Solving k-cluster problems to optimality with semidefinite programming

Jérôme Malick, Frédéric Roupin

Mathematical Programming manuscript No.
(will be inserted by the editor)

Solving k-cluster problems to optimality with semidefinite programming

Jérôme Malick · Frédéric Roupin

the date of receipt and acceptance should be inserted later

Abstract This paper deals with the computation of exact solutions of a classical NP-hard problem in combinatorial optimization, the k-cluster problem. This problem consists in finding a heaviest subgraph with k nodes in an edge weighted graph. We present a branch-and-bound algorithm that applies a novel bounding procedure, based on recent semidefinite programming techniques. We use new semidefinite bounds that are less tight than the standard semidefinite bounds, but cheaper to get. The experiments show that this approach is competitive with the best existing ones.

Keywords combinatorial optimization, k-cluster, exact resolution, semidefinite programming, Lagrangian relaxation, branch-and-bound

1 Introduction, motivations

Given an edge weighted graph with n vertices, the k-cluster problem consists in finding a subgraph with the heaviest weight and with exactly k nodes ($1 < k < n - 1$). This problem is a classical problem of combinatorial optimization; it is also known under the name of "heaviest k-subgraph problem", "k-dispersion problem", "k-defence-sum problem" [KPP02], and "densest subgraph problem" when all the edge weights are equal to 1. This problem can be seen as a generalization of the max-clique problem, and also as a particular quadratic knapsack problem where all the costs are equal.

We recall briefly some complexity results about the k-cluster problem in order to argument that it is a hard combinatorial problem. This problem does not admit a polynomial time approximation scheme [Kho05], and it is known to be NP-hard even in very special cases: in unweighted bipartite graphs of maximal degree 3 [FL01] and planar graphs [KB91]. It can be solved in polynomial-time in trees [PS83], cographs

J. Malick
CNRS, Lab. Jean Kuntzmann, Grenoble (France)
E-mail: jerome.malick@inria.fr

F. Roupin
LIPN - CNRS : UMR7030 - Université Paris 13 (France)
E-mail: Frederic.Roupin@lipn.univ-paris13.fr

and split graphs [CP84]. Strong negative results have been obtained about the approximability of the clique problem, but there is no direct way to use them for the k-cluster problem: it is open whether an approximation algorithm with a fixed ratio exists in general. The best general approximation ratio is actually $O(n^{\frac{1}{3}})$ [FKP01], and several approximation algorithms with a better ratio have been designed for special graphs (see e.g. [LMZ08] for recent results and references).

These results give an idea of how tough the k-cluster problem is to be solved to optimality. Even if the machinery of linear-optimization-based branch-and-bound algorithms has reached a high degree of development (see for example the recent [TA05]), only medium-size instances of the k-cluster problem can be solved in general with these solvers (see [Erk90], [Bil05], [Pis06]). Linear relaxations seem to be not tight enough for k-cluster in order to be used efficiently within an algorithm for exact resolution. Therefore, the best exact and approximate resolution methods devoted to this problem are based on nonlinear approaches, such as convex quadratic programming [BEP09] or semidefinite programming [HYZ02], [Rou04], [JS05].

We present in this paper an approach for solving k-cluster problems to optimality by using standard branch-and-bound techniques together with new semidefinite programming bounds. The bounding procedure trades computing time for a (small) deterioration of the quality of the bound, so that it fits well within a branch-and-bound algorithm. To get these semidefinite bounds, we follow the scheme sketched in [Mal07] for general binary quadratic program: we simplify and specify it for the k-cluster problem – and we push the development to show numerically that this approach is efficient. The main contribution of this paper is to prove numerically that these new semidefinite bounds are interesting in view of solving hard combinatorial optimization problems to optimality. For large-scale k-cluster problems indeed, our approach is competitive with the best known approach [BE06], [BEP09] that uses CPLEX.

The paper is organized as follows. We formulate the k-cluster problem in Section 2 in a way to introduce the new semidefinite bounds in Section 3. We compare these bounds with the usual semidefinite bound in Section 4 to argue that they have nice features in view of exact resolution. Finally we embed them within a simple branch-and-bound algorithm in Section 5 and compare the overall efficiency of this approach with the best known solver for exact resolution of k-cluster problems.

2 Equivalent formulations

This section introduces the notation used in the sequel, and presents a sequence of equivalent formulations of the k-cluster problem: namely, the initial formulation (1), simplified (2), augmented with additional constraints (3), the formulation as a binary quadratic programming problem (4) and the formulation as semidefinite programming problems (6), (8). All these formulations are standard, except the last one. We emphasize: all of them are exact reformulations; we start relaxing in the next section.

Initial modeling as a $\{0,1\}$-quadratic programming problem. Consider an undirected weighted graph $G = (V, E)$ with n vertices $\{v_1, \ldots, v_n\}$ and with nonnegative weights w_{ij} on edges (v_i, v_j). For an integer k in $\{2, \ldots, n-2\}$, the so-called k-cluster problem consists in determining a subset S of k vertices such that the total edge weight of the subgraph induced by S is maximized. To select subgraphs, assign a decision variable $z_i \in \{0, 1\}$ for each node ($z_i = 1$ if the node is taken, and $z_i = 0$ if the node is not). The

weight of the subgraph given by z is $\sum_{(i,j)\in E} w_{ij}\, z_i\, z_j$. Thus the k-cluster problem can be phrased as the 0-1 quadratic problem

$$\begin{cases} \max & \frac{1}{2}\sum_{i=1}^n \sum_{j=1}^n w_{ij}\, z_i\, z_j \\ & \sum_{i=1}^n z_i = k, \quad z \in \{0,1\}^n. \end{cases} \tag{1}$$

We rewrite this problem synthetically as

$$\begin{cases} \max & \frac{1}{2}\, z^\top W\, z \\ & e^\top z = k, \quad z \in \{0,1\}^n. \end{cases} \tag{2}$$

with the vector of all ones $e = (1, \ldots, 1) \in \mathbb{R}^n$ and the weight-matrix $W = (w_{ij})_{ij}$ of the graph G (which is a symmetric $n \times n$ matrix).

Reformulation 1: Enforcement of redundant constraints. In view of relaxing, we add redundant constraints to have a better duality gap. We introduce the standard product-constraints as follows. Observe that a feasible z (that is, such that $e^\top z = k$ and $z \in \{0,1\}^n$) also satisfies for all $j \in \{1, \ldots, n\}$

$$\sum_{i=1}^n z_i z_j = k z_j.$$

The left-hand side of this equality is quadratic in z; so we introduce the symmetric $n \times n$-matrix C_j such that $\sum_{i=1}^n z_i z_j = \frac{1}{2} z^\top C_j z$. Adding these n product constraints, we come up with the following equivalent formulation of k-cluster

$$\begin{cases} \max & \frac{1}{2} z^\top W\, z \\ & e^\top z = k, \quad z \in \{0,1\}^n \\ & z^\top C_j\, z = 2k\, z_j\,, \quad j \in \{1, \ldots, n\}. \end{cases} \tag{3}$$

These product constraints are the only one we add, because they are tight in a certain way; this follows indeed from the general results that we recall briefly here. For a $\{0,1\}$-quadratic problem, it is well-known (see e.g. [LO99]) that the best bound by Lagrangian duality is obtained when dualizing only the $\{0,1\}$-constraints (and not the linear constraints). Though this bound does not lead directly to a SDP problem, the semidefinite relaxation of the general problem reinforced by the product constraints (e.g. (3) for k-cluster) is equivalent to this best bound [FR07]. Adding other redundant quadratic *equality* constraints would lead to the same bound, so we stick with (3).

We also recall that adding to (2) the single constraint $(e^\top z - k)^2 = 0$ instead of the n product constraints is an equivalent approach: the two formulations lead to two SDP relaxations which give the same bound - but on which solvers behave differently. The numerical comparison between the two SDP formulations is made in [MR10a] showing that (3) is preferable for our developments. We will come back to this in Section 4.2.

