HAL open science

Improved semidefinite branch-and-bound algorithm for k-cluster

Nathan Krislock, Jérôme Malick, Frédéric Roupin

To cite this version:

Nathan Krislock, Jérôme Malick, Frédéric Roupin. Improved semidefinite branch-and-bound algorithm for k-cluster. 2012. hal-00717212v3

HAL Id: hal-00717212

https://inria.hal.science/hal-00717212v3

Preprint submitted on 13 Jul 2012 (v3), last revised 6 Feb 2015 (v6)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Improved semidefinite branch-and-bound algorithm for k-cluster

Nathan Krislock * Jérôme Malick † Frédéric Roupin ‡

July 13, 2012

Abstract

This paper presents a method to solve k-cluster problems by intersecting semidefinite and polyhedral relaxations. We use a generic branch-and-bound method featuring an improved semidefinite bounding procedure. Extensive numerical experiments show that our algorithm outperforms the best known methods in both time and ability to solve large instances. For the first time, numerical results are reported for k-cluster problems on unstructured graphs with 140 and 160 vertices.

Keywords combinatorial optimization, semidefinite programming, quasi-Newton algorithm, triangle inequalities

1 Introduction

Given a graph $G = (V, E)$, the *k-cluster problem* consists of determining a subset $S \subseteq V$ of k vertices such that the sum of the weights of the edges between vertices in S is maximized. This is a classical problem of combinatorial optimization, also known under the names of "heaviest k-subgraph problem", "k-dispersion problem", and "k-defence-sum problem". Letting $n = |V|$ denote the number of vertices, and w_{ij} denote the edge weight for $ij \in E$ and $w_{ij} = 0$ for $ij \notin E$, the problem can be modeled as the optimization problem

$$\text{(KC)} \qquad \begin{array}{ll} \text{maximize} & \frac{1}{2} z^T W z \\ \text{subject to} & \sum_{i=1}^n z_i = k \\ & z \in \{0,1\}^n, \end{array} \tag{1}$$

where $W := (w_{ij})_{ij} \in \mathbb{S}^n$ is the weighted adjacency matrix of the graph G. It is well-known that problem (1) is NP-hard; see the introduction of [9] for references. Currently one of the best methods for solving problem (1) to optimality is the semidefinite-based branch-and-bound algorithm of [9], which compared positively to the convex quadratic relaxation method of [2, 3].

Our recent work [7] presents an improved semidefinite bounding procedure for Max-Cut, another classical combinatorial optimization problem. Max-Cut problems can be written as maximizing a quadratic function over the vertices of an hypercube, that is, as (1) but without the linear equality constraint $\sum_{i=1}^n z_i = k$.

In this current paper, which is primarily a computational paper, we build upon both [9] and [7] by adapting and extending the bounding procedure of [7] for the k-cluster problem. Extending techniques to k-cluster that have proven to be effective for Max-Cut may be conceptually easy, but the presence of the additional linear constraint causes complications from both theoretical

*INRIA Grenoble Rhône-Alpes, nathan.krislock@inria.fr
†CNRS, Lab. J. Kunztmann, Grenoble, jerome.malick@inria.fr
‡LIPN - CNRS UMR7030 - Université Paris-Nord, roupin@lipn.univ-paris13.fr

and practical points of view. The extensive numerical experiments of this paper show that the resulting algorithm clearly outperforms the methods of [2, 3, 9], which are the best existing methods to solve the k-cluster problem to optimality. Our algorithm is also able to solve unstructured k-cluster problems of sizes 140 and 160, for which, as far as we are aware, no numerical results have been reported in the literature.

In Section 2 we describe our improved semidefinite bounding procedure for the k-cluster problem (1). In Section 3 we describe our branch-and-bound implementation using our improved semidefinite bounding procedure for solving k-cluster problems to optimality. In Section 4 we present our numerical results. Finally, we give our concluding remarks in Section 5.

2 Improved semidefinite bounding procedure for k-cluster

In this section we describe the improved bounding procedure that is based on semidefinite programming (SDP) bounds of k-cluster. We start by briefly recalling the standard strengthened SDP relaxation of k-cluster that we will approximate with our bounds.

