Eigenvalue Methods for Linearly Constrained Quadratic 0-1 Problems with Application to the Densest k-Subgraph Problem

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

[1] Laboratoire CEDRIC, Conservatoire National des Arts et Métiers,
292 rue Saint Martin, F-75141 Paris
mc.plateau@cnam.fr

[2] Laboratoire CEDRIC, Institut d'Informatique d'Entreprise,
18 allée Jean Rostand, F-91025 Evry
billionnet@iie.cnam.fr

[3] Laboratoire CEDRIC, Conservatoire National des Arts et Métiers,
292 rue Saint Martin, F-75141 Paris
elloumi@cnam.fr

Résumé Given problem (QP) of minimizing a quadratic 0-1 function under a set of linear equality constraints, we adopt the following general framework for an exact solution method. First, transform (QP) into an equivalent problem (QP'). Problem (QP') has an additional property : its objective function is convex. Second, solve (QP') by use of convex quadratic relaxation and Branch & Bound. This framework was already suggested by Hammer and Rubin ([4]) where the transformation of (QP) into (QP') was based on the computation of a smallest eigenvalue. In this paper, we show that the method by Hammer and Rubin can be embedded within a family of transformations depending on a scalar parameter. Computing the parameter leading to the best transformation can be done efficiently by solving a semidefinite programming problem. In order to have a first computational experience of this general method, we use it to solve the densest k-subgraph problem. We show that the method by Hammer and Rubin is drastically improved. Our method is even largely competitive with state-of-the-art methods.

Mots-Clefs. exact solution ; convex 0-1 quadratic relaxation ; experiments

1 Introduction

This paper is concerned with the following linearly-constrained zero-one quadratic programming problem :

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

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 and that $Ax = b$ has more than one solution.

Quadratic zero-one programming with linear equality constraints is a general model that allows to formulate numerous important problems in combinatorial optimization including, for example, the following ones : quadratic assignment, graph partitioning, task allocation, capital budgeting and heaviest k-subgraph problem.

Three important classes of solution procedures for problem (QP) are available : (1) solve the problem directly, (2) transform problem (QP) into an equivalent linear mixed-integer linear program, and then solve the obtained problem, and (3) transform (QP) into an equivalent quadratic zero-one problem having a positive semidefinite matrix in the objective function and then solve the obtained problem. In this paper we consider the third approach. Its interest is enhanced by the fact that the new versions of standard commercially available software are now able to solve mixed integer quadratic programs that have a convex quadratic objective function and a set of linear constraints (MIQP).

Hammer and Rubin have already proposed in [4] to convexify the objective function of a linearly constrained quadratic 0-1 program without using the constraints. Our main contribution is to improve the method by taking into account the equality constraints of the program in the convexification phase.

Under the constraints $Ax = b$, for any scalar a, $a(Ax-b)^t(Ax-b) = 0$. A problem (QP_a), equivalent to (QP), is therefore obtained by adding $a(Ax-b)^t(Ax-b)$ to the objective function :

$$\text{Min}\left\{q'_a(x) = x^tQx + a(Ax-b)^t(Ax-b) : Ax = b, x \in \{0,1\}^n\right\}$$

This problem can be written

$$\text{Min}\left\{q'_a(x) = x^tQ_ax + c_a^tx + l_a : Ax = b, x \in \{0,1\}^n\right\}$$

by adequately defining Q_a, c_a and l_a. Then (QP_a) is transformed into another equivalent problem with a convex objective function by using the fact that $x_i^2 = x_i$ for $x_i \in \{0,1\}$. So we add to $q'_a(x)$ the quantity $-\lambda_{\min}(Q_a)\sum_{i=1}^n \left(x_i^2 - x_i\right)$ where $\lambda_{\min}(Q_a)$ is the smallest eigenvalue of matrix Q_a. Finally, we obtain a new problem

$$(QP_a) : \text{Min}\left\{q_a(x) = q'_a(x) - \lambda_{\min}(Q_a)\sum_{i=1}^n (x_i^2 - x_i) : Ax = b, x \in \{0,1\}^n\right\}$$

which is equivalent to (QP) and can be handled by an MIQP solver. The efficiency of the approach strongly depends on the coefficient a. We show in this paper how to choose a, and consequently $\lambda_{\min}(Q_a)$, in order to maximize the continuous relaxation of (QP_a).