Reformulation 2: Change of variables. Dealing with quadratic problems with $\{-1, 1\}$ constraints and purely quadratic problems will ease the next developments. So we introduce the new variable $x = (x_0, \ldots, x_n) \in \mathbb{R}^{n+1}$ performing two operations:

1. The change of binary variable $x_i = 2z_i - 1 \in \{-1, 1\}$ for $i \in \{1, \ldots, n\}$.

2. The "homogenization" of the quadratic forms with the extra-variable $x_0 \in \{-1, 1\}$. See more precisely the role of this homogenization in Lemma 3 in appendix. A graph interpretation is as follows: we add an isolated vertex x_0 to the graph, and we want to compute a $(k+1)$-cluster containing this vertex; in other words, x_0 is like a "flag" that indicates which vertices are in the k-cluster.

The change of variable $z \to x$ gives the following equivalent formulation of k-cluster

$$\begin{cases} \max \quad x^\top Q\, x \\ \quad x^\top Q_j\, x = 4k - 2n, \quad j \in \{0, \dots, n\} \\ \quad x \in \{-1, 1\}^{n+1} \end{cases} \tag{4}$$

where the $n+2$ symmetric matrices Q and Q_j (for $j \in \{0, \dots, n\}$) are defined by

$$Q := \frac{1}{4} \begin{bmatrix} e^\top W e & e^\top W \\ We & W \end{bmatrix}, \qquad Q_0 := \begin{bmatrix} 0 & e^\top \\ e & 0 \end{bmatrix} \qquad \text{and} \qquad Q_j := \begin{bmatrix} 0 & \tilde{e}_j^\top \\ \tilde{e}_j & C_j \end{bmatrix} \tag{5}$$

with $\tilde{e}_j = e + (n - 2k)e_j$ the vector of $\mathbb{R}^n$ made up from e and e_j the j-th element of the canonical basis of $\mathbb{R}^n$. The change of variable and the reformulation are detailed in Lemma 3 in appendix.

Reformulation 3: Lifting in matrix space. We now lift the problem (4) up to the matrix space $\mathcal{S}_{n+1}$ in order to transform the quadratic forms with respect to $x \in \mathbb{R}^n$ to linear forms with respect to a matrix variable X. We follow the classical pattern (e.g. [Lov79], [GW95]) and we use classical notation, recalled below. The natural inner product in matrix space $\mathcal{S}_{n+1}$ is defined for any $X, Y \in \mathcal{S}_{n+1}$ by

$$\langle X, Y \rangle = \sum_{i,j=1}^{n+1} X_{ij} Y_{ij} = \operatorname{trace}(XY).$$

This inner product is very convenient for our purposes through the relation:

$$\forall\, x \in \mathbb{R}^{n+1},\ \forall\, A \in \mathcal{S}_{n+1}, \qquad x^\top A\, x = \langle A, x x^\top \rangle.$$

We denote $\|\cdot\|$ the associated norm; it is the same notation as the norm in $\mathbb{R}^n$, but no confusion should be possible since the matrices are represented by capital letters.

The idea of the standard lifting is to introduce the symmetric matrix of rank one $X = xx^\top$. With the help of X, we express the binary constraints $x_i \in \{-1, 1\}$ (that is ${x_i}^2 = 1$) as $X_{ii} = 1$, and the quadratic constraint $x^\top A\, x = c$ as $\langle A, X \rangle = c$. So we get the following equivalent formulation of (4)

$$\begin{cases} \max \quad \langle Q, X \rangle \\ \quad \langle Q_j, X \rangle = 4k - 2n, \quad j \in \{0, \dots, n\} \\ \quad X_{ii} = 1, \quad i \in \{0, \dots, n\} \\ \quad X = xx^\top. \end{cases}$$

Notice then that $X = xx^\top$ is a rank-one symmetric semidefinite matrix, and that conversely any rank-one symmetric semidefinite matrix can be written this way. So we can cast the above problem as an SDP linear problem with rank-one constraint

$$\begin{cases} \max \quad \langle Q, X \rangle \\ \quad \langle Q_j, X \rangle = 4k - 2n, \quad j \in \{0, \dots, n\} \\ \quad \langle E_i, X \rangle = 1, \quad i \in \{0, \dots, n\} \\ \quad \operatorname{rank}(X) = 1, \quad X \succeq 0. \end{cases} \tag{6}$$

where E_i is the matrix with zeros entries except in position (i, i) where there is a one.

Reformulation 4: Introducing the spherical constraint. The hard constraint in (6) is the rank-one constraint, which is moreover difficult to handle. So we go one step further to the standard SDP lifting by rewriting this rank-one constraint as a norm, along the lines of [Mal07]. The (curious and) useful property is the following (see [Mal07, Theorem 1]): for all $X \succeq 0$ satisfying $X_{ii} = 1$, we have $\|X\| \leq n+1$ and moreover

$$\|X\| = n+1 \iff \operatorname{rank} X = 1. \tag{7}$$

Therefore the idea is to replace the rank-one constraint in the SDP formulation of the k-cluster by the constraint $\|X\|^2 = (n+1)^2$, called the "spherical constraint". Thus we have the following formulation of k-cluster that we formalize in the next proposition; this is the formulation we will use in this paper.

Lemma 1 *With the notation of this section, the optimal value of the k-cluster problem is equal to the optimal value of*

$$\begin{cases} \max \quad \langle Q, X\rangle \\ \quad \langle Q_j, X\rangle = 4k-2n, \quad j \in \{0,\dots,n\} \\ \quad \langle E_i, X\rangle = 1, \quad i \in \{0,\dots,n\} \\ \quad \|X\|^2 = (n+1)^2 \\ \quad X \succeq 0. \end{cases} \tag{8}$$

Proof The proof is obvious in view of the previous developments: by Lemma 3, the optimal value of (1) is equal to the one of (4) which is in turn equivalent to (6), and then to (8) by (7). □

We have transformed a quadratic problem with quadratic constraint in $\mathbb{R}^{n+1}$ to a linear SDP problem with one single quadratic constraint (the spherical constraint $\|X\|^2 = (n+1)^2$). This quadratic constraint carries the nonconvexity and the combinatorial difficulty of the initial problem. The idea is now to treat it by Lagrangian dualization, as explained in the Section 3.2.

3 Semidefinite relaxations

3.1 Standard semidefinite bounds

The previous section explains the formulation of the k-cluster problem as SDP problems (6) and (8). All the combinatorial difficulty of the original problem is now located in the rank-one constraint in (6), or in the spherical constraint in (8). The standard SDP relaxation then consists in enlarging the feasible set in (6) by dropping the rank constraint, to derive a convex problem. In our situation, the standard SDP relaxation of the k-cluster problem (6) is indeed

$$\begin{cases} \max \quad \langle Q, X\rangle \\ \quad \langle Q_j, X\rangle = 4k-2n, \quad j \in \{0,\dots,n\} \\ \quad \langle E_i, X\rangle = 1, \quad i \in \{0,\dots,n\} \\ \quad X \succeq 0. \end{cases} \tag{9}$$

We note that this SDP problem is equivalent to the one obtained from the $\{0,1\}$-formulation (3) (used by [BEP09] in particular). As mentioned in the discussion following (3), it is also equivalent to the SDP problem coming from (2) augmented by

the squared constraint $(e^\top x - k)^2 = 0$. More precisely, by following the same lines to go from (3) to (9), we get the equivalent SDP problem

$$\begin{cases} \max & \langle Q, X \rangle \\ & \langle Q_S, X \rangle = -(2k-n)^2 \\ & \langle E_i, X \rangle = 1, \quad i \in \{0, \ldots, n\} \\ & X \succeq 0 \end{cases} \tag{10}$$

with a particular $Q_S \in \mathcal{S}_{n+1}$. In practice, this bound is known to be tight and several approximation algorithms have been proposed using it (see e.g. [HYZ02]).