2.1 Strengthened semidefinite relaxation

The inner product of two matrices X and Y is defined as $\langle X, Y\rangle := \operatorname{trace}(X^T Y)$. The notation $X \succeq 0$ means that X is symmetric positive semidefinite. The following SDP problem gives a bound for the k-cluster problem (1):

$$(\mathrm{SDP}_I) \qquad \begin{array}{ll} \text{maximize} & \langle Q, X\rangle \\ \text{subject to} & \langle Q_j, X\rangle = 4k - 2n, \quad j \in \{0, \ldots, n\}, \\ & \operatorname{diag}(X) = e, \;\; X \succeq 0, \\ & A_I(X) \geq -e. \end{array} \tag{2}$$

For the sake of brevity, we will only describe the SDP relaxation (2) very generally; for more details about the formulation of this SDP relaxation we refer the interested reader to [9]. In problem (2), the matrix X lies in $\mathbb{S}^{n+1}$. The vector of all ones is denoted by e (we let the dimension of e depend on the context), and we have

$$Q := \frac{1}{4}\begin{bmatrix} e^T W e & e^T W \\ We & W \end{bmatrix}, \qquad Q_j := \begin{bmatrix} 0 & e^T + (n-2k)e_j^T \\ e + (n-2k)e_j & e_j e^T + e e_j^T \end{bmatrix}, \quad j \in \{0, \ldots, n\},$$

with $e_j \in \mathbb{R}^n$ being the j-th column of the $n \times n$ identity matrix, for $j \in \{1, \ldots, n\}$, and $e_0 := 0 \in \mathbb{R}^n$. The $j = 0$ constraint in problem (2) comes from the constraint $\sum_{i=1}^n z_i = k$ in the original k-cluster problem (1). In addition, we add the reinforcing equality constraints corresponding to $j \in \{1, \ldots, n\}$ that come from including the redundant product constraints $\sum_{i=1}^n z_i z_j = k z_j$, for $j \in \{1, \ldots, n\}$, to the original problem before forming the SDP relaxation. It is known that these reinforcing equality constraints provide the tightest possible SDP bound when considering only the inclusion of valid *equality* constraints (see, e.g., [6]). We further strengthen the bound by adding valid *inequality* constraints: we add reinforcing triangle inequalities, defined for $1 \leq i < j < k \leq n+1$ by

$$\begin{aligned} X_{ij} + X_{ik} + X_{jk} &\geq -1, \\ X_{ij} - X_{ik} - X_{jk} &\geq -1, \\ -X_{ij} + X_{ik} - X_{jk} &\geq -1, \\ -X_{ij} - X_{ik} + X_{jk} &\geq -1. \end{aligned}$$

Since there are a large number, $4\binom{n+1}{3}$, of triangle inequalities in total, we will iteratively add a subset of the most violated inequalities. For a subset of triangle inequalities I, we let $A_I \colon \mathbb{S}^n \to \mathbb{R}^{|I|}$ be the corresponding linear function describing the inequalities in this subset. For every set of triangle inequalities I, the SDP relaxation in problem (2) gives us an upper bound on the value of the maximum weight k-cluster:

$$(\mathrm{KC}) \leq (\mathrm{SDP}_I). \tag{3}$$

As was shown in [12], the (SDP_I) bound with all triangle inequalities (i.e., $|I| = 4\binom{n+1}{3}$) is tight in that it achieves the optimal value of the k-cluster problem on some instances, but it is also expensive to compute. Instead of using the (SDP_I) bound directly, our approach here is based upon the semidefinite bounds of [8] that we present in the next section.

Unlike the SDP relaxation of Max-Cut used in [7], the SDP relaxation (2) is not strictly feasible. This is formalized in the next easy result.

Lemma 1. *The semidefinite relaxation* (2) *of the k-cluster problem is not strictly feasible.*

Proof. Let X be feasible for problem (2). Let $v := \begin{bmatrix} 4k-2n \\ -2e \end{bmatrix} \in \mathbb{R}^{n+1}$. We will show that $Xv = 0$. By partitioning X consistently with the vector v, we have

$$Xv = \begin{bmatrix} X_{11} & X_{21}^T \\ X_{21} & X_{22} \end{bmatrix} \begin{bmatrix} 4k-2n \\ -2e \end{bmatrix} = \begin{bmatrix} (4k-2n)X_{11} - 2X_{21}^T e \\ (4k-2n)X_{21} - 2X_{22}e \end{bmatrix}.$$