Computational experiments regarding this method are reported for a graph theory problem, the unweighted version of the heaviest k-subgraph problem, the so-called densest k-subgraph problem ([1], [10], [11]) which can be easily formulated by (QP). 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. Computational results show that the proposed method outperforms the methods presented in the literature for this graph problem since it allows to obtain exact solutions of most of the instances with up to 80 nodes with k equal to $\frac{n}{2}$ and $\frac{3n}{4}$, regardless of their density.

The rest of the paper is organized as follows. In Section 2, we recall the method proposed in [4]. In Section 3, we show how this method can be improved by transforming the objective function. Then, a computational comparison for the densest k-subgraph problem follows in Section 4, and Section 5 gives a conclusion.

2 The smallest eigenvalue method [4]

For any scalar $\lambda \in \mathbb{R}$, let us associate with $q(x)$ the perturbed function :

$$
\begin{aligned}
q^\lambda(x) &= q(x) + \lambda \textstyle\sum_{i=1}^n (x_i^2 - x_i) \\
&= x^t(Q + Diag(\lambda))x - \lambda \textstyle\sum_{i=1}^n x_i
\end{aligned}
$$

where $Diag(\lambda)$ is the diagonal matrix whose all diagonal terms are equal to λ. Note that Q is not a positive semidefinite matrix because all diagonal terms of Q are equal to 0.

For all $x \in \{0,1\}^n$, $q^\lambda(x) = q(x)$. So, we can solve (QP) by solving the equivalent problem :

$$(QP^\lambda) \quad \text{Min}\left\{q^\lambda(x) : Ax = b, x \in \{0,1\}^n\right\}$$

If $(Q + Diag(\lambda)) \succeq 0$, the continuous relaxation of (QP^λ) can be solved in polynomial time and gives a lower bound :

$$\mu(\lambda) = \text{Min}\left\{q^\lambda(x) : Ax = b, x \in [0,1]^n\right\}$$

Our aim is to find the best (largest) lower bound, i.e. :

$$\mu^* = \text{Max}\left\{\mu(\lambda) : \lambda \geq -\lambda_{\min}(Q),\ \lambda \in \mathbb{R}\right\}$$

Note that $\lambda_{\min}(Q)$ is a real negative number and that $(Q + Diag(\lambda)) \succeq 0$ and $q^\lambda(x)$ is convex if and only if $\lambda \geq -\lambda_{\min}(Q)$.

Proposition 1. *Let* λ_1 *and* λ_2 *be such that* $(Q + Diag(\lambda_1)) \succeq 0$ *and* $(Q + Diag(\lambda_2)) \succeq 0$. *If* $\lambda_1 \geq \lambda_2$ *then* $\mu(\lambda_1) \leq \mu(\lambda_2)$.

So, in order to maximize $\mu(\lambda)$, λ should be as small as possible and we obtain Corollary 1 :

Corollary 1.

$$\mu^* = \mu\left(-\lambda_{min}(Q)\right) = Min\left\{q^{-\lambda_{min}(Q)}(x) : Ax = b, x \in [0,1]^n\right\}.$$

According to Proposition 1 and Corollary 1, a resolution algorithm can be elaborated to solve (QP) :

Algorithm 1 :
1. Compute the smallest eigenvalue of $Q : \lambda_{min}(Q)$
2. Solve the quadratic problem $\left(QP^{-\lambda_{min}(Q)}\right)$ *whose continuous relaxation is convex.*

The smallest eigenvalue method is a way to solve linearly constrained non-convex 0-1 quadratic problems and permits to quickly compute a lower bound. This algorithm is used in [2] for the unconstrained case. Computational results are reported in this reference.