Those SDP problems (9) and (10) have linear objective functions and as well as linear constraints together with the conic constraint. Over the last decade, several algorithms and associated solvers have been developed to solve problems of this type. We refer for instance to the introduction of the recent [MPRW09] for references and for some explanations about the different approaches; here we just point out:

- **Interior points**. The most known and used methods for SDP; they have nice complexity theory (see e.g. the review [Tod01]), and reliable numerical performances. For the forthcoming numerical experiments of Section 4.2, we use the software CSDP [HRVW96], [Bor99].
- **Spectral bundle**. The bundle methods of convex optimization [HUL93] have been specialized successfully to semidefinite programming, and especially to applications of semidefinite programming in combinatorial optimization [HR00]. For the numerical experiments, we use the software SB [Hel04].
- **Regularization**. Recently other approaches based on regularization (augmented Lagrangian and proximal method) have been developed for solving large-scale SDP problems, as for example the relaxation of max-stable problem [MPRW09]. Let us also mention the penalized augmented Lagrangian method of PENNON [KS07].
- **First-order methods**. To tackle ever larger SDP problems, several first-order methods have been developed: we mention for example low-rank methods (see [BM03] and references therein) that have recently drawn renewed interest.

Though there exist several types of methods and several efficient solvers, computing the SDP bound (9) or (10) is still expensive, and this may prevent its direct use as bounding procedure within a branch-and-bound for solving k-cluster to optimality. Note in particular that the method of [BEP09] uses the SDP bound only once at the beginning of the search tree (as an initial calibration of the quadratic convex relaxation used later for bounding, see more in Section 5). We are not aware of research for k-cluster in the line of [RRW10] for max-cut. Here, we consider different bounds that have an SDP-quality but that are less tight that (9) and (10). In the next sections, we argue that those new bounds are easier to compute and then well-adapted to be embedded within a branch-and-bound.

3.2 New family of semidefinite bounds

We investigate now a way to keep a SDP-like quality of bound without paying the full computational price to get it. The idea is to keep the rank-one constraint, to write it with the help of the spherical constraint as in (8) and then to dualize it. This approach is sketched in [Mal07] in a general setting; here, we specialize the study for the k-cluster to push to the end and to get tools for exact resolution.

For the real parameter $\alpha \in \mathbb{R}$, we consider the Lagrangian function

$$L(X;\alpha) := \langle Q, X\rangle - \frac{\alpha}{2}(\|X\|^2 - (n+1)^2)$$

and the associated dual function

$$\Theta(\alpha) := \left\{ \begin{array}{ll} \max & L(X;\alpha) \\ & \langle Q_j, X\rangle = 4k-2n, \quad j \in \{0,\ldots,n\} \\ & \langle E_i, X\rangle = 1, \quad i \in \{0,\ldots,n\} \\ & X \succeq 0. \end{array} \right. \tag{11}$$

By weak duality, each value $\Theta(\alpha)$ is an upper bound for the optimal value of k-cluster, since we can write: for all feasible X for (8) (and then in (11)),

$$\langle Q, X\rangle = L(X,\alpha) \leq \Theta(\alpha) \tag{12}$$

so that $\Theta(\alpha)$ is upper bound for (8) indeed.

Hence we have a new family of SDP bounds $\Theta(\alpha)$ (parameterized by $\alpha \in \mathbb{R}$). In a way, these bounds generalize the standard SDP bound (9): observe indeed that for $\alpha = 0$, (11) is exactly (9). They have moreover interesting properties: the important theoretical properties are gathered in the following proposition; we discuss the numerical properties in the next section. In what follows, val$(*)$ denotes the optimal value of an optimization problem numbered by $(*)$.

Proposition 1 *Function $\Theta\colon \mathbb{R} \to \mathbb{R}$ defined by* (11) *satisfies the following properties:*

- *For any $\alpha \in \mathbb{R}$, we have a bound for k-cluster*

$$\Theta(\alpha) \geq \text{val}\,(8) = \text{val}\,(1).$$

- *If $\alpha > 0$, this bound is weaker than the SDP bound*

$$\Theta(\alpha) \geq \text{val}\,(9) \geq \text{val}\,(8) = \text{val}\,(1),$$

  but we get arbitrarily close to the SDP bound when α tends to 0

$$\lim_{\alpha \to 0, \alpha > 0} \Theta(\alpha) = \text{val}\,(9).$$

- *If $\alpha > 0$, we may compute $\Theta(\alpha)$ by solving a semidefinite least-squares problem:*

$$\Theta(\alpha) = \Big(\frac{\alpha}{2}(n+1)^2 + \frac{1}{2\alpha}\|Q\|^2\Big) - \alpha \left\{ \begin{array}{ll} \min & \frac{1}{2}\|X - Q/\alpha\|^2 \\ & \langle Q_j, X\rangle = 4k-2n, \quad j \in \{0,\ldots,n\} \\ & \langle E_i, X\rangle = 1, \quad i \in \{0,\ldots,n\} \\ & X \succeq 0 \end{array} \right. \tag{13}$$

Proof The first property comes from (12). The second property about the comparison between $\Theta(\alpha)$ and $\Theta(0)$ $=$val(9) follows from Theorem 3 of [Mal07]: the function $\Theta\colon \mathbb{R} \to \mathbb{R}$ is convex (so continuous at 0) and nondecreasing (so $\Theta(\alpha) \geq \Theta(0)$ if $\alpha > 0$). To prove the third point, we write for $\alpha > 0$

$$\begin{aligned} L(X,\alpha) &= \frac{\alpha}{2}(n+1)^2 - \frac{\alpha}{2}(\|X\|^2 - 2\langle X, Q/\alpha\rangle) \\ &= \frac{\alpha}{2}(n+1)^2 - \frac{\alpha}{2}\Big(\Big\| X - \frac{Q}{\alpha} \Big\|^2 - \frac{\|Q\|^2}{\alpha^2}\Big) \\ &= \Big(\frac{\alpha}{2}(n+1)^2 + \frac{1}{2\alpha}\|Q\|^2\Big) - \frac{\alpha}{2}\Big\| X - \frac{Q}{\alpha} \Big\|^2, \end{aligned}$$

which yields that (11) corresponds to (13). □

The point is thus the following: for $\alpha > 0$, the new SDP bound $\Theta(\alpha)$ is always less tight than the usual SDP bound $\Theta(0)$, but it can be reduced arbitrary close to it by decreasing α. The key question is now how easier is $\Theta(\alpha)$ to compute; this is addressed in the next section.

4 Relaxed resolution: computation of bounds

The new SDP bound $\Theta(\alpha)$ looks more complicated than the usual SDP bound $\Theta(0)$. It turns out that nice geometrical properties make it cheaper to compute. Section 4.1 explains the computation, and then Section 4.2 presents a numerical comparison.

4.1 Computing the bound by projection

By (13), computing the new SDP bound $\Theta(\alpha)$ when $\alpha > 0$ comes down to solving

$$\begin{cases} \min \quad \frac{1}{2}\|X - Q/\alpha\|^2 \\ \quad \langle Q_j, X\rangle = 4k - 2n, \quad j \in \{0, \dots, n\} \\ \quad \langle E_i, X\rangle = 1, \quad i \in \{0, \dots, n\}, \quad X \succeq 0. \end{cases} \tag{14}$$

This quadratic SDP problem has a simple geometric interpretation: Consider indeed the affine subspace Aff in $\mathcal{S}_{n+1}$ defined by the affine constraints of this problem

$$\text{Aff} := \Big\{ X \in \mathcal{S}_{n+1} : \ \langle Q_j, X\rangle = 4k - 2n, \ \langle E_i, X\rangle = 1, \ \text{for } i, j \in \{0, \dots, n\} \Big\}.$$

Then the problem (14) is

$$\begin{cases} \min \quad \|X - Q/\alpha\|^2 \\ \quad X \in \text{Aff} \cap \mathcal{S}^+_{n+1} \end{cases}$$

and thus consists in projecting the matrix Q/α onto the intersection $\text{Aff} \cap \mathcal{S}^+_{n+1}$ of the affine subspace Aff and the cone $\mathcal{S}^+_{n+1}$. This problem is an instance of the so-called semidefinite least-squares [Mal04]. Efficient algorithms have been recently developed to solve these problems, based on three paradigms: (1) alternating projection methods [Hig02], (2) interior-point methods [TTT06], (3) duality [Mal04] (see also [QS06] and [BH08] for developments on an important special case).