Since $\mathrm{diag}(X) = e$, we have $X_{11} = 1$. The $j = 0$ constraint can be written as

$$\left\langle \begin{bmatrix} 0 & e^T \\ e & 0 \end{bmatrix}, \begin{bmatrix} X_{11} & X_{21}^T \\ X_{21} & X_{22} \end{bmatrix} \right\rangle = 4k - 2n.$$

Simplifying we get $2X_{21}^T e = 4k - 2n$. Therefore,

$$(4k-2n)\, X_{11} - 2X_{21}^T e = (4k-2n) - (4k-2n) \; = 0.$$

Next we let $j \in \{1, \ldots, n\}$ and consider constraint j:

$$\left\langle \begin{bmatrix} 0 & e^T + (n-2k)e_j^T \\ e + (n-2k)e_j & e_j e^T + e e_j^T \end{bmatrix}, \begin{bmatrix} X_{11} & X_{21}^T \\ X_{21} & X_{22} \end{bmatrix} \right\rangle = 4k - 2n.$$

Simplifying we get

$$2X_{21}^T e - (4k-2n)e_j^T X_{21} + 2e_j^T X_{22} e = 4k - 2n.$$

Since $2X_{21}^T e = 4k - 2n$, we have

$$(4k-2n)e_j^T X_{21} - 2e_j^T X_{22} e = 0.$$

Since this holds for all $j \in \{1, \ldots, n\}$, we have

$$(4k-2n)X_{21} - 2X_{22}e = 0.$$

Thus we can conclude that

$$Xv = \begin{bmatrix} (4k-2n)X_{11} - 2X_{21}^T e \\ (4k-2n)X_{21} - 2X_{22}e \end{bmatrix} = 0.$$

Since $v \neq 0$, we have that X is not positive definite. Therefore, problem (2) is not strictly feasible. $\square$

2.2 Our semidefinite bounds for k-cluster

We now summarize some useful results about the nonlinear SDP bounds we use to solve k-cluster to optimality. These bounds are based on the general nonlinear SDP bounds of [8], but are specialized here for the k-cluster problem.

Let us first introduce some notation. We gather all the equality constraints in problem (2) together as $B(X) = b$, where $B\colon \mathbb{S}^{n+1} \to \mathbb{R}^{2n+2}$ is a linear function and $b \in \mathbb{R}^{2n+2}$. For any matrix A, we denote by $\|A\|_F$ the *Frobenius norm* of A, which is defined as $\|A\|_F := \sqrt{\langle A, A\rangle}$. For a real number a, we denote its *nonnegative part* by $a_+ = \max\{a, 0\}$. We extend this definition to vectors and matrices as follows: for $x \in \mathbb{R}^n$, we define $(x_+)_i := (x_i)_+$, for $i = 1, \ldots, n$, and for a given symmetric matrix A, we define $A_+ := U \operatorname{Diag}(\lambda_+)U^T$, where an eigendecomposition of A is given by $A = U \operatorname{Diag}(\lambda)U^T$, with eigenvalues $\lambda \in \mathbb{R}^n$ and orthogonal matrix $U \in \mathbb{R}^{n\times n}$. We denote similarly a_-, x_-, and A_-.

Proposition 1 (Our semidefinite bounds for k-cluster)**.** *Let I be a set of triangle inequalities. For $y \in \mathbb{R}^{2n+2}$ and $z \in \mathbb{R}^{|I|}$, let the matrix $X_I(y, z) \in \mathbb{S}^{n+1}$ be defined by*

$$X_I(y, z) := [Q - B^*(y) + A_I^*(z)]_+ .$$

Let $\alpha > 0$ and let $F_I^\alpha\colon \mathbb{R}^{2n+2} \times \mathbb{R}^{|I|} \to \mathbb{R}$ be the function defined by

$$F_I^\alpha(y, z) := \frac{1}{2\alpha} \|X_I(y, z)\|_F^2 + b^T y + e^T z + \frac{\alpha}{2}(n + 1)^2. \tag{4}$$

Then, for all $y \in \mathbb{R}^{2n+2}$ and $z \in \mathbb{R}_+^{|I|}$, we have that $F_I^\alpha(y, z)$ is a valid upper bound for problem (1)*; that is,*

$$(\mathrm{KC}) \le F_I^\alpha(y, z), \qquad \textit{for all } y \in \mathbb{R}^{2n+2},\ z \in \mathbb{R}_+^{|I|}. \tag{5}$$