Our aim, in the following of this paper, is to improve this lower bound that means to find a largest one and, in the same time, to find efficiently an exact solution.

3 Improving the smallest eigenvalue method by using the constraints

This section details our new approach : an adequate transformation of the objective function by addition of a zero function with the aim of obtaining an efficient lower bound. This idea of adding a zero function is used in [3] to improve linearization based methods to solve (QP).

Let a be a real number and

$$\begin{aligned} q'_a(x) &= q(x) + a\,(Ax-b)^t\,(Ax-b) \\ &= x^t Q_a x + c_a^t x + l_a \end{aligned}$$

where $Q_a = Q + aA^tA$, $c_a = c - 2aA^tb$ and $l_a = ab^tb$. Note that $q'_a(x) = q(x)$ for all x such that $Ax = b$.

First, our idea is to add the zero function $a\,(Ax-b)^t\,(Ax-b)$ to the objective function of (QP) and to apply the smallest eigenvalue method (cf. Section 2) in order to convexify the problem. So, we add $-\lambda_{\min}(Q_a)\sum_{i=1}^n(x_i^2-x_i)$ to $q'_a(x)$ and we define a new problem (QP_a), equivalent to (QP) :

$$(QP_a):\ \text{Min}\left\{q_a(x)=q'_a(x)-\lambda_{\min}(Q_a)\sum_{i=1}^n(x_i^2-x_i):Ax=b,\ x\in\{0,1\}^n\right\}$$

Let $\beta(a)$ be the lower bound of the optimal value of (QP_a) obtained by continuous relaxation, i.e. :

$$\beta(a)=\text{Min}\left\{q_a(x):Ax=b,\ x\in[0,1]^n\right\}$$

Note that $\beta(0)=\mu^*$ (cf. Section 2). Our approach is to adjust the parameter $a\in\mathbb{R}$ in order to obtain the highest value of $\beta(a)$. We will show that $\beta(a)$ is a nondecreasing function of a and then $\beta(a)\geq\mu^*,\quad\forall a>0$.

Proposition 2. *If* $a_1\geq a_2$ *then* $\lambda_{min}(Q_{a_1})\geq\lambda_{min}(Q_{a_2})$.

Proof. Recall the Rayleigh expression of the extreme eigenvalues $\lambda_{min}(Q_{a_1})=\min\limits_{x:\|x\|=1}x^tQ_{a_1}x$ and $\lambda_{min}(Q_{a_2})=\min\limits_{x:\|x\|=1}x^tQ_{a_2}x$. Now, for any $x\in\mathbb{R}^n$, $x^tQ_{a_1}x-x^tQ_{a_2}x=(a_1-a_2)x^tA^tAx\geq0$ because $A^tA\succeq0$, which yields $\lambda_{min}(Q_{a_1})\geq\lambda_{min}(Q_{a_2})$. ■

Proposition 3. *Let* a_1 *and* a_2 *such that* $\lambda_{min}(Q_{a_1})\geq\lambda_{min}(Q_{a_2})$ *then* $\beta(a_1)\geq\beta(a_2)$.

Proof. Let $x\in[0,1]^n$ such that $Ax=b$.

$$\begin{aligned}q_{a_1}(x)-q_{a_2}(x)&=q(x)+a_1\left(Ax-b\right)^t\left(Ax-b\right)-\lambda_{\min}(Q_{a_1})\textstyle\sum_{i=1}^n(x_i^2-x_i)\\&\quad-q(x)-a_2\left(Ax-b\right)^t\left(Ax-b\right)+\lambda_{\min}(Q_{a_2})\textstyle\sum_{i=1}^n(x_i^2-x_i)\\&=\left(\lambda_{\min}(Q_{a_2})-\lambda_{\min}(Q_{a_1})\right)\textstyle\sum_{i=1}^n(x_i^2-x_i)\geq0\end{aligned}$$

So, for all feasible solutions, $q_{a_1}(x)\geq q_{a_2}(x)$. Consequently, $\beta(a_1)\geq\beta(a_2)$. ■

Corollary 2. *If* $a_1\geq a_2$ *then* $\beta(a_1)\geq\beta(a_2)$.