We have developed our own semidefinite least-squares solver to solve (14). This solver follows the general method of [Mal04] while being specific to be more efficient in exploiting the particular features of (14). Interesting properties are summarized in the next proposition. theorem. Consider the linear operator $\mathcal{A}\colon \mathcal{S}_{n+1} \to \mathbb{R}^{2n+2}$ defined by

$$\mathcal{A}(X) := \big(\langle Q_0, X\rangle, \dots, \langle Q_n, X\rangle, \langle E_0, X\rangle, \dots, \langle E_n, X\rangle\big),$$

and the vector $b \in \mathbb{R}^{2n+2}$ defined by

$$b := (4k - 2n, \dots, 4k - 2n, 1, \dots, 1),$$

such that the affine subspace Aff admits $\mathcal{A}X = b$ as an equation. Notice that the adjoint of $\mathcal{A}$, denoted by $\mathcal{A}^*\colon \mathbb{R}^{2n+2} \to \mathcal{S}_{n+1}$, and defined by $\langle \mathcal{A}^*(y), X\rangle = y^\top \mathcal{A}(X)$ for all $y \in \mathbb{R}^{2n+2}$ and $X \in \mathcal{S}_{n+1}$, is

$$\mathcal{A}^*(y) = \sum_{i=0}^{n} y_i Q_i + \sum_{i=0}^{n} y_{i+n+1} E_i.$$

For a given $y \in \mathbb{R}^{2n+2}$, we also define the matrix

$$X(y) := \mathrm{P}_{\mathcal{S}^+_{n+1}}\left(Q/\alpha + \mathcal{A}^*(y)\right)$$

where $\mathrm{P}_{\mathcal{S}^+_{n+1}}$ is the projection onto $\mathcal{S}^+_{n+1}$. Note that the dependence with respect to α is dropped for simplicity.

Proposition 2 *The function $f\colon \mathbb{R}^{2n+2} \to \mathbb{R}$ defined by*

$$f(y) := \frac{1}{2}\|X(y)\|^2 - y^\top b$$

is convex and differentiable with gradient $\nabla f(y) = \mathcal{A}X(y) - b$. Besides, we have

$$\mathrm{val}\,(14) \geq \frac{\|Q\|^2}{2\alpha^2} - f(y) \tag{15}$$

for all $y \in \mathbb{R}^{2n+2}$. Assume furthermore that there exists a vector $y^\star \in \mathbb{R}^{2n+2}$ that attains the minimum $f^\star$ of f; then equality holds in (15) *and $X^\star = X(y^\star)$ is the unique solution of* (14).

Proof Up to a change of sign and to the constant $\frac{\|Q\|^2}{2\alpha^2}$, the function f corresponds to the function θ in [Mal04]. We just apply results of this paper: the convexity and differentiability of f comes from Theorems 3.1 and 3.2; (15) is equation (4.2); and the rest comes from Theorem 4.1. □

The differentiability of f allows us to use standard algorithms for unconstrained differentiable optimization for solving (14) thus to compute the bound $\Theta(\alpha)$. We can cite, among others: gradient, quasi-Newton, Newton-like methods as truncated generalized Newton (see [NW99], [BGLS03]). For its simplicity and robustness, we choose the limited memory quasi-Newton algorithm [GL89], which has proved to be efficient in many (academic and industrial) applications.

4.2 Numerical comparisons of semidefinite bounds

This section gives numerical comparisons of the new SDP bound $\Theta(\alpha)$ and the usual SDP bound $\Theta(0)$ in terms of tightness, computation time and balance between both.

Solvers and settings. To compute the SDP bound $\Theta(0)$, that is solving (9), we use:

- SB [Hel04], a SDP solver based on the spectral bundle method [HR00]. This software can handle large-scale problems and is known to be efficient in the context of combinatorial optimization (see e.g. the recent [RRW10]). We use it with default settings, except that we add an initial scaling of the constraints and we set the stopping criterion to `1e-4` (instead of `1e-5`).
- CSDP [Bor99], a robust and efficient interior point solver. We use it with default settings, except that we activate "Fastmode" and we set the stopping criterion "objtol" to `1e-5` (instead of `1e-8`).

To compute the SDP bound $\Theta(\alpha)$, that is essentially solving (14), we use:

– SDLS, a home-made solver implementing a version of the algorithm of [Mal04] adapted for solving (14). We use the quasi-Newton algorithm [GL89] to minimize the function f of Proposition 2. We set the stopping criterion to `1e-7`, and, as for SB, the constraints of (14) are scaled so that the constraints matrices are of norm 1. For sake of simplicity, we also decide to keep α constant. In view of preliminary experiments, we then fix α to `1e-4`: our strategy is indeed to have a SDP-like bound, and $\Theta(10^{-4})$ is almost as tight as $\Theta(0)$, while its computing time is reasonable. We illustrate this with Figure 1 which plots (for an instance with $n = 300$) the decrease of $\Theta(\alpha)$ to $\Theta(0)$ when $\alpha \to 0$ and the increase of the corresponding computing time. For example, $\Theta(\alpha)$ for $\alpha = 10^{-4}$ (the leftmost point) is much more tight than for $\alpha = 0.002$ with less than twice the computing price.

Different formulations of the same bound. Computing times depend obviously on the solvers, and in turn the performances of the solvers depend on the formulation of the bounds. In our case, the SDP bound can be computed by solving (9) or (10), and each of both turns out to advantage one solver.

To have a fair comparison between the three solvers, we have done preliminary tests (reported in [MR10a]) to choose the best semidefinite formulation for each solver. As expected, (10) is better suited for CSDP (when $n = 100$ about five times faster than using (9)), since here the computing time directly depends on the number of variables and constraints. On the other hand, the formulation (9) provides the best results for SB and SDLS (when $n = 100$ about four times faster than using (10)). In the numerical experiments below, we thus have: SB solves (9), CSDP solves (10), and SDLS solves (14).

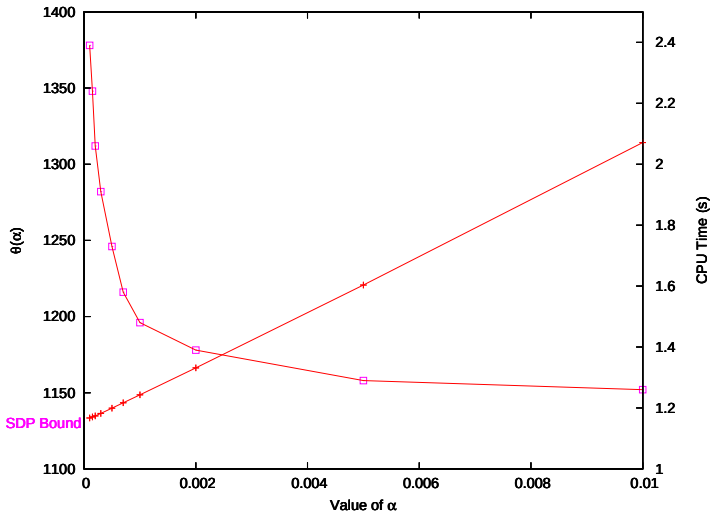

Fig. 1 Illustration of the features of $\Theta(\alpha)$ on a graph with $n = 300$, $k = 75$ and $d = 25\%$: when $\alpha \to 0$, we get more tightness ($\Theta(\alpha) \to \Theta(0)$) but for an increasing computing cost. The leftmost point corresponds to $\alpha = 10^{-4}$.

.

Test problems. We consider instances of k-cluster, randomly generated as follows. Given a density $d \in [0,1]$, a random number $\rho \in [0,1]$ is generated for any pair of indexes $i < j$; if $\rho > d$, then w_{ij} is set to 0, otherwise it is set to 1.

The parameters of these instances are: the size of the graph n, the graph density d and the value of k. In this section, we take

$$n = 80, 100, 300, \qquad d = 25\%, 50\%, 75\% \qquad \text{and} \qquad k = n/4, n/2, 3n/4.$$

We have 5 instances of k-cluster problems for each set of parameters. Each forthcoming numerical result is thus an averaged of 5 results.

The instances with 80 and 100 vertices have already been used in several others papers, as [Bil05] and especially [BEP09] which is the reference method for the comparison of Section 5.2. Note that these problems have unweighted graphs, which are at least as hard as the ones with weighted graphs.

