

First, consider results of graphs with $n = 40$ (Tables 1, 2 and 3) where *Algorithm 1* and *Algorithm 2* are compared : the gap of *Algorithm 1* is about 20 times smaller than the gap of *Algorithm 2*. The CPU time is hence drastically improved. In sight of these results, we do not compare *Algorithm 1* and *Algorithm 2* with more than 40 nodes.

For $n = 40$ the optimal solution is always found by *Algorithm 1* in less than one second of CPU time, except for $d = 0.75$. In this case, the required CPU time is less than 5 seconds. Note that *Algorithm 2* solves only 20 out of 45 instances in one hour of CPU time.

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

For $n = 100$ and $k = \frac{3n}{4}$, all the instances are solved in less than one hour of CPU time. It is also the case for $k = \frac{n}{2}$ when the density is 0.25 or 0.75. In these experimental conditions, the most difficult instances for the approach seem to be those with $k = \frac{n}{4}$. Note that, in this case and for $n = 100$, 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%.

Instances with 40 and 80 nodes have already been used in [28] : the eigenvalue method and the reformulation $(QP'_{\beta^*,\lambda^*})$ have been applied to solve (QP). Note that for all instances, our new method improves the value of the bound and the CPU time of [28]: it is about 2 times smaller. In addition, the method exposed in Section 2 allows to solve all instances with $n = 80$ whereas with the reformulation $(QP'_{\beta^*,\lambda^*})$, when $k = 20$, only 3 out of 15 instances are solved in less than one hour.

Moreover, we can solve larger problems than recent publications which use same instances but different approaches. In [27], the reported results concern instances with no more than 60 nodes. The integer linear programming approach presented in [5] and based on six different formulations doesn't allow to solve instances with 80 nodes, except in the case of the 75% density instances.

6 Conclusion

In the paper, we have considered the problem of minimizing a quadratic 0-1 function $g(x)$ subject to linear constraints $Ax = b$ and $A'x \leq b'$. 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. The new function $g_{\alpha,u}(x)$ is obtained by adding to $g(x)$ the two following functions, null on the feasible set, $g_1(x) = \sum_{i=1}^{n} u_i \left(x_i^2 - x_i\right)$ and $g_2(x) = \sum_{i=1}^{n}\sum_{k=1}^{m} \alpha_{ki} x_i \left(\sum_{j=1}^{n} a_{kj} x_j - b_k\right)$. The n-vector u and the $m \times n$-matrix α are determined in order to make $g_{\alpha,u}(x)$ convex and to maximize its value over the relaxed domain $\overline{X} = \{Ax = b, A'x \leq b', x \in [0,1]^n\}$. These optimal values of u and α are determined by semidefinite programming. We applied this approach to a difficult combinatorial optimization problem, the densest k-subgraph problem. The 0-1 optimization of $g_{\alpha,u}(x)$, subject to the cardinality constraint, was carried out by $CPLEX9$. The 45 instances considered with 80 vertices and different values of k (20, 40, 60) and density (0.25, 0.50, 0.75) are solved in less than one hour of CPU time. Moreover, 30 instances over 45 with 100 vertices are also solved within one hour of CPU time. To the best of our knowledge, such results have not been reported in the literature to date. This approach drastically improves the simpler method consisting in minimizing $g_{\alpha,u}(x)$ with $\alpha = 0$, u being determined, as previously, in order to make $g_{\alpha,u}(x)$ convex and to maximize its value over $\overline{X}$. As a conclusion of this work, introducing the $\alpha-$parametrized part $g_2(x)$ in the convexification of the objective function $g(x)$ may be particularly efficient. This $\alpha-$parametrized part uses the equality constraints and is a general way for building null functions over the associated feasible set. Finally, the approach can be applied to many combinatorial optimization problems.

References

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

[2] 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.

[3] 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.

[4] 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.

[5] A. Billionnet. Different formulations for solving the heaviest k-subgraph problem. *Technical Report CEDRIC 384*, http://cedric.cnam.fr/PUBLIS/RC384.pdf, 2002.