According to Corollary 2, larger a is fixed, better is the bound. But, we will show that the value of $\beta(a)$ is bounded.

By Proposition 2 and Proposition 3, maximizing $\beta(a)$ is equivalent to the following problem :

$$\text{Max}\left\{\lambda_{\min}(Q_a):a\in\mathbb{R}\right\}\qquad(1)$$

It is well-known (see for example [7]) that this problem can be stated as a semidefinite program which dual is

$$(SDP)\begin{cases}\text{Minimize} < Q, X > & & \\ \text{s.t.} \qquad tr(X) & = 1 & (2) \\ \qquad < A^tA, X > & = 0 & (3) \\ \qquad X \succeq 0 & & \end{cases}$$

where $< A, B >= tr(A^tB) = \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}B_{ij}$ for all A and B in S_n, which is the set of $n \times n$ symmetric real matrices.

The following Proposition shows that (SDP) is feasible since (QP) is feasible. Then the dual problem (1) is bounded.

Proposition 4. *If (QP) is feasible then (SDP) is feasible.*

Proof. Let x such that $x \neq 0$ be a solution of $Ax = 0$ (because there exists several solutions of $Ax = b$). Let us define v such that $v = \frac{x}{\|x\|}$. So $Av = 0$. Let $V = v^tv$. Consequently, $tr(V) = 1$ and V is positive semidefinite. V is a feasible solution of (SDP). ■

The previous results permit to elaborate a new resolution algorithm :

Algorithm 2 :

1. Solve the semidefinite program (SDP), let a^* be the dual variable value associated to constraint (3) of (SDP) and $\lambda_{min}(Q_{a^*})$ be its optimal value.

2. Solve the quadratic problem (PQ_{a^*}) whose continuous relaxation is convex.

Observe that a^* of step *1* could be determined by iteratively computing eigenvalues of Q_a for an increasing series of a.

4 Computational results for the densest k-subgraph problem

We choose to apply our method on the densest k-subgraph problem (defined in Section 1) which can be formulated as the following linearly constrained 0-1 quadratic optimization problem :

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

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. (DSP) is also known under the name of *k-cluster problem.*

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

$$(QP): \ \text{Min}\left\{ q(x) = x^t Q x : \sum_{i=1}^{n} x_i = k, \ x \in \{0,1\}^n \right\}$$

where the general term of Q is $Q_{ij} = -\frac{1}{2}\delta_{ij}, \ \forall i,j$. The optimal value of (QP) is equal to the opposite of the (DSP) one.

According to previous sections, we can define a new problem, depending on a new parameter a :

$$(QP_a): \ \text{Min}\left\{ q_a(x) : \sum_{i=1}^{n} x_i = k, \ \ x \in \{0,1\}^n \right\}$$

where $q_a(x) = x^t Q_a x - ak^2 - \lambda_{min}(Q_a)\sum_{i=1}^{n}(x_i^2 - x_i)$ and the general term of Q_a is $(Q_a)_{ij} = -\frac{1}{2}\delta_{ij} + a, \ \forall i,j$.

It is important to note that the new versions of commercial software such as $CPLEX9$ solve exactly quadratic integer problems having a convex quadratic objective function and a set of linear constraints. Moreover, we choose to solve semidefinite programs by using SB ([6], [8]), a software applying the spectral bundle method on eigenvalue optimization problems. The minimum eigen value of matrix Q (*Algorithm 1*) is computed by using $Scilab$ ([12]). All the numerical 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.