Proof. It is clear (see, e.g., [7, Lemma 1]) that if $X \in \mathbb{S}^{n+1}$ satisfies $\operatorname{diag}(X) = e$ and $X \succeq 0$, then $\|X\|_F \le n + 1$. Therefore, since $\alpha > 0$, the following problem

$$(\mathrm{SDP}_I^\alpha) \qquad \begin{array}{ll} \text{maximize} & \langle Q, X\rangle + \frac{\alpha}{2}\left((n+1)^2 - \|X\|_F^2\right) \\ \text{subject to} & B(X) = b,\ \ A_I(X) \ge -e,\ \ X \succeq 0, \end{array}$$

satisfies

$$(\mathrm{SDP}_I) \le (\mathrm{SDP}_I^\alpha). \tag{6}$$

The Lagrangian of the (SDP_I^α) problem is given by

$$\begin{aligned} L(X; y, z) &:= \langle Q, X\rangle + \frac{\alpha}{2}\left((n+1)^2 - \|X\|_F^2\right) + \langle y, b - B(X)\rangle + \langle z, e + A_I(X)\rangle \\ &= \langle Q - B^*(y) + A_I^*(z), X\rangle - \frac{\alpha}{2}\|X\|_F^2 + b^T y + e^T z + \frac{\alpha}{2}(n+1)^2, \end{aligned}$$

where $y \in \mathbb{R}^{2n+2}$ and $z \in \mathbb{R}_+^{|I|}$. By [8, Theorem 2] we have that F_I^α is the corresponding dual function; that is,

$$F_I^\alpha(y, z) = \max_{X\succeq 0} L(X; y, z).$$

Then, by weak duality we have

$$(\mathrm{SDP}_I^\alpha) \le F_I^\alpha(y, z), \qquad \text{for all } y \in \mathbb{R}^{2n+2},\ z \in \mathbb{R}_+^{|I|}. \tag{7}$$

Therefore, combining inequalities (3), (6), and (7), we obtain inequality (5). □

The bounds $F_I^\alpha(y,z)$ coincide, up to change of sign and notation, with the bounds $\Theta(\lambda,\mu,\alpha)$ of [8]. For fixed $\alpha > 0$ and set of triangle inequalities I, the best possible bound $F_I^\alpha(y,z)$ can be found by solving the problem

$$\begin{array}{ll} \text{minimize} & F_I^\alpha(y,z) \\ \text{subject to} & y \text{ free}, \;\; z \geq 0. \end{array}$$

The following proposition gives us an important fact that supports the practical use of these bounds: the above problem is a convex and differentiable problem.

Proposition 2 (Differentiability)**.** *The function F_I^α is convex and differentiable; its gradients are given by*

$$\nabla_y F_I^\alpha(y,z) = b - \frac{1}{\alpha} B\left(X_I(y,z)\right), \quad \textit{and} \quad \nabla_z F_I^\alpha(y,z) = e + \frac{1}{\alpha} A_I\left(X_I(y,z)\right). \tag{8}$$

Proof. From the proof of Proposition 1, the function F_I^α can be interpreted as a dual function, which immediately implies its convexity. The differentiability of F_I^α follows from [8, Theorem 2]. □

This smoothness allows us to use a quasi-Newton method that can handle bound constraints, such as L-BFGS-B [4, 10], to efficiently minimize $F_I^\alpha(y,z)$ over y and $z \geq 0$. For the management of the different parameters, we closely follow the bounding procedure of [7] for Max-Cut: we minimize $F_I^\alpha(y,z)$ with increasing accuracy (the stopping tolerance ε of the quasi-Newton algorithm is driven to 0) while gradually adding inequalities and reducing α. The sketch of the our bounding procedure is given in Algorithm 1; for more details, we refer to [7, Algorithm 1].