Numerical results. The numerical experiments have been carried out on a Pentium IV 2.2 GHz with 1 Go of RAM under Linux. Table 1 reports the computation times (in seconds) of the three solvers to compute the SDP bounds. Figure 2 illustrates the run on two characteristic instances. The results show that $\Theta(\alpha)$ provides a SDP-quality bound which is easier to get than the usual SDP bound $\Theta(0)$.

Since the solvers do not compute the same bounds (remember that SB and CSDP compute $\Theta(0)$ and SDLS computes $\Theta(\alpha)$), we report:

- gap SDP – the relative difference between $\Theta(\alpha)$ and $\Theta(0)$, i.e., $(\Theta(\alpha) - \Theta(0))/\Theta(0)$,
- time SDLS – the computing time for SB and CSDP to achieve the value $\Theta(\alpha)$.

Some comments are in order. Although less tight, the new bounds $\Theta(\alpha)$ are very close to $\Theta(0)$: the relative gap always lower than 0.25%. Remember moreover that the gap will be further reduced within the branch-and-bound by rounding down the bound to the integer value. We notice also that the gap is almost constant in absolute value so that its relative value decreases mechanically, when k or d increase (since the optimal value of the instance increases in this case).

The crucial point is that our solver provides bounds faster than SB and CSDP. As expected, CSDP is very robust and efficient for the medium-size instances ($n = 80$); but compared to SB, its performance deteriorates when the size of the problem gets bigger. In this case, the interest of SDLS becomes clear: the gap between the bounds does not increase, whereas the running times increase slower than those of SB and CSDP.

Furthermore, our solver provides these bounds faster whenever the solving process is interrupted. This is illustrated in Figure 2 for example: the convergence curve is always below the one of CSDP and also the one of SB (excepted at the very beginning). Note also that none of SB and CSDP dominates the other.

We insist finally on the three following points.

1. Robustness. As expected, the running times CSDP are almost constant for the instances of the same size (see for example the two examples of Figure 2). Our solver has also a similar behavior, which is a desirable property in the context of branch-and-bound. Note that SB does not have this behavior (it was faster than CSDP on some instances, but slower on others). For example, let us give the standard computing time deviations for $n = 100$. More precisely, the mean standard deviations for the five instances at given triplet (n, k, d) are 0.02 for SDLS, 0.11 for CSDP, and 2.32 for SB.

			$\Theta(\alpha)$		$\Theta(0)$ by SB		$\Theta(0)$ by CSDP	
n	k	$d(\%)$	time (sec.)	gap(%) SDP	time (sec.)	time SDLS	time (sec.)	time SDLS
80	20	25	0.18	0.21%	0.55	0.23	0.29	0.19
		50	0.18	0.14%	0.87	0.41	0.45	0.35
		75	0.16	0.12%	1.37	0.61	0.31	0.24
	40	25	0.17	0.06%	0.54	0.37	0.35	0.28
		50	0.10	0.04%	0.98	0.80	0.33	0.28
		75	0.07	0.03%	1.39	0.81	0.40	0.32
	60	25	0.15	0.02%	1.17	0.85	0.35	0.29
		50	0.14	0.01%	1.34	1.30	0.35	0.29
		75	0.15	0.01%	1.64	1.41	0.33	0.29
		mean	**0.15**	**0.07%**	**1.09**	**0.76**	**0.34**	**0.28**
100	25	25	0.19	0.23%	1.27	0.51	0.64	0.50
		50	0.23	0.15%	2.40	0.86	0.53	0.42
		75	0.29	0.13%	2.54	1.14	0.56	0.45
	50	25	0.23	0.07%	1.15	0.67	0.52	0.43
		50	0.12	0.05%	2.77	1.01	0.57	0.47
		75	0.12	0.05%	2.02	1.10	0.59	0.48
	75	25	0.17	0.02%	3.12	1.75	0.57	0.47
		50	0.17	0.01%	1.86	1.37	0.65	0.55
		75	0.16	0.01%	8.99	6.25	0.54	0.47
		mean	**0.19**	**0.08%**	**2.9**	**1.63**	**0.58**	**0.47**
300	75	25	3.51	0.15%	19.48	6.70	15.26	12.40
		50	3.56	0.09%	41.78	8.21	7.04	5.38
		75	3.79	0.08%	25.15	10.80	6.89	5.60
	150	25	1.66	0.05%	11.45	8.33	16.25	12.64
		50	0.92	0.02%	13.78	11.93	12.04	9.92
		75	0.74	0.04%	66.12	26.30	7.30	6.01
	225	25	2.68	0.02%	58.09	50.18	8.43	6.74
		50	4.01	0.01%	82.77	79.92	7.84	6.53
		75	4.45	0.01%	38.14	38.07	10.06	7.19
		mean	**2.81**	**0.05%**	**39.64**	**25.15**	**10.12**	**8.05**

Table 1 Upper bounds: Average results for randomly generated instances of k-cluster. Five problems are tested for each (n, k, d). All computing times are in seconds.

2. Fast initial decrease. Our solver SDLS (as well as SB) has a very fast initial convergence, as illustrated on Figure 2. This is also a highly desirable property in the context of branch-and-bound. In particular, the improvement of the bound computed by SDLS is big in the first iterations.
3. Low-memory. We also mention that our solver uses little memory: for instance, it requires about 4 MB to solve (14) when $n = 100$.

The conclusion of these first experiments is that the new SDP bounds have an advantageous trade-off between tightness and speed of computation, together with interesting features in view of exact resolution.

5 Exact resolution: solving k-cluster to optimality

We use the solver to compute $\Theta(\alpha)$ as the bounding procedure of a branch-and-bound for solving k-cluster problems to optimality. As presented in Section 5.1, our branch-and-bound algorithm is very basic; the novelty is essentially the use of $\Theta(\alpha)$. In Section 5.2, we compare it with [BEP09] the best known method to solve k-cluster (that mixes nicely CPLEX and quadratic relaxations parameterized with SDP).

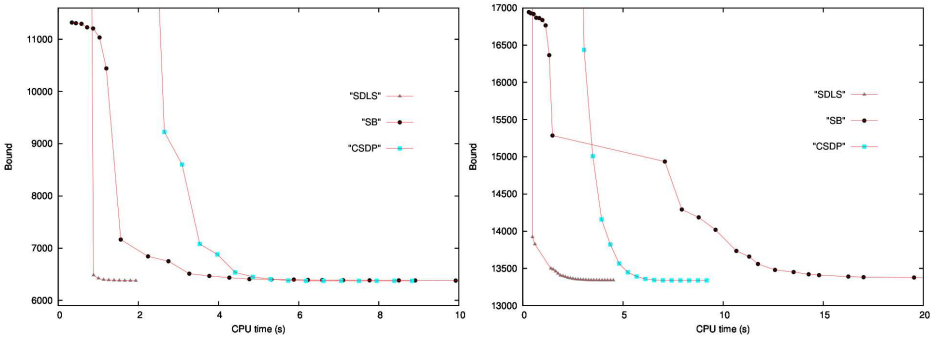

Fig. 2 Comparison of the CPU time of the three solvers for two instances of the k-cluster problem with $n = 300$, $d = 0.5$, and $k = 150$ (left side) $k = 225$ (right side)

5.1 Branch-and-bound for k-cluster

We summarize in this section the main ingredients of our branch-and-bound algorithm.

Heuristics for initial feasible point. Good feasible points give good lower bounds at the beginning of the branch-and-bound, and this yields better pruning. Following [Pis06], we apply the following two-step heuristic that gives very good feasible points:

1. Apply the natural greedy algorithm (see e.g. [AITT00]) consisting in removing $n-k$ vertices from G by choosing successively the vertex with the minimum degree in the reduced graph.
2. Enhance the feasible point by a local search which consists in swapping two nodes (one in the k-cluster and not the other) until no improvement is possible.