The two methods defined by *Algorithm 1* and *Algorithm 2* are applied on randomly generated graphs. The following tables show the results for different graph sizes ($n = 40, 80$), different densities ($d = 25\%, 50\%, 75\%$) and different k values ($k = \frac{n}{4}, \frac{n}{2}, \frac{3n}{4}$). Observe that for each couple (k,d), there are 5 instances (used in [1] for $n = 40$). 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.

Tables 1, 2 and 3 (i.e. $n = 40$) present for a given density, the average values of cpu time, GAP and a^*. The corresponding standard deviation is given between brackets and the sign '-' means that no instance could be solved within 3600 seconds of cpu time. Tables 4, 5 and 6 (i.e. $n = 80$) present complete results.

Legend of the tables :
d, density of the graph
opt, value of the optimal solution
cpu, cpu time required by $CPLEX$
$\beta(0) = \mu^*$, as defined in Section 2
a^* and $\beta(a^*)$, as defined in Section 3
GAP (*Algorithm 1*) $= \frac{\beta(0) - opt}{opt} * 100$

GAP (*Algorithm 2*) $= \frac{\beta(a^*)-opt}{opt} * 100$

Tab. 1. $n = 40, k = 10$ (average and standard deviation)

	Algo 1		*Algo 2*		
d	*cpu*	GAP	*cpu*	a^*	GAP
25	264 (240)	87.4 (8.3)	0.5 (0.4)	9.8 (6.9)	17.4 (4.1)
50	-	153.6 (9.4)	1.5 (1.5)	14.4 (18.4)	16.1 (1.8)
75	-	226 (3)	47.9 (13.3)	13.6 (17.5)	20.9 (1.2)

- : no instance could be solved within 3600 seconds

Tab. 2. $n = 40, k = 20$ (average and standard deviation)

	Algo 1		*Algo 2*		
d	*cpu*	GAP	*cpu*	a^*	GAP
25	1031 (1432.3)	35.2 (3.6)	0.4 (0.4)	9.8 (6.9)	6.3 (0.8)
50	-	56.1 (5.2)	0.9 (1.5)	14.4 (18.4)	4.3 (0.9)
75	-	72 (1.3)	0.5 (0.3)	13.6 (17.5)	2.9 (0.3)

- : no instance could be solved within 3600 seconds

Tab. 3. $n = 40, k = 30$ (average and standard deviation)

	Algo 1		*Algo 2*		
d	*cpu*	GAP	*cpu*	a^*	GAP
25	14.1 (18)	13.8 (2.3)	0.1 (0.1)	9.8 (6.9)	3 (0.6)
50	-	19.4 (1.2)	0.1 (0.03)	14.4 (18.4)	1.3 (0.7)
75	-	24.2 (0.6)	0.05 (0.01)	13.6 (17.5)	1.01 (0.1)

- : no instance could be solved within 3600 seconds

Tab. 4. $n = 80, k = 20$

		Algo 1			Algo 2			
d	*opt*	*cpu*	$-\beta(0)$	GAP	*cpu*	$-\beta(a^*)$	a^*	GAP
25	94	>3600	203.16	116	>3600	109.85	4.53	16.8
	93	>3600	199.58	114	2208	107.22	5.41	15.2
	100	>3600	203.22	103	331	114.08	23.03	14.1
	96	>3600	207.91	116	1569	110.25	4.48	14.8
	97	>3600	212.5	119	>3600	111.88	2.9	15.3
50	149	>3600	394.85	165	>3600	167.71	0.33	12.5
	148	>3600	394.30	166.4	>3600	166.21	3.458	12.3
	144	>3600	398.55	176	>3600	164.95	0.724	14.5
	144	>3600	395.09	174	>3600	162.05	4.739	12.5
	149	>3600	402.85	170	>3600	169.21	5.34	13.5
75	182	>3600	594.10	226	>3600	202.188	0.545	11.1
	182	>3600	592.94	225	>3600	203.70	0.832	11.9
	183	>3600	600.12	228	>3600	205.95	1.503	12.5
	183	>3600	595.6	225	>3600	202.97	12.03	10.9
	183	>3600	590.14	222	>3600	201.59	5.159	10.1