Algorithm 1 Sketch of bounding procedure for k-cluster

Input: Scalars $\alpha_1 > 0$, $\varepsilon_1 > 0$, and initial set of triangle inequalities $I_1 = \emptyset$

for $k = 1, 2, \ldots$ **do**

Use L-BFGS-B to compute (y_k, z_k) such that

$$\max\left\{ \left\| b - B\left(X_k\right) \right\|_\infty, \; \left\| \left[e + A_I\left(X_k\right) \right]_- \right\|_\infty \right\} < \varepsilon_k, \qquad \text{where} \qquad X_k \leftarrow \frac{1}{\alpha_k} X_{I_k}(y_k, z_k).$$

Remove inequalities I_k^- that are not active, and add some inequalities I_k^+ that are violated by X_k:

$$I_{k+1} \leftarrow (I_k \setminus I_k^-) \cup I_k^+.$$

if the number of inequalities added $|I_k^+|$ is small **then**

Decrease α_k and ε_k

end if

end for

Since problem (2) is not strictly feasible (see Lemma 1), some of the theoretical convergence results of [7] cannot be applied here, and difficulties can also arise when trying to solve problem (2) numerically. However, since the bounds presented in [8] for general quadratic optimization problems are derived from weak duality, we only needed the feasibility of problem (2) to obtain the bounds we use here, as we saw in the proof of Proposition 1. In addition, in our numerical tests, we did not observe any great difficulties when running our bounding procedure in Algorithm 1.

For illustration, we plot the convergence curve for our bounding procedure in Figure 1 on a k-cluster problem ($n = 160$, edge density 25%, $k = 80$). We can see in Figure 1 the benefit of adding triangle inequalities and reducing the value of α since it allows us to attain a much lower bound.

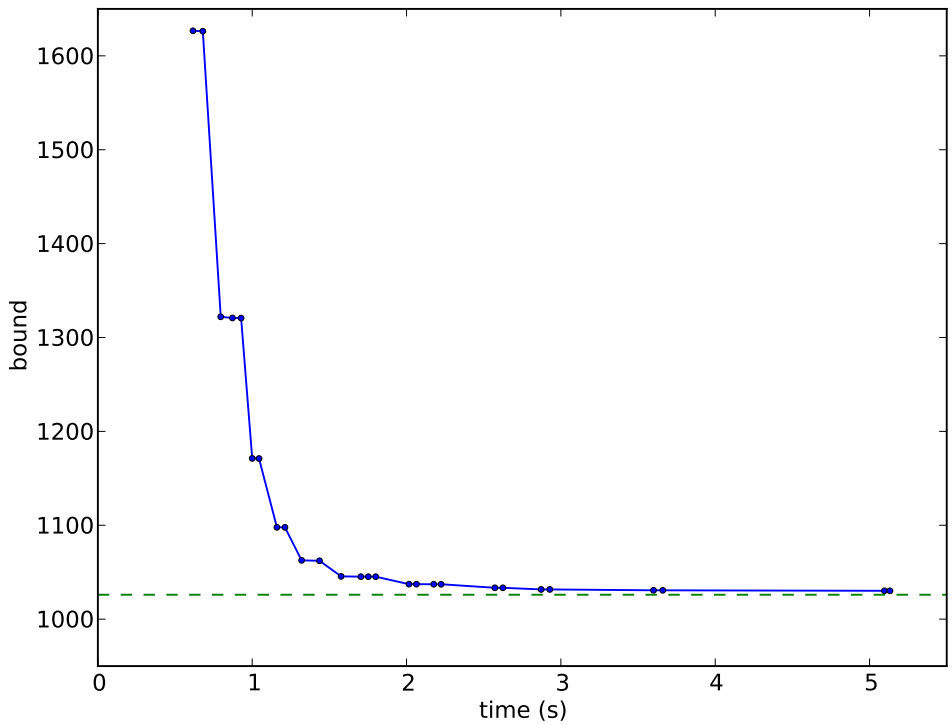

Figure 1: Time/bound plot of the Algorithm 1 bounding procedure on problem PB3_n160_d025_k80 which has optimal value 1026.

3 Branch-and-bound method for k-cluster

In this section, we describe our method for solving k-cluster to optimality. We list the main ingredients of our branch-and-bound implementation, and briefly compare them with the branch-and-bound implementation used in [9].

