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

Marie-Christine Plateau, Alain Billionnet and Sourour Elloumi

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@lie.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:

\[(QP): \quad \text{Min } \{ q(x) = x^TQx + c^Tx : Ax = b, x \in \{0, 1\}^n \}\]

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:

$$\text{Min } \{ q_a(x) = x^t Q x + a(Ax - b)^t(Ax - b) : Ax = b, x \in \{0, 1\}^n \}$$

This problem can be written

$$\text{Min } \{ q'_a(x) = x^t Q x + c_a x + t_a : Ax = b, x \in \{0, 1\}^n \}$$

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

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

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 $\lambda_{\text{min}}(Q_a)$, 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 $\{v_1, \ldots, v_n\}$ and a positive integer $k$ belonging to $\{3, \ldots, n - 2\}$, 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 $\frac{n}{3}$ and $\frac{1}{2}$, 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:

$$q^\lambda(x) = q(x) + \lambda \sum_{i=1}^{n} (x_i^2 - x_i)$$

$$= x^t(Q + \text{Diag}(\lambda))x - \lambda \sum_{i=1}^{n} x_i$$

where $\text{Diag}(\lambda)$ is the diagonal matrix whose all diagonal terms are equal to $\lambda$. 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 $(Q + \text{Diag}(\lambda)) \succeq 0$, the continuous relaxation of $(QP^\lambda)$ 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. :

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

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


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

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

Corollary 1.

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

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

Algorithm 1:
1. Compute the smallest eigenvalue of $Q : \lambda_{\text{min}}(Q)$
2. Solve the quadratic problem $(QP - \lambda_{\text{min}}(Q))$ 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

$$q'_a(x) = q(x) + a (Ax - b)^T (Ax - b) = x^T Q_a x + c_a^T x + l_a$$

where $Q_a = Q + aA^T A$, $c_a = c - 2aA^T b$ and $l_a = ab^T b$. 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 \(-\lambda_{\min}(Q_a) \sum_{i=1}^n (x_i^2 - x_i)\) 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.:

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

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 \(\beta(a) \geq \mu^*, \ \forall a > 0\).

**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

\[
\lambda_{\min}(Q_{a_1}) = \min_{x : \|x\|=1} x^T Q_{a_1} x \quad \text{and} \quad \lambda_{\min}(Q_{a_2}) = \min_{x : \|x\|=1} x^T Q_{a_2} x.
\]

Now, for any \(x \in \mathbb{R}^n, x^T Q_{a_1} x - x^T Q_{a_2} x = (a_1 - a_2) x^T A^T A x \geq 0\) because \(A^T A \succeq 0\), 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 \(Ax = b\).

\[
q_{a_1}(x) - q_{a_2}(x) = q(x) + a_1 (Ax - b)^T (Ax - b) - \lambda_{\min}(Q_{a_1}) \sum_{i=1}^n (x_i^2 - x_i)
\]

\[
= -q(x) - a_2 (Ax - b)^T (Ax - b) + \lambda_{\min}(Q_{a_2}) \sum_{i=1}^n (x_i^2 - x_i) \geq 0
\]

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:

\[
\text{Max } \{ \lambda_{\min}(Q_a) : a \in \mathbb{R} \} \quad (1)
\]

It is well-known (see for example [7]) that this problem can be stated as a semidefinite program which dual is
Minimize $<Q, X>$

\[ \text{s.t. } tr(X) = 1 \quad (2) \]

\[ <A' A, X> = 0 \quad (3) \]

\[ X \succeq 0 \]

where $<A, B> = tr(A'B) = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} B_{ij}$ 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 $V = v^t v$. 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 $\lambda_{\text{min}}(Q_{a^*})$ be its optimal value.
2. Solve the quadratic problem $(QP_{a^*})$ 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 :

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

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 $q_a(x) = x^T Q_a x - ak^2 - \lambda_{\text{min}}(Q_a) \sum_{i=1}^{n} (x_i^2 - x_i)$ 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 ($k = \frac{n}{2}, \frac{3n}{4}$). 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 $\rho$ from $[0, 1]$. If $\rho > d$ then $s_{ij}$ is set to 1, otherwise, $s_{ij}$ is set to 0.

