Computational Optimization and Applications (2022) 82:251–291
https://doi.org/10.1007/s10589-022-00355-1

SDP-based bounds for graph partition via extended ADMM

Angelika Wiegele[1] · **Shudian Zhao**[1]

Received: 19 May 2021 / Accepted: 3 February 2022 / Published online: 17 March 2022

Abstract
We study two NP-complete graph partition problems, k-equipartition problems and graph partition problems with knapsack constraints (GPKC). We introduce tight SDP relaxations with nonnegativity constraints to get lower bounds, the SDP relaxations are solved by an extended alternating direction method of multipliers (ADMM). In this way, we obtain high quality lower bounds for k-equipartition on large instances up to $n = 1000$ vertices within as few as 5 min and for GPKC problems up to $n = 500$ vertices within as little as 1 h. On the other hand, interior point methods fail to solve instances from $n = 300$ due to memory requirements. We also design heuristics to generate upper bounds from the SDP solutions, giving us tighter upper bounds than other methods proposed in the literature with low computational expense.

Keywords Graph partitioning · Semidefinite programming · ADMM · Combinatorial optimization

1 Introduction

Graph partition problems have gained importance recently due to their applications in the area of engineering and computer science such as telecommunication [16] and parallel computing [12]. The solution of a graph partition problem would serve to partition the vertices of a graph $G(V, E)$ into several groups under certain constraints for capacity or cardinality in each group. The optimal solution is expected to have the smallest total weight of cut edges. This problem is NP-complete [8]. Previous

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement MINOA No 764759.

✉ Shudian Zhao
shudian.zhao@aau.at

Angelika Wiegele
angelika.wiegele@aau.at

[1] Institut für Mathematik, Alpen-Adria-Universität Klagenfurt, Universitätsstraße 65-67, 9020 Klagenfurt, Austria

studies worked on improving quadratic programming or linear programming formulations to reduce the computational expense with commercial solvers [6]. Relaxations are also used to approximate this problem. Garey et al. [8] were the first to use eigenvalue and eigenvector information to get relaxations for graph partition. Ghaddar et al. [9] have recently used a branch-and-cut algorithm based on SDP relaxations to compute global optimal solutions of k-partition problems.

The k-equipartition problem, which is to find a partition with the minimal total weight of cut edges and the vertex set V is equally partitioned into k groups, is one of the most popular graph partition problems. Another problem that interests us is the graph partition problem under knapsack constraints (GPKC). In the GPKC, each vertex of the graph network is assigned a weight and the knapsack constraint needs to be satisfied in each group.

Lisser and Rendl [16] compared various semidefinite programming (SDP) relaxations and linear programming (LP) relaxations for k-equipartition problems. They showed that the nonnegativity constraints become dominating in the SDP relaxation when k increases. However, the application of this formulation is limited by the computing power of SDP solvers, because adding all the sign constraints for a symmetric matrix of dimension n causes $O(n^2)$ new constraints and entails a huge computational burden especially for large instances. Nguyen [19] proposed a tight LP relaxation for GPKC problems and a heuristic to build upper bounds as well. Semidefinite programming has shown advantages in generating tight lower bounds for quadratic problems with knapsack constraints [11] and k-equipartition problem, but so far there have been no attempts to apply SDP relaxations to GPKC.

Algorithms for solving SDPs have been intensively studied in the previous years. Malick et al. [18] designed the boundary point method to solve SDP problems with equations. It can solve instances with a huge number of constraints that interior point methods (IPMs) fail to solve. This method falls into the class of alternating direction method of multipliers (ADMM). ADMM has been studied in the area of convex optimization and been proved of linear convergence when one of the objective terms is strongly convex [20]. In recent years, there have been studies focusing on generalizing this idea on solving convex optimizations with more blocks of variables. Chen et al. [3], for example, have proved the convergence of 3-block ADMM on certain scenarios, but the question as to whether the direct extension 3-block ADMM on SDP problems is convergent is still open.