- **Bounds.** We have described our bounding procedure in Algorithm 1 in the previous section. We now discuss the differences between the bounding procedure of Algorithm 1 and the one used in [9]. With our notation, the bounds used in the branch-and-bound algorithm of [9] are given by (see [9, Lemma 2]):

$$\Theta(y,\alpha) := \frac{\alpha}{2}\left\|[Q/\alpha + B^*(y)]_+\right\|_F^2 - \alpha b^T y + \frac{\alpha}{2}(n+1)^2, \qquad \text{for } \alpha > 0. \tag{9}$$

  Observe therefore that we have $\Theta(y,\alpha) = F_\emptyset^\alpha(-\alpha y)$ for all y and $\alpha > 0$. Thus, the bounds used in [9] essentially correspond (up to a scaling of the vector y) to the bounds that we use here, but with no triangle inequalities ($I = \emptyset$). The presence (and the efficient management) of the inequalities in Algorithm 1 is the key difference between our bounding procedure and the one of [9]. Another important difference is that the values of α and ε are kept fixed in the bounding procedure of [9], whereas we gradually reduce the values of α and ε to zero in the bounding procedure in Algorithm 1. Thus in practice, both bounding procedures (ours and the one of [9]) use quasi-Newton algorithms, but here we have to use a quasi-Newton solver that can handle bound constraints, and we also have to run this solver several times (see Algorithm 1).

- **Heuristics.** We use the same two heuristic methods in the current branch-and-bound implementation as in the one of [9]:

  1. For the initial feasible point, we use the classical two-step greedy heuristic for k-cluster, since it gives very good feasible solutions.
  2. After running the bounding procedure on a subproblem having k' nodes added to the cluster, we add the remaining $k - k'$ nodes having the largest fractional values.

- **Branching rules.** In the current branch-and-bound implementation, we use the "difficult first" branching rule to decide which variables to branch on next: in the $\{0,1\}$ problem formulation, we select the variable having fractional value closest to $\frac{1}{2}$. This is the same branching rule that was used in [9]. In addition, given a list of subproblems in the branch-and-bound method, we must decide which subproblem to branch on next. We use the BOB Branch & Bound platform [5] that provides an easy and flexible way to implement a branch-and-bound algorithm. The BOB platform automatically handles the management of subproblems, and we select the subproblem with the weakest bound with the hope of making the most progress. In contrast, a basic depth-first traversal of the branch-and-bound tree was used for the selection of subproblems in [9].

4 Numerical results

In our tests we used a Dell T-1600 (using a single core) with 8 GB of memory running the Linux operating system. We implemented our algorithm in C / FORTRAN and have used the Intel Math Kernel Library (MKL) for the eigenvalue computations.

As far as we are aware, the only challenging k-cluster test-problems publicly available are the ones created by the graph generator `rudy`, used by [1, 3, 9] among others. The parameters of the

Table 1: The number of nodes and CPU time, averaged over five instances for each triple (n, k, d), required to solve a library of 270 k-cluster problems. Entire set of results available online at http://hal.archives-ouvertes.fr/hal-00717212.

(a) Smaller problems

n	k	$d(\%)$	nodes	CPU time (s)
40	10	25	1.0	0.08
		50	1.0	0.07
		75	1.8	1.24
	20	25	1.4	0.11
		50	1.8	0.24
		75	1.0	0.08
	30	25	1.0	0.11
		50	1.0	0.10
		75	1.0	0.09
80	20	25	3.4	3.15
		50	7.4	6.14
		75	13.8	9.51
	40	25	1.4	1.10
		50	1.0	0.82
		75	6.6	7.98
	60	25	1.0	1.09
		50	1.0	0.81
		75	1.0	0.66
100	25	25	20.6	31.27
		50	35.0	41.63
		75	30.6	32.27
	50	25	3.4	5.25
		50	25.4	46.12
		75	1.4	2.09
	75	25	1.0	1.88
		50	1.8	3.82
		75	1.0	1.63

(b) Larger problems

n	k	$d(\%)$	nodes	CPU time (s)
120	30	25	119.4	315.80
		50	194.2	425.13
		75	422.2	889.83
	60	25	59.8	198.53
		50	85.8	263.22
		75	43.0	143.00
	90	25	1.8	6.78
		50	22.2	96.93
		75	1.0	3.03
140	35	25	366.2	1165.82
		50	1063.4	2888.61
		75	1558.6	4079.52
	70	25	134.2	542.98
		50	780.6	3035.34
		75	52.2	202.94
	105	25	2.6	14.06
		50	11.0	61.51
		75	6.6	34.78
160	40	25	744.6	2856.42
		50	11325.4	37565.20
		75	8050.6	26302.60
	80	25	395.4	1835.20
		50	993.4	4654.52
		75	3829.0	18653.90
	120	25	31.4	219.82
		50	17.4	143.42
		75	9.8	82.15

instances of k-cluster are the size of the graph n, the value of k, and the graph density d. The instances are randomly generated as follows: given a density $d \in [0,1]$, a uniformly distributed 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. We take:

$$n = 40, 80, 100, 120, 140, 160, \qquad k = \frac{1}{4}n, \frac{1}{2}n, \frac{3}{4}n, \qquad d = 25\%, 50\%, 75\%.$$

We have five instances of k-cluster problems for each set of parameters giving us a total of 270 instances. We report the average number of nodes and time required to solve each set of five problems in Table 1.