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
- $\text{GAP (Algorithm 1)} = \frac{\beta(0) - opt}{opt} \times 100$
\[
\text{GAP (Algorithm 2)} = \frac{\bar{\beta}(a^*) - \text{opt}}{\text{opt}} \times 100
\]

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

<table>
<thead>
<tr>
<th></th>
<th>(\text{Algo 1} )</th>
<th>(\text{Algo 2} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(d)</td>
<td>(\text{cpu} )</td>
</tr>
<tr>
<td>25</td>
<td>264 (240)</td>
<td>87.4 (8.3)</td>
</tr>
<tr>
<td>50</td>
<td>153.6 (9.4)</td>
<td>1.5 (1.5)</td>
</tr>
<tr>
<td>75</td>
<td>226 (3)</td>
<td>47.9 (13.3)</td>
</tr>
</tbody>
</table>

- : no instance could be solved within 3000 seconds

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

<table>
<thead>
<tr>
<th></th>
<th>(\text{Algo 1} )</th>
<th>(\text{Algo 2} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(d)</td>
<td>(\text{cpu} )</td>
</tr>
<tr>
<td>25</td>
<td>1031 (1432.3)</td>
<td>35.2 (3.6)</td>
</tr>
<tr>
<td>50</td>
<td>-</td>
<td>56.1 (5.2)</td>
</tr>
<tr>
<td>75</td>
<td>-</td>
<td>72 (1.3)</td>
</tr>
</tbody>
</table>

- : no instance could be solved within 3000 seconds

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

<table>
<thead>
<tr>
<th></th>
<th>(\text{Algo 1} )</th>
<th>(\text{Algo 2} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(d)</td>
<td>(\text{cpu} )</td>
</tr>
<tr>
<td>25</td>
<td>14.1 (18)</td>
<td>13.8 (2.3)</td>
</tr>
<tr>
<td>50</td>
<td>-</td>
<td>19.4 (1.2)</td>
</tr>
<tr>
<td>75</td>
<td>-</td>
<td>24.2 (0.6)</td>
</tr>
</tbody>
</table>