Branching strategy. For simplicity, the branching scheme is based on a basic depth-first search. Moreover, we have to choose, at each node of the search tree, a vertex to add to the cluster or to exclude from it (in other words, we want to fix one of the $\{-1, 1\}$-variables). Again for simplicity, we use a branching order fixed in advance, as follows. As in [Pis06], we estimate of the expected contribution of each node, by solving for each i

$$D_i = \max \left\{ \sum_{j \in V \setminus \{i\}} w_{ij} z_j^i \ : \ \sum_{j \in V \setminus \{i\}} z_j^i = p - 1, \ z_j^i \in \{0, 1\}, \ j \in V \setminus \{i\} \right\}.$$

This problem asks to choose $p-1$ largest values among the edges connected to i – which can be done efficiently by a median search algorithm. Then we order the variables with decreasing D_i. Numerical testing has showed us that this choice is competitive with other approaches (such as branching on the most fractional variable).

Branching step. Branching on the index i_0 consists in decomposing (14) into two smaller instances, as follows. To obtain the new objective function, we remove the i_0th line from Q, and add it (or subtract if the vertex is excluded) to the first one. We do the same for the i_0th column. We also remove two constraints: $\langle E_{i0}, X \rangle = 1$

and $\langle Q_{i0}, X\rangle = 4k - 2n$. Finally we use the previous optimal y to make a warm-start for the two reduced problems: we keep the current values associated to the remaining constraints. We observed that this speeds up the solving process of a mean factor of 3.

Early termination. Within branch-and-bound algorithms, the early termination of the bounding procedure often provides a substantial overall speed-up (see e.g. [FR05] and [RRW10]). In our case, the next observation is important, so we formalize it as a lemma.

Lemma 2 *For any variable $y \in \mathbb{R}^{2n+2}$ and any $\alpha > 0$, the value $\alpha((n+1)^2/2 + f(y))$ is a bound for the k-cluster problem.*

Proof Just combine (13) and (15) to get that

$$\mathrm{val}\,(1) \leq \Theta(\alpha) \leq \alpha\Big(\frac{(n+1)^2}{2} + f(y)\Big).$$

Thus we get a bound of the optimal value of (1). □

As any differentiable optimization algorithm, the algorithm to minimize f is stopped when an approximate solution is computed; more precisely when

$$\|\nabla f(y_k)\| = \|\mathcal{A}X(y_k) - b\| \leq \varepsilon$$

with the fixed tolerance ε. Though the computed $X(y_k)$ does not satisfies perfectly the constraints, the lemma guarantees that we still get a bound for our problem with y_k. In fact, we can stop the algorithm anytime before convergence, and we still get a bound. This is very useful within the branch-and-bound: we can stop the bounding procedure when the bound gets lower than the previous threshold. More precisely, we use the following early termination rule. Let β be the current best lower bound given by a k-subgraph; then we stop the run of the algorithm when

$$(n+1)^2/2 + f(y) < (\beta + 1)/\alpha.$$

Fixing α and ε. For simplicity, we keep the level-parameter α and the stopping criterion ε constant. We chose these parameters empirically: we did numerical tests on some instances (as those of Section 4, see also Figure 1), we observed that $\varepsilon = 10^{-7}$ and $\alpha = 10^{-4}$ seem to be good values, and we take those values for all the tests. Dynamic adjustement of those parameters might provide additional speed-ups, but we postpone this technical point for further research.

5.2 Experiments on solving k-cluster to optimality: comparison

The best approach for k-cluster. As far as we are aware, the currently strongest results are obtained by the convex quadratic relaxation procedure of [BEP09] (see also [BE06], and [Pla06]), that we briefly recall here. Start with the initial formulation (2), and observe that for any u, γ, the problem is equivalent to

$$\begin{cases} \max \quad z^\top Q(u,\gamma) z \\ \quad e^\top z = k \\ \quad z \in \{0,1\}^n \end{cases} \tag{16}$$

with the quadratic form

$$z^\top Q(u,\gamma)z := \frac{1}{2}\, z^\top W\, z + \sum_{i=1}^{n} u_i(z_i^2 - z_i) + (e^\top z - k)\sum_{i=1}^{n}\gamma_i z_i.$$

Relaxing the integrality constraints gives the upper bound

$$R(u,\gamma) = \begin{cases} \max & z^\top Q(u,\gamma)z \\ & e^\top z = k \\ & z \in [0,1]^n. \end{cases}$$

If u and γ are chosen such that $Q(u,\gamma)$ is negative semidefinite, the bound $R(u,\gamma)$ is obtained by solving a convex quadratic problem. In practice, the variables (u^*,γ^*) are chosen such that they minimize the upper bound

$$\begin{cases} \min & R(u,\gamma) \\ & Q(u,\gamma) \preceq 0 \end{cases}$$

which turns out to be a linear SDP problem, more precisely, the dual of (9). Thus a solution of the problem (16) with (u^*,γ^*) is computed by a standard solver for convex quadratic 0-1 problem, more precisely a branch-and-bound algorithm using $R(u^*,\gamma^*)$ as bounding procedure. The interest is then that they can advantageously use CPLEX implementing the state-of-the-art for mixed-integer quadratic solver.

Numerical results. We solve to optimality, on the same computer, the same test-problems as in [BEP09] (see also [Pla06]). Table 2 presents the numerical results: we can solve all the instances with comparable times; since we have implemented a very basic branch-and-bound (simple branching strategy and depth-first search), whereas the results of [BEP09] are obtained with the mixed-integer programming solver of CPLEX, this shows the interest of our approach.

Table 2 reports for each method the computing time and the number of nodes. Recall that each entry of the table is the mean over the 5 instances with the same parameters settings. To go beyond average numbers, the table also shows the number of problems (out of 5) solved faster by our method (in the column "# pbs faster").

Let us comment more precisely the comparison of Table 2. We observe that QCR is faster in average than our approach for graphs with $n = 80$ vertices but it is slower when $n = 100$. As in [BEP09], the computation is stopped after one hour: we indicate 'n.a.' if the corresponding problems were not solved within the time limit, and we indicate '(x)' next to the computing time when we were able to solve only 'x' instances (out of 5) in less within the time limit.

We did not include the results for small graphs ($n = 40$) that are in [BEP09]: for those graphs, QCR clearly outperforms our method (by a factor between 5 and 10), but both methods converge very quickly (less than few seconds). In that case, the tightness of the bound is actually not crucial, and it seems like there is no need to spend time in computing semidefinite bounds.

Though the two methods are comparable with respect to computing times, they have opposite strategies. QCR does many (cheap) bound evaluations, whereas our algorithm computes more expensive bounds and prunes better: we enumerate 15-20 times fewer nodes on average. The bound of [BEP09] is of SDP-quality at the root of the branch-and-bound tree only, but deteriorates down in the tree, while $\Theta(\alpha)$ is

of SDP-quality everywhere, without paying the whole price in computing time. Thus our method is particularly well-adapted for the instances with small d and k; these are the most difficult instances, where good pruning is essential. Moreover, the increase of number of nodes with increase of dimension is less dramatic for our method than for QCR (passing from $n = 80$ to $n = 100$, the number is multiplied by 2 instead of 3).

Finally, we point out that the computing time of our method is not strongly correlated to the number of the evaluated nodes in the search tree. This is a consequence of our solver ability to be interrupted during the solving process.

n	k	$d(\%)$	B&B using $\Theta(\alpha)$				B&B of [BEP09]	
			time (sec.)	# nodes	# pbs faster	gap(%) (root)	time (sec.)	# nodes
80	20	25	186.6	14869	1/5	9.43	72.4	207562
		50	674.2	43702	0/5	8.19	325.6	527300
		75	1809.4	132455	0/5	6.60	1322.6	2170499
	40	25	28.5	1426	1/5	2.76	19.6	31442
		50	53.6	6774	0/5	1.86	24.8	36544
		75	413.4	19784	0/5	1.39	176.9	276198
	60	25	3.69	200	0/5	0.90	0.92	1699
		50	6.91	330	0/5	0.62	2.09	4358
		75	20.44	1098	0/5	0.43	3.60	7592
		mean	**355.2**	**24515**			**216.5**	**362577**
100	25	25	1443 (3)	155658	2/3	9.45	1483 (3)	2499727
		50	1802 (2)	172912	2/2	8.10	1978 (1)	3510428
		75	n.a.	n.a.	n.a.	5.75	n.a.	n.a.
	50	25	314.2	13532	4/5	2.63	473.6	504644
		50	349 (1)	12020	1/1	2.23	392 (1)	396780
		75	704.2	29503	2/5	1.09	548.4	580512
	80	25	16.08	565	0/5	0.86	9.00	12996
		50	72.09	2547	0/5	0.68	50.81	74196
		75	40.49	2662	0/5	0.36	14.32	21045
		mean	**592.6**	**48675**			**668.6**	**950041**

Table 2 Computing times (in seconds) for exact resolution: comparison on the test-problems of [BEP09] and [Pla06].