[6] A. Billionnet and S. Elloumi. Using a mixed integer quadratic programming solver for the unconstrained quadratic 0-1 problem. *Technical Report CEDRIC 466*, http://cedric.cnam.fr/PUBLIS/RC466.pdf, 2003.

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

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

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

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

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

[12] A. Faye and F. Roupin. Partial lagrangian and semidefinite relaxations of quadratics programs. *In 6ème congrès ROADEF, Tours, 14-16 février*, 2005.

[13] 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.

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

[15] 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.

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

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

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

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

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

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

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

[23] 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.

[24] 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, Kluwer*, pages 119–134, 2001.

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

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

[27] D. Pisinger. Upper bounds and exact algorithm for p-dispersion problems. *To appear in Computers and Operations Research.*

[28] 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.

[29] 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.

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

[31] N.Z. Shor. Quadratic optimization problems. *Soviet Journal of computer Systems Sciences*, 25:1–11, 1987.

Appendix

Proposition 1. *Consider problems (QP_{α^*,u^*}) and $(QP'_{\beta^*,\lambda^*})$ respectively defined in Section 2 and Section 3. The solution (α^*, u^*) is obtained by solving the following problem :*

$$(3):\ \underset{\substack{\alpha\in\mathbb{R}^{n\times m}\\ u\in\mathbb{R}^n\\ Q_{\alpha,u}\succeq 0}}{Max}\left\{L_1(\alpha,u)=\underset{\substack{x:Ax=b\\ A'x\le b'\\ 0\le x\le 1}}{Min}\left\{\Phi^1_{\alpha,u}(x)=g(x)+\sum_{i=1}^{n}\sum_{k=1}^{m}\alpha_{ki}x_i\left(\sum_{j=1}^{n}a_{kj}x_j-b_k\right)+\sum_{i=1}^{n}u_i\left(x_i^2-x_i\right)\right\}\right\}$$

and the solution (β^*, λ^*) is obtained by solving the following one :

$$(4):\ \underset{\substack{\beta,\lambda\in\mathbb{R}^2\\ Q_{\beta,\lambda}\succeq 0}}{Max}\left\{L_2(\beta,\lambda)=\underset{\substack{x:Ax=b\\ A'x\le b'\\ 0\le x\le 1}}{Min}\left\{\Phi^2_{\beta,\lambda}(x)=g(x)+\beta\sum_{k=1}^{m}\left(\sum_{j=1}^{n}a_{kj}x_j-b_k\right)^2+\lambda\sum_{i=1}^{n}\left(x_i^2-x_i\right)\right\}\right\}$$

The optimal value of (3) *is higher or equal to the one of* (4)*. As a consequence, (QP_{α^*,u^*}) has a better relaxation than $(QP'_{\beta^*,\lambda^*})$.*

Proof. First, note that $\Phi^2_{\beta,\lambda}(x)$ can be rewritten as follows :

$$\Phi^2_{\beta,\lambda}(x)=g(x)+\sum_{k=1}^{m}\sum_{i=1}^{n}\beta\, a_{ki}x_i\sum_{j=1}^{n}\left(a_{kj}x_j-b_k\right)-\beta\sum_{k=1}^{m}b_k\left(\sum_{j=1}^{n}a_{kj}x_j-b_k\right)+\lambda\sum_{i=1}^{n}\left(x_i^2-x_i\right)$$

Let (β, λ) be a feasible solution to (4).
Let $(\tilde{\alpha}, \tilde{u})$ such that $\tilde{\alpha}_{ki} = \beta a_{ki}$ and $\tilde{u}_i = \lambda$.
$(\tilde{\alpha}, \tilde{u})$ is a feasible solution to (3) since $\Phi^1_{\tilde{\alpha},\tilde{u}}(x)$ and $\Phi^2_{\beta,\lambda}(x)$ have precisely the same hessian matrix.
Moreover, observe that, for all x such that $Ax = b$,

$$\Phi^2_{\beta,\lambda}(x) = \Phi^1_{\tilde{\alpha},\tilde{u}}(x) = g(x) + \lambda \sum_{i=1}^{n} \left(x_i^2 - x_i\right)$$

This shows that $L_1(\tilde{\alpha}, \tilde{u}) = L_2(\beta, \lambda)$. $\square$