- : no instance could be solved within 3000 seconds
<table>
<thead>
<tr>
<th>$d$</th>
<th>$opt$</th>
<th>$cpu$</th>
<th>$\beta(0)$</th>
<th>GAP</th>
<th>$cpu$</th>
<th>$\beta(a^*)$</th>
<th>$a^*$</th>
<th>GAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>94</td>
<td>&gt;3600</td>
<td>203.16</td>
<td>116</td>
<td>&gt;3600</td>
<td>109.85</td>
<td>4.53</td>
<td>16.8</td>
</tr>
<tr>
<td>93</td>
<td>&gt;3600</td>
<td>190.58</td>
<td>114</td>
<td>2208</td>
<td>&gt;3600</td>
<td>107.22</td>
<td>5.41</td>
<td>15.2</td>
</tr>
<tr>
<td>100</td>
<td>&gt;3600</td>
<td>203.22</td>
<td>103</td>
<td>331</td>
<td>&gt;3600</td>
<td>114.08</td>
<td>23.03</td>
<td>14.1</td>
</tr>
<tr>
<td>96</td>
<td>&gt;3600</td>
<td>207.91</td>
<td>116</td>
<td>1569</td>
<td>&gt;3600</td>
<td>110.25</td>
<td>4.48</td>
<td>14.8</td>
</tr>
<tr>
<td>97</td>
<td>&gt;3600</td>
<td>212.5</td>
<td>119</td>
<td>&gt;3600</td>
<td>&gt;3600</td>
<td>111.88</td>
<td>2.9</td>
<td>15.3</td>
</tr>
<tr>
<td>50</td>
<td>149</td>
<td>&gt;3600</td>
<td>394.85</td>
<td>169</td>
<td>&gt;3600</td>
<td>167.71</td>
<td>0.33</td>
<td>12.9</td>
</tr>
<tr>
<td>148</td>
<td>&gt;3600</td>
<td>394.30</td>
<td>166.4</td>
<td>&gt;3600</td>
<td>&gt;3600</td>
<td>166.21</td>
<td>3.458</td>
<td>12.3</td>
</tr>
<tr>
<td>144</td>
<td>&gt;3600</td>
<td>398.55</td>
<td>176</td>
<td>&gt;3600</td>
<td>&gt;3600</td>
<td>164.95</td>
<td>0.724</td>
<td>14.5</td>
</tr>
<tr>
<td>144</td>
<td>&gt;3600</td>
<td>395.09</td>
<td>174</td>
<td>&gt;3600</td>
<td>&gt;3600</td>
<td>162.05</td>
<td>4.739</td>
<td>12.5</td>
</tr>
<tr>
<td>149</td>
<td>&gt;3600</td>
<td>402.85</td>
<td>170</td>
<td>&gt;3600</td>
<td>&gt;3600</td>
<td>169.21</td>
<td>5.34</td>
<td>13.5</td>
</tr>
<tr>
<td>75</td>
<td>182</td>
<td>&gt;3600</td>
<td>591.10</td>
<td>226</td>
<td>&gt;3600</td>
<td>202.188</td>
<td>0.545</td>
<td>11.1</td>
</tr>
<tr>
<td>182</td>
<td>&gt;3600</td>
<td>592.94</td>
<td>225</td>
<td>&gt;3600</td>
<td>&gt;3600</td>
<td>203.70</td>
<td>0.832</td>
<td>11.9</td>
</tr>
<tr>
<td>183</td>
<td>&gt;3600</td>
<td>600.12</td>
<td>228</td>
<td>&gt;3600</td>
<td>&gt;3600</td>
<td>205.95</td>
<td>1.503</td>
<td>12.5</td>
</tr>
<tr>
<td>183</td>
<td>&gt;3600</td>
<td>596.6</td>
<td>225</td>
<td>&gt;3600</td>
<td>&gt;3600</td>
<td>202.97</td>
<td>12.03</td>
<td>10.9</td>
</tr>
<tr>
<td>183</td>
<td>&gt;3600</td>
<td>590.14</td>
<td>222</td>
<td>&gt;3600</td>
<td>&gt;3600</td>
<td>201.59</td>
<td>5.159</td>
<td>10.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$d$</th>
<th>$opt$</th>
<th>$cpu$</th>
<th>$\beta(0)$</th>
<th>GAP</th>
<th>$cpu$</th>
<th>$\beta(a^*)$</th>
<th>$a^*$</th>
<th>GAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>280</td>
<td>&gt;3600</td>
<td>406.33</td>
<td>45</td>
<td>347</td>
<td>293.62</td>
<td>4.53</td>
<td>4.8</td>
</tr>
<tr>
<td>273</td>
<td>&gt;3600</td>
<td>399.16</td>
<td>46.2</td>
<td>1975</td>
<td>287.63</td>
<td>5.41</td>
<td>5.3</td>
<td>5.3</td>
</tr>
<tr>
<td>285</td>
<td>&gt;3600</td>
<td>406.44</td>
<td>42.6</td>
<td>639</td>
<td>300.13</td>
<td>23.03</td>
<td>5.3</td>
<td>9.3</td>
</tr>
<tr>
<td>284</td>
<td>&gt;3600</td>
<td>415.83</td>
<td>46.4</td>
<td>249</td>
<td>296.77</td>
<td>4.48</td>
<td>4.4</td>
<td>4.4</td>
</tr>
<tr>
<td>292</td>
<td>&gt;3600</td>
<td>425.04</td>
<td>45.6</td>
<td>487</td>
<td>305.9</td>
<td>2.9</td>
<td>4.7</td>
<td>4.7</td>
</tr>
<tr>
<td>50</td>
<td>488</td>
<td>&gt;3600</td>
<td>789.71</td>
<td>61.8</td>
<td>591</td>
<td>504.89</td>
<td>0.33</td>
<td>3.4</td>
</tr>
<tr>
<td>489</td>
<td>&gt;3600</td>
<td>788.62</td>
<td>61.3</td>
<td>1067</td>
<td>506.08</td>
<td>3.458</td>
<td>3.6</td>
<td>3.6</td>
</tr>
<tr>
<td>484</td>
<td>&gt;3600</td>
<td>797.10</td>
<td>64.7</td>
<td>691</td>
<td>500.02</td>
<td>0.724</td>
<td>3.3</td>
<td>3.3</td>
</tr>
<tr>
<td>479</td>
<td>&gt;3600</td>
<td>790.18</td>
<td>65</td>
<td>1295</td>
<td>494.99</td>
<td>4.739</td>
<td>3.3</td>
<td>3.3</td>
</tr>
<tr>
<td>495</td>
<td>&gt;3600</td>
<td>805.71</td>
<td>62.8</td>
<td>806</td>
<td>512.14</td>
<td>5.34</td>
<td>3.46</td>
<td>3.46</td>
</tr>
<tr>
<td>75</td>
<td>671</td>
<td>&gt;3600</td>
<td>1188.22</td>
<td>77</td>
<td>247</td>
<td>683.11</td>
<td>0.545</td>
<td>1.8</td>
</tr>
<tr>
<td>668</td>
<td>&gt;3600</td>
<td>1185.9</td>
<td>77.5</td>
<td>1182</td>
<td>682.66</td>
<td>0.832</td>
<td>2.2</td>
<td>2.2</td>
</tr>
<tr>
<td>665</td>
<td>&gt;3600</td>
<td>1200.25</td>
<td>80.5</td>
<td>&gt;3600</td>
<td>&gt;3600</td>
<td>685.54</td>
<td>1.503</td>
<td>3</td>
</tr>
<tr>
<td>669</td>
<td>&gt;3600</td>
<td>1191.20</td>
<td>78.1</td>
<td>2376</td>
<td>683.86</td>
<td>12.03</td>
<td>2.2</td>
<td>2.2</td>
</tr>
<tr>
<td>657</td>
<td>&gt;3600</td>
<td>1180.28</td>
<td>79.6</td>
<td>&gt;3600</td>
<td>&gt;3600</td>
<td>674.3</td>
<td>5.159</td>
<td>2.6</td>
</tr>
</tbody>
</table>