5.3 Experiments on solving k-cluster to optimality: larger problems

We have conducted numerical experiments on problems beyond the actual state of the art. Table 3 reports the results for instances with $n = 120$.

For those large graphs, we change the experimental protocol, as follows. We do not compare with QCR (the largest tests of [BEP09] and [Pla06] are with $n = 100$). So we do need anymore to use the same computer as their: we run this third experiment on a Dell precision T7500 Intel Xeon 2.80GHz with 4GB RAM under linux, using single-thread (single core). We fix a computing time limit of 60000 seconds, and we report the number of instances solved for each setting (in the column "# pbs solved").

As far as we know, this is the first time that k-cluster problems with unrestricted graphs of size larger than 100 are solved to optimality. On the other hand, we see that our solver runs into some trouble, especially for the most difficult problems (those with low density). Two reasons could explain those difficuties, and open ways to improve the approach. First, recall that, except for the bounding procedure, our branch-and-bound is very basic. Including more sophisticated strategies on each point of Section 5.1 would probably bring much performance improvement. Second, the bound $\Theta(\alpha)$ is not

tight enough to avoid losing oneself in the branch-and bound tree (see the gap at the root). Considering stronger bounds of the same kind (adding cuts for example) would therefore be interesting. Those two points are subject to current investigations.

n	k	$d(\%)$	B&B using $\Theta(\alpha)$			
			time (sec.)	# nodes in search tree	# pbs solved	gap (%) (root)
120	30	25	57156	964968	1/5	10.00
		50	n.a.	n.a.	0/5	n.a.
		75	n.a.	n.a.	0/5	n.a.
	60	25	16230	223187	3/5	3.26
		50	3716	56101	3/5	2.61
		75	9020	176355	5/5	1.47
	90	25	124	2952	4/5	1.07
		50	12818	179349	5/5	1.19
		75	1845	24549	5/5	1.03

Table 3 Computing times (in seconds) for exact resolution of large-scale instances.

5.4 Conclusions, perspectives

This paper develops a branch-and-bound algorithm using a novel bounding procedure to solve k-cluster problems to optimality. The key is to use new semidefinite bounds for k-cluster that trade some quality of bound for a speed-up of computation time. This is the first successful attempt to solve this NP-hard optimization to optimality with a semidefinite programming approach competitive with state-of-the-art methods.

The numerical experiments of Sections 4.2 and 5.2 show that the new semidefinite bounds $\theta(\alpha)$ have a practical interest. They have indeed good balance between tightness and computing time; they provide SDP-quality bounds while being faster to compute. The dedicated solver for computing $\theta(\alpha)$ also combines advantages of the SDP solvers SB and CSDP: like SB, it gives guaranteed upper bounds, has a fast initial convergence, and allows to be interrupted; and like CSDP it is reliable, in the sense that we observe only small variations in computational times. The branch-and-bound using $\theta(\alpha)$ takes advantage of SDP-like bounds (all way long) to prune well. Its performance is comparable with the best method for this problem. In practice, our method works particularly fine on the most difficult instances of k-cluster (with a large number of vertices, small density and small k).

The exact resolution scheme presented here could be adapted to other combinatorial problems: the semidefinite bounds upon which the approach is based are indeed introduced in [Mal07] for general binary quadratic problems. The first step toward a generalization would be to extend the numerical study of Section 4.2 (and [MR10a]); this is what proposes our recent work [MR10b] (which also presents a different derivation of the SDP bounds).

Acknowledgement

This work was supported by CNRS (through "GdR Recherche Opérationnelle") and Grenoble University (Université Joseph Fourier, through "Pôle Math-STIC"). We thank Quentin Monnet and Lise-Marie Veillon (Master students of ENSIIE, Evry, France) for their valuable help in developing parts of the solver. We also thank the two anonymous referees for numerous suggestions about a preliminary version of this article.

References

[AITT00] Y. Asahiro, K. Iwama, H. Tamaki, and T. Tokuyama. Greedily finding a dense subgraph. Journal of Algorithms, 34(2):203–221, 2000.

[BE06] A. Billionnet and S. Elloumi. Using a mixed integer quadratic programming solver for the unconstrained quadratic 0-1 problem. Mathematical Programming, 109(1):55–68, 2006.

[BEP09] A. Billionnet, S. Elloumi, and M.-C. Plateau. Improving the performance of standard solvers for quadratic 0-1 programs by a tight convex reformulation: The QCR method. Discrete Applied Mathematics, 157(6):1185–1197, 2009.

[BGLS03] J.F. Bonnans, J.Ch. Gilbert, C. Lemaréchal, and C. Sagastizábal. Numerical Optimization. Springer Verlag, 2003.

[BH08] R. Borsdorf and N. Higham. A preconditionned newton algorithm for the nearest correlation matrix. Submitted, 2008.

[Bil05] A. Billionnet. Different formulations for solving the heaviest k-subgraph problem. Information Systems and Operational Res., 43(3):171–186, 2005.

[BM03] S. Burer and R.D.C Monteiro. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. Mathematical Programming (Series B), 95:329–357, 2003.

[Bor99] B. Borchers. CSDP, a C library for semidefinite programming. Optimization Methods and Software, 11(1):613–623, 1999.

[CP84] D.G. Corneil and Y. Perl. Clustering and domination in perfect graphs. Discrete Applied Mathematics, 9(1):7–39, 1984.

[Erk90] E. Erkut. The discrete p-dispersion problem. European Journal of Operational Research, 46(1):48–60, 1990.

[FKP01] U. Feige, G. Kortsarz, and D. Peleg. The dense k-subgraph problem. Algorithmica, 29(3):410–421, 2001.

[FL01] U. Feige and M. Langberg. Approximation algorithms for maximization problems arising in graph partitioning. Journal of Algorithms, 41(2):174–211, 2001.

[FR05] A. Faye and F. Roupin. A cutting planes algorithm based upon a semidefinite relaxation for the quadratic assignment problem. In ESA 2008, LNCS 3669, pages 850–861, 2005.

[FR07] A. Faye and F. Roupin. Partial lagrangian for general quadratic programming. 4'OR, A Quarterly Journal of Operations Research, 5(1):75–88, 2007.

[GL89] J.Ch. Gilbert and C. Lemaréchal. Some numerical experiments with variable-storage quasi-Newton algorithms. Mathematical Programming, 45:407–435, 1989.

[GW95] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of the ACM, 6:1115–1145, 1995.

[Hel04] C. Helmberg. A C++ implementation of the Spectral Bundle Method. http://www-user.tu-chemnitz.de/~helmberg/SBmethod/, 2004. Version 1.1.3.

[Hig02] N. Higham. Computing a nearest symmetric correlation matrix - a problem from finance. IMA Journal of Numerical Analysis, 22(3):329–343, 2002.

[HR00] C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. SIAM Journal on Optimization, 10(3):673–696, 2000.

[HRVW96] C. Helmberg, F. Rendl, R. Vanderbei, and H. Wolkowicz. An interior point method for semidefinite programming. SIAM Journal on Optimization, 6:342–361, 1996.

[HUL93] J.-B. Hiriart-Urruty and C. Lemaréchal. Convex Analysis and Minimization Algorithms. Springer Verlag, Heidelberg, 1993. Two volumes.

[HYZ02] Q. Han, Y. Ye, and J. Zhang. An improved rounded method and semidefinite programming relaxation for graph partition. Mathematical Programming, 92(3):509–535, 2002.

[JS05] G. Jäger and A. Srivastav. Improved approximation algorithms for maximum graph partitioning problems. Journal of Combinatorial Optimization, 10(2):133–167, 2005.

[KB91] J. M. Keil and T. B. Brecht. The complexity of clustering in planar graphs. J. Combinatorial Mathematics and Combinatorial Computing, 9:155–159, 1991.