There have also been varied ideas about combining other approaches with ADMM for solving SDP problems. De Santis et al. [4] added the dual factorization in the ADMM update scheme, while Sun et al. [23] combined ADMM with Newton's methods. Both attempts have improved the performance of the algorithms.

1.1 Main results and outline

In this paper, we will introduce an extended ADMM algorithm and apply it to the tight SDP relaxations for graph partition problems with nonnegativity constraints. We will also introduce heuristics to obtain a feasible partition from the solution of the SDP relaxation.

This paper is structured as follows. In Sect. 2, we will introduce two graph partition problems, the k-equipartition problem and the graph partition problems with knapsack constraints (GPKC). We will discuss different SDP relaxations for both problems. In Sect. 3, we will design an extended ADMM and illustrate its advantages in solving large SDP problems with nonnegativity constraints. In Sect. 4, we will introduce two post-processing methods used to generate lower bounds using the output from the extended ADMM. In Sect. 5, we will design heuristics to build a tight upper bound from the SDP solution to address the original problem. Numerical results of experiments carried out on graphs with different sizes and densities will be presented in Sect. 6. Section 7 concludes the paper.

1.2 Notation

We define by e_n the vector of all ones of length n, by $\mathbf{0}_n$ the vector of all zeros of length n and by $\mathbf{0}_{n\times n}$ the square matrix of all zeros of dimension n. We omit the subscript in case the dimension is clear from the context. The notation $[n]$ stands for the set of integers $\{1,\dots,n\}$. Let $\mathcal{S}^n$ denote the set of all $n\times n$ real symmetric matrices. We denote by $M \succeq 0$ that the matrix M is positive semidefinite and let $\mathcal{S}^n_+$ be the set of all positive semidefinite matrices of order $n \times n$. We denote by $\langle\cdot,\cdot\rangle$ the trace inner product. That is, for any $M, N \in \mathbb{R}^{n\times n}$, we define $\langle M, N\rangle := \operatorname{trace}(M^\top N)$. Its associated norm is the Frobenius norm, denoted by $\|M\|_F := \sqrt{\operatorname{trace}(M^\top M)}$. We denote by $\operatorname{diag}(M)$ the operation of getting the diagonal entries of matrix M as a vector. The projection on the cone of positive semidefinite matrices is denoted by $\mathcal{P}_{\succeq 0}(\cdot)$. The projection onto the interval $[L, U]$ is denoted by $\mathcal{P}_{[L,U]}$. We denote by $\lambda(\cdot)$ the eigenvalues. That is, for any $M \in \mathbb{R}^{n\times n}$, we define $\lambda(M)$ the set of all eigenvalues of M. Also, we denote $\lambda_{\max}(\cdot)$ the largest eigenvalue. We denote by $x \sim U(0,1)$ a variable x from uniform distribution between 0 and 1. We define by $\operatorname{argmaxk}(\cdot, s)$ the index set of the s largest elements.

2 Graph partition problems

2.1 *k*-equipartition problem

For a graph $G(V, E)$, the k-equipartition problem is the problem of finding an equipartition of the vertices in V with k groups that has the minimal total weight of edges cut by this partition. The problem can be described with binary variables,

$$\begin{aligned} \min\ & \frac{1}{2}\langle L, YY^\top\rangle \\ \text{s.t. } & Ye_k = e_n, \\ & Y^\top e_n = me_k, \\ & Y_{ij} \in \{0,1\}, \forall i \in [n], j \in [k], \end{aligned} \tag{1}$$

where L is the Laplacian matrix for G, variable $Y \in \mathbb{R}^{n\times k}$ indicates which group each vertex is assigned to and e_n (resp. e_k) is the all-one vector of dimension n (resp. k).

This problem is NP-hard and Lisser and Rendl [16] proposed the SDP relaxation

$$\begin{aligned} \min\ & \frac{1}{2}\langle L, X\rangle \\ \text{s.t. } & \operatorname{diag}(X) = e, \\ & Xe = me, \\ & X \succeq 0, \end{aligned} \tag{2}$$

where $X \in \mathcal{S}_n$, and $e \in \mathbb{R}^n$ is the all-one vector.