Tab. 5. $n = 80, k = 40$

		Algo 1			Algo 2			
d	*opt*	*cpu*	$-\beta(0)$	GAP	*cpu*	$-\beta(a^*)$	a^*	GAP
25	280	>3600	406.33	45	347	293.62	4.53	4.8
	273	>3600	399.16	46.2	1975	287.63	5.41	5.3
	285	>3600	406.44	42.6	639	300.13	23.03	5.3
	284	>3600	415.83	46.4	249	296.77	4.48	4.4
	292	>3600	425.04	45.6	487	305.9	2.9	4.7
50	488	>3600	789.71	61.8	591	504.89	0.33	3.4
	489	>3600	788.62	61.3	1067	506.08	3.458	3.6
	484	>3600	797.10	64.7	691	500.02	0.724	3.3
	479	>3600	790.18	65	1295	494.99	4.739	3.3
	495	>3600	805.71	62.8	806	512.14	5.34	3.46
75	671	>3600	1188.22	77	247	683.11	0.545	1.8
	668	>3600	1185.9	77.5	1182	682.66	0.832	2.2
	665	>3600	1200.25	80.5	>3600	685.54	1.503	3
	669	>3600	1191.20	78.1	2376	683.86	12.03	2.2
	657	>3600	1180.28	79.6	>3600	674.3	5.159	2.6

TAB. 6. $n = 80, k = 60$

		Algo 1			Algo 2			
d	*opt*	*cpu*	$-\beta(0)$	GAP	*cpu*	$-\beta(a^*)$	a^*	GAP
25	518	>3600	609.36	17.6	54.27	530.701	4.53	2.4
	512	>3600	598.60	16.9	56.13	524.5	5.41	2.4
	523	>3600	609.64	16.5	47.44	536.43	23.03	2.5
	528	>3600	623.70	18.1	84.58	541.02	4.48	2.46
	543	>3600	637.46	17.39	24.22	554.56	2.9	2.1
50	968	>3600	1184.55	22.4	36.36	982.85	0.33	1.5
	983	>3600	1182.91	20.3	2.32	993.72	3.458	1.09
	972	>3600	1195.65	23	191.39	988.51	0.724	1.7
	972	>3600	1185.275	21.9	4.05	982.25	4.739	1.05
	989	>3600	1208.56	22.2	53.93	1004.16	5.34	1.5
75	1413	>3600	1782.32	26.1	28.99	1424.47	0.545	0.8
	1410	>3600	1778.84	26.1	7.75	1420.72	0.832	0.76
	1417	>3600	1800.37	27.05	126.45	1431.36	1.503	1.01
	1416	>3600	1786.80	26.2	5.88	1425.73	12.03	0.68
	1394	>3600	1770.42	27	20.05	1404.80	5.159	0.77

Note that the running time of the computation of $\lambda_{min}(Q)$, a^* and $\lambda_{min}(Q_{a^*})$ is very small (less than 1 second). Moreover, the computation of $\beta(0)$ and $\beta(a^*)$ is very fast too.

First, consider results of graphs with $n = 40$ (Tables 1, 2 and 3). It is clear that the bound obtained by *Algorithm 2* ($a = a^*$) is better that the one obtained by *Algorithm 1* ($a = 0$). Indeed, for $k = 10$ and 20 and for $d = 0.25$, the gap is divided by 5 (on average). For $d = 0.5$, it is divided by 11 and for $d = 0.75$, by 13 ($k = 10$) or 24 ($k = 20$). The best results are obtained when $k = \frac{3n}{2} = 30$. In addition, by *Algorithm 2*, the *cpu* time to find the exact solution is very small. Whereas the *cpu* of the first method is larger than one hour, the new method permits to find the optimal value on less than one second (for $k = 20$ and $k = 40$).