**Table 4.** $n = 80, k = 20$

**Table 5.** $n = 80, k = 40$
Note that the running time of the computation of $\lambda_{\min}(Q)\cdot a^*$ 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 $k = \frac{4n}{7} = 30$. 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 $k = \frac{4n}{7}$.

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 $(QP_n)$, totally equivalent

<table>
<thead>
<tr>
<th>$d$</th>
<th>opt</th>
<th>$\beta(0)$</th>
<th>GAP</th>
<th>$\beta(a^*)$</th>
<th>$a^*$</th>
<th>GAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>518</td>
<td>&gt;3600</td>
<td>695.36</td>
<td>17.6</td>
<td>64.27</td>
<td>530.70</td>
</tr>
<tr>
<td>512</td>
<td>&gt;3600</td>
<td>598.60</td>
<td>16.9</td>
<td>56.13</td>
<td>524.5</td>
<td>5.41</td>
</tr>
<tr>
<td>523</td>
<td>&gt;3600</td>
<td>609.64</td>
<td>16.5</td>
<td>47.44</td>
<td>536.43</td>
<td>23.03</td>
</tr>
<tr>
<td>528</td>
<td>&gt;3600</td>
<td>623.70</td>
<td>18.1</td>
<td>84.58</td>
<td>541.02</td>
<td>4.48</td>
</tr>
<tr>
<td>543</td>
<td>&gt;3600</td>
<td>637.46</td>
<td>17.39</td>
<td>24.22</td>
<td>554.56</td>
<td>2.9</td>
</tr>
<tr>
<td>75</td>
<td>1413</td>
<td>&gt;3600</td>
<td>1782.32</td>
<td>26.1</td>
<td>28.99</td>
<td>1434.47</td>
</tr>
<tr>
<td>1410</td>
<td>&gt;3600</td>
<td>1778.84</td>
<td>26.1</td>
<td>7.75</td>
<td>1420.72</td>
<td>0.832</td>
</tr>
<tr>
<td>1417</td>
<td>&gt;3600</td>
<td>1800.37</td>
<td>27.05</td>
<td>12645</td>
<td>1431.36</td>
<td>1.508</td>
</tr>
<tr>
<td>1416</td>
<td>&gt;3600</td>
<td>1786.80</td>
<td>26.2</td>
<td>5.88</td>
<td>1425.73</td>
<td>12.08</td>
</tr>
<tr>
<td>1394</td>
<td>&gt;3600</td>
<td>1770.42</td>
<td>27</td>
<td>20.06</td>
<td>1404.80</td>
<td>5.159</td>
</tr>
</tbody>
</table>
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 found 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 \textit{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