To tighten this SDP relaxation, we can add more inequalities to problem (2). Here, we introduce two common inequalities for SDP relaxations derived from 0/1 problems.

The process of relaxing $YY^\top$ to X implies that X is a nonnegative matrix, hence the first group of inequalities we consider is $X \geq 0$ and the corresponding new SDP relaxation is

$$\begin{aligned} \min\ & \frac{1}{2}\langle L, X\rangle \\ \text{s.t. } & \operatorname{diag}(X) = e, \\ & Xe = me, \\ & X \succeq 0, \\ & X \geq 0. \end{aligned} \tag{3}$$

This kind of SDP is also called a doubly nonnegative program (DNN) since the matrix variable is both, positive semidefinite and elementwise nonnegative.

Another observation is the following. For any vertices triple (i, j, k), if vertices i and j are in the same group, and vertices j and k are in the same group, then vertices i and k must be in the same group. This can be modeled by the transitivity constraints [16] given as follows

$$\text{MET} := \{X = (X_{ij}) \mid X_{ij} + X_{ik} \leq 1 + X_{jk}, \forall i, j, k \in [n]\}.$$

The set formed by these inequalities is the so-called metric polytop. Adding the transitivity constraints to the SDP relaxation (3) gives

$$\begin{aligned} \min\ & \frac{1}{2}\langle L, X\rangle \\ \text{s.t. } & \operatorname{diag}(X) = e, \\ & Xe = me, \\ & X \succeq 0, \\ & X \geq 0, \\ & X \in \text{MET}. \end{aligned} \tag{4}$$

2.2 Graph partition problem under knapsack constraints (GPKC)

Given a graph $G(V, E)$ with nonnegative weights on the vertices and a capacity bound W, the GPKC asks to partition the vertices such that the total weight of cut edges is minimized and the total weight of vertices in each group does not exceed the capacity bound W.

A mathematical programming formulation is given as

$$\begin{aligned} \min\ & \frac{1}{2}\langle L, YY^\top\rangle \\ \text{s.t. } & Ye_n = e_n, \\ & Y^\top a \leq We_n, \\ & Y_{ij} \in \{0,1\}^{n\times n}, \forall i \in [n], j \in [n], \end{aligned} \tag{5}$$

where $Y \in \mathbb{R}^{n\times n}$, $a \in \mathbb{R}^n$ is the vertex weight vector and W is the capacity bound. We assume $a_i \leq W\ \forall i \in [n]$, otherwise the problem is infeasible. Again, we can derive the SDP relaxation

$$\begin{aligned} \min\ & \frac{1}{2}\langle L, X\rangle \\ \text{s.t. } & \operatorname{diag}(X) = e, \\ & Xa \leq We, \\ & X \succeq 0. \end{aligned} \tag{6}$$

Similar as the k-equipartition problem, we can tighten the relaxation by imposing sign constraints, i.e.,

$$\begin{aligned} \min\ & \frac{1}{2}\langle L, X\rangle \\ \text{s.t. } & \operatorname{diag}(X) = e, \\ & Xa \leq We, \\ & X \succeq 0, \\ & X \geq 0, \end{aligned} \tag{7}$$

and additionally by imposing the transitivity constraints which gives

$$\begin{aligned} \min\ & \frac{1}{2}\langle L, X\rangle \\ \text{s.t. } & \operatorname{diag}(X) = e, \\ & Xa \leq We, \\ & X \succeq 0, \\ & X \geq 0, \\ & X \in \text{MET}. \end{aligned} \tag{8}$$