Now consider graphs with $n = 80$ (Tables 4, 5 and 6). The optimal solution is never found by the first method within one hour of *cpu* time while, on the average, 15 minutes (for $k = 40$ and $d = 25\%, 50\%$) and one minute (for $k = 60$) are required by the new method. Moreover, the gap is divided by 30 (the best performance is attained for $k = \frac{n}{2}$, $k = \frac{3n}{2}$ and for $d = 0.75$). Note that the worst results are obtained when $k = \frac{n}{4}$.

5 Conclusion

In this paper, we are interested in the exact resolution of general linearly constrained 0-1 quadratic programming problems. Our approach is to modify the objective function while keeping it quadratic. We elaborated a method in two phases : (I) we transform (QP) into a new problem (QP_a), totally equivalent to (QP), which is convex and which depends on a parameter $a \in \mathbb{R}$. A family of problems (QP_a) is built. The principal difference between these problems is that the bound founded by continuous relaxation of (QP_a) depends on a. We denote by a^* the optimal value of a computed by a SDP software. (II) we resolve (QP_{a^*}) by a MIQP software. Note that for $a = 0$, the two-phases method is the one of Hammer and Rubin ([4]).

We choose to apply our method on the densest k-subgraph problem, in graphs with size up to 80. Regarding our results, the method with $a = 0$ is drastically improved not only on the value of GAP but also on the total *cpu* time to find the exact solution. Moreover, we can solve larger problems than recent publications which use same instances but different approaches. In [11], the reported results concern instances with no more than 50 nodes. The integer linear programming approach presented in [1] and based on six different formulations doesn't allow to solve instances with 80 nodes, except in the case of the 75% density instances.

Note that fields of application of our approach are very large with equality constraints. In addition, our method is also adapted to quadratic 0-1 problems with equality and inequality constraints, for example to the generalized quadratic assignment problem. A natural extension of this work is to study other convexifications of (QP) in order to improve the continuous optimum of the convexified quadratic problem.

Références

1. A. Billionnet : Different formulations for solving the heaviest k-subgraph problem. Technical Report CEDRIC 384 (2002) http ://cedric.cnam.fr/PUBLIS/RC384.pdf.
2. A. Billionnet, S. Elloumi : Using a Mixed Integer Quadratic Programming Solver for the Unconstrained Quadratic 0-1 Problem. Technical Report CEDRIC 466 (2003) http ://cedric.cnam.fr/PUBLIS/RC466.pdf.
3. A. Billionnet, A. Faye : A lower bound for a Constrained Quadratic 0-1 Minimization Problem. pp. 135-146. Discrete Applied Mathematics, 74 (1997)
4. PL. Hammer, AA. Rubin : Some remarks on quadratic programming with 0-1 variables. pp. 67-79. RAIRO vol. 3 (1970)
5. C. Hemlberg : Semidefinite Programming for Combinatorial Optimization. ZIB Report (2000)
6. C. Hemlberg : A C++ implementation of the Spectral Bundle Method. Manual version 1.1.1 (2000) http ://www.zib.de/helmberg/SBmethod.
7. C. Helmberg, F. Oustry : Bundle methods to minimize the maximum eigenvalue function. Ch. 11. Hanbook on Semidefinite Programming. Theory, Algorithms and Applications. Lieven (2000)
8. C. Hemlberg, F. Rendl : A spectral bundle method for semidefinite programming. pp. 673-696. SIAM Journal on Optimization (2000)
9. ILOG CPLEX 9.0 http ://www.ilog.com/products/cplex.

10. J. Krarup, D. Pisinger, F. Plastria : Discrete location problems with push-pull objectives. pp. 365-378. Discrete Applied Mathematics 123 (2002)
11. D. Pisinger : Exact solution of p-dispersion problems. Technical Report 99/14, DIKU, University of Copenhagen, Denmark (1999)
12. Scilab http ://scilabsoft.inria.fr.