[Kho05] Subhash Khot. Ruling out ptas for graph min-bisection, dense k-subgraph, and bipartite clique. SIAM J. Comput, 36:1025–1071, 2005.

[KPP02] J. Krarup, D. Pisinger, and F. Plastria. Discrete location problems with push-pull objectives. Discrete Applied Mathematics, 123:363–378, 2002.

[KS07] M. Kočvara and M. Stingl. On the solution of large-scale sdp problems by the modified barrier method using iterative solvers. Math. Program., 109(2):413–444, 2007.

[LMZ08] M. Liazi, I. Milis, and V. Zissimopoulos. A constant approximation algorithm for the densest k-subgraph problem on chordal graphs. Inf. Process. Lett., 108(1):29–32, 2008.

[LO99] C. Lemaréchal and F. Oustry. Semidefinite relaxations and Lagrangian duality with application to combinatorial optimization. Research report 3710, INRIA, 1999.

[Lov79] L. Lovász. On the Shannon capacity of a graph. IEEE Transactions on Information Theory, IT 25:1–7, 1979.

[Mal04] J. Malick. A dual approach to semidefinite least-squares problems. SIAM Journal on Matrix Analysis and Applications, 26, Number 1:272–284, 2004.

[Mal07] J. Malick. Spherical constraint in Boolean quadratic programming. Journal of Global Optimization, 39(4):609–622, 2007.

[MPRW09] J. Malick, J. Povh, F. Rendl, and A. Wiegele. Regularization methods for semidefinite programming. SIAM Journal on Optimization, 20(1):336–356, 2009.

[MR10a] J. Malick and F. Roupin. Numerical study of SDP bounds for the k-cluster problem. E. Notes on Discrete Mathematics, 36(1):399–406, 2010. Proceedings of ISCO 2010.

[MR10b] J. Malick and F. Roupin. On the bridge between combinatorial optimization and nonlinear optimization: a family of semidefinite bounds leading to newton-like methods. Submitted, 2010.

[Nes05] Y. Nesterov. Smooth minimization of non-smooth functions. Mathematical Programming, 103(1), 2005.

[NW99] J. Nocedal and S. Wright. Numerical Optimization. Springer Verlag, NY, 1999.

[Pis06] D. Pisinger. Upper bounds and exact algorithms for p-dispersion problems. Computers and Operations Research, 33:1380–1398, 2006.

[Pla06] M.-C. Plateau. Reformulations quadratiques convexes pour la programmation quadratique en variables 0-1. PhD thesis, Conservatoire National des Arts et Métiers (CNAM, Paris), 2006.

[PS83] Y. Perl and Y. Shiloach. Efficient optimization of monotonic functions on trees. SIAM Journal on Algebraic and Discrete Methods, 4(4):512–516, 1983.

[QS06] H. Qi and D. Sun. Quadratic convergence and numerical experiments of Newton's method for computing the nearest correlation matrix. SIAM Journal on Matrix Analysis and Applications, 28:360–385, 2006.

[Rou04] F. Roupin. Form linear to semidefinite programming: an algorithm to obtain semidefinite relaxations for bivalent quadratic problems. Journal of Combinatorial Optimization, 8(4), 2004.

[RRW10] F. Rendl, G. Rinaldi, and A. Wiegele. Solving max-cut to optimality by intersecting semidefinite and polyedral relaxations. Math. Programming, 121:307–335, 2010.

[TA05] A. Martin T. Achterberg, T. Koch. Branching rules revisited. Operational Research Letters, 33(1):342–54, 2005.

[Tod01] M.J. Todd. Semidefinite optimization. Acta Numerica, 10:515–560, 2001.

[TTT06] R.H Tutuncu, K.C. Toh, and M.J. Todd. Inexact primal-dual path-following algorithms for a special class of convex quadratic SDP and related problems. Pacific Journal on Optimization, 2006.

Appendix: Formulation of the k-cluster as a quadratic $\{-1, 1\}$-problem

The spherical constraints appears more easily on purely quadratic $\{-1, 1\}$-optimization problems (see [Mal07]). The second reformulation in Section 2 considers the transformation of the k-cluster problem, from the natural modeling as a quadratic $\{0, 1\}$-problem (with linear and quadratic constraints), to a purely quadratic $\{-1, 1\}$-problem. We specify here this transformation, that uses standard techniques.

Lemma 3 *With the notation of this section, the k-cluster problem* (1) *is equivalent to the quadratic problem in dimension* $n+1$

$$\begin{cases} \max \quad x^\top Q\, x \\ \quad x^\top Q_j\, x = 4k - 2n, \quad j \in \{0, \ldots, n\} \\ \quad x \in \{-1, 1\}^{n+1} \end{cases} \tag{17}$$

where the symmetric $(n+1) \times (n+1)$*-matrices* Q *and* Q_j $(j \in \{0, \ldots, n\})$ *are defined by* (5)*. More precisely, this equivalence means that the optimal values of* (1) *and* (17) *are the same and that the optimal solutions coincide as follows:*

- *if* $\bar{z}$ *is a solution of* (1) *then* $\bar{x} = (1, 2\bar{z} - e)$ *is a solution of* (17)*;*
- *if* $\bar{x}$ *is a solution of* (17)*, then* $\bar{z} = ((\bar{x}_0\bar{x}_1, \ldots, \bar{x}_0\bar{x}_n) + e)/2$ *is a solution of* (1)*.*

Proof Operate first the change of variable $x = 2z - e$, and express the objective and the constraints with respect to $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$. Just develop the objective

$$z^\top W z = \frac{(x+e)^\top}{2} W \frac{(x+e)}{2} = \frac{1}{4}(x^\top W x + 2x^\top W e + e^\top W e),$$

and similarly transform $e^\top z = k$ as $e^\top x = 2k - n$, and for all $j = 1, \ldots, n$

$$z^\top C_j z = 2kz_j \iff x^\top C_j x + 2x^\top (C_j e - 2ke_j) = 4k - e^\top C_j e \iff x^\top C_j x + 2x^\top \tilde{e}_j = 4k - 2n$$

the last equivalence coming from $e^\top C_j e = 2n$ and $\tilde{e}_j = C_j e - 2ke_j$. So we get quadratic problem in $x \in \{-1, 1\}^n$

$$\begin{cases} \max \quad \frac{1}{4}(x^\top W x + 2x^\top W e + e^\top W e) \\ \quad e^\top x = 2k - n \\ \quad x^\top C_j x + 2x^\top \tilde{e}_j = 4k - 2n, \quad j \in \{1, \ldots, n\} \\ \quad x_i \in \{-1, 1\} \quad i \in \{1, \ldots, n\}. \end{cases} \tag{18}$$

The formulation of (3) is equivalent to (18) in the sense that the optimal values are the same and the solutions are in one-to-one correspondance with $x = 2z - e$.

We consider now that the following purely quadratic problem with the additional variable $x_0 \in \{-1, 1\}$

$$\begin{cases} \max \quad \frac{1}{4}(x^\top W x + 2x^\top W e\, x_0 + e^\top W e\, {x_0}^2) \\ \quad e^\top x\, x_0 = 2k - n \\ \quad x^\top C_j x + 2x^\top \tilde{e}_j\, x_0 = 4k - 2n, \quad j \in \{1, \ldots, n\} \\ \quad x_i \in \{-1, 1\} \quad i \in \{0, \ldots, n\}. \end{cases} \tag{19}$$

We observe that this problem is equivalent to (17) in view of the definitions of the matrices in (5). So we just have to establish that (18) is equivalent to (19); we do so in two steps. If $(x_1, \ldots, x_n)$ is feasible in (18), then obviously $(1, x)$ and $(-1, -x)$ are both feasible in (19), with same objective value. It follows that $\mathrm{val}\,(19) \leq \mathrm{val}\,(18)$. Conversely if $(x_0, \ldots, x_n)$ is feasible in (19), then $x_0 = \pm 1$ and $(x_0x_1, \ldots, x_0x_n)$ is feasible in (18) with the same objective value. It follows that $\mathrm{val}\,(18) \leq (19)$, so that we have the equality in fact. The relation between the argmins then becomes clear. □