3 Extended ADMM

The SDP relaxations introduced in Sect. 2 have a huge number of constraints, even for medium-sized graphs. The total number of sign constraints for X is $O(n^2)$ and adding the constraint $X \in \text{MET}$ in the SDP relaxations causes $3\binom{n}{3}$ extra constraints. Therefore, solving these tight relaxations is out of reach for state-of-the-art algorithms like interior point methods (IPMs). However, finding high quality lower bounds by tight SDP relaxations for graph partition problems motivates us to develop an efficient algorithm that can deal with SDP problems with inequalities and sign constraints on large-scale instances. Since the 2-block alternating direction method of multiplier (ADMM) has shown efficiency in solving large-scale instances that interior point methods fail to solve, we are encouraged to extend this algorithm for SDP problems with inequalities in the form

$$\begin{aligned} \min\ & \langle C, X\rangle \\ \text{s.t. } & \mathcal{A}(X) = b, \\ & \mathcal{B}(X) = s, \\ & X \succeq 0, \\ & L \le X \le U, \\ & l \le s \le u, \end{aligned} \tag{9}$$

where $C \in \mathcal{S}^n$, $\mathcal{A} : \mathcal{S}^n \to \mathbb{R}^m$, $\mathcal{B} : \mathcal{S}^n \to \mathbb{R}^q$, $b \in \mathbb{R}^m$, $l, u \in \mathbb{R}^q$. We have the slack variable $s \in \mathbb{R}^q$ to form the inequality constraints, and l and u can be set to $-\infty$ and $+\infty$ respectively. Also, $L \in \mathcal{S}^n$ and $U \in \mathcal{S}^n$ can be symmetric matrices filled with all elements as $-\infty$ and $+\infty$ respectively. That makes formulation (9) able to represent SDP problems with any equality and inequality constraints. This formulation is inspired by the work of [23]. All semidefinite programs given above fit into this formulation. E.g., in (3) operator $\mathcal{A}$ includes the diagonal-constraint and the constraint $Xe = me$, the operator $\mathcal{B}$ as well as the variables s are not present, L is the matrix of all zeros and U the matrix having $+\infty$ everywhere.

Following the ideas for the 2-block ADMM [18], we form the update scheme in Algorithm 1 to solve the dual of problem (9).

Lemma 1 *The dual problem for (9) is given as*

$$\begin{aligned} \max\ & b^\top y + \mathcal{F}_1(S) + \mathcal{F}_2(v) \\ \text{s.t. } & \mathcal{A}^* y + \mathcal{B}^* \bar{y} + S + Z = C, \\ & \bar{y} = v, \\ & Z \succeq 0, \end{aligned} \tag{10}$$

where $\mathcal{F}_1(S) = \inf_W \{\langle S, W\rangle \mid L \le W \le U\}$ *and* $\mathcal{F}_2(v) = \inf_\omega \{\langle v, \omega\rangle \mid l \le \omega \le u\}$.

Proof We derive this dual problem by rewriting the primal SDP problem (9) in a more explicit way, namely

$$\begin{aligned} \min\ & \langle C, X\rangle \\ \text{s.t. } & \mathcal{A}(X) = b, \\ & \mathcal{B}(X) - s = \mathbf{0}_q, \\ & X \succeq 0, \\ & X \geq L, \\ & -X \geq -U, \\ & s \geq l, \\ & -s \geq -u. \end{aligned} \tag{11}$$

Then, the dual of (11) is

$$\begin{aligned} \max \quad & b^\top y + \mathbf{0}_q^\top \bar{y} + \langle \mathbf{0}_{n\times n}, Z\rangle + \langle L, S_L\rangle - \langle U, S_U\rangle + l^\top v_l - u^\top v_u \\ \text{s.t.} \quad & \mathcal{A}^* y + \mathcal{B}^* \bar{y} + Z + S_L - S_U = C, \\ & -\bar{y} + v_l - v_u = \mathbf{0}_q, \\ & Z \succeq 0, \\ & S_L, S_U, v_l, v_u \geq 0. \end{aligned} \tag{12}$$

The following equivalences hold for each entry of the dual variables S_L and S_U in (12)

$$\begin{aligned} X_{ij} = L_{ij} &\iff S_{L,ij} \neq 0, S_{U,ij} = 0; \\ X_{ij} = U