The instances with $n = 40, 80, 100$ have already been used in several articles about k-cluster, such as [1, 3]. The instances with $n = 120$ have been used by [9]. As far as we are aware, this paper is the first one where numerical results are reported for instances with $n = 140$ and $n = 160$.

We have made the entire dataset of problems available together with their optimal values and the full numerical results of our tests at:

http://hal.archives-ouvertes.fr/hal-00717212.

These extensive numerical experiments show that our algorithm clearly outperforms the methods of [2, 3, 9], previously the best existing methods to solve the k-cluster problem to optimality. Comparing to the numerical results reported in [9], we solve the problems in the test set on average 80 times faster than previous approaches, and up to 615 times faster. We refer to the results of [9] without re-running them because the results are so much further ahead that it is clearly not due to using a faster machine – in fact, the machine we are using for our tests in this paper is only about ten times faster than the machine used in [9]. Our algorithm is also able to solve unstructured k-cluster problems of sizes $n = 140$ and $n = 160$, for which, as far as we are aware, no numerical results have been reported in the literature. Finally we note that 96.6% of the 270 test problems in are solved within three hours.

5 Conclusions

In this primarily computational paper, we have presented an improved branch-and-bound algorithm to solve k-cluster problems to optimality. Our method is based on the previous work of [9] and [7] and here we have summarized the main ideas we have borrowed from these papers and have highlighted some of the differences.

The main reason the method presented here is better than the results reported in [9] is the introduction of triangle inequalities and the strategy of reducing α and ϵ in the semidefinite bounding procedure presented in Algorithm 1. The result is that we do not have to visit a lot of nodes in the branch-and-bound search tree; for example, we have found that 55% of the problems with size $n \leq 120$ are solved at the root of the branch-and-bound tree.

Since the semidefinite relaxation (2) of the k-cluster problem is not strictly feasible, we will consider in a future work the use of semidefinite facial reduction (see, e.g., [11]) with the hope of improving even more the time to solve k-cluster problems to optimality.

References

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

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

[3] Billionnet, A., Elloumi, S., Plateau, M.C.: 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)

[4] Byrd, R.H., Lu, P., Nocedal, J., Zhu, C.: A limited memory algorithm for bound constrained optimization. SIAM Journal on Scientific Computing **16**(5), 1190–1208 (1995)

[5] Cun, B.L., Roucairol, C., The PNN Team: BOB: A unified platform for implementing branch-and-bound like algorithms. Tech. Rep. 95/16, University of Versailles Saint-Quentin-en-Yvelines (1995)

[6] Faye, A., Roupin, F.: Partial Lagrangian relaxation for general quadratic programming. 4OR: A Quarterly Journal of Operations Research **5**, 75–88 (2007)

[7] Krislock, N., Malick, J., Roupin, F.: Improved semidefinite bounding procedure for solving Max-Cut problems to optimality. Tech. Rep. HAL-00665968 (2012)

[8] Malick, J., Roupin, F.: On the bridge between combinatorial optimization and nonlinear optimization: a family of semidefinite bounds leading to Newton-like methods. To appear in Mathematical Programming (2011). Available online as preprint hal-00662367

[9] Malick, J., Roupin, F.: Solving k-cluster problems to optimality with semidefinite programming. To appear in Math. Programming B, special issue on Mixed-Integer Nonlinear Programming (2011)

[10] Morales, J.L., Nocedal, J.: Remark on "Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound constrained optimization". ACM Trans. Math. Softw. **38**(1), 1–4 (2011)

[11] Ramana, M.V., Tunçel, L., Wolkowicz, H.: Strong duality for semidefinite programming. SIAM Journal on Optimization **7**(3), 641–662 (1997)

[12] Roupin, F.: From linear to semidefinite programming: An algorithm to obtain semidefinite relaxations for bivalent quadratic problems. Journal of Combinatorial Optimization **8**(4), 469–493 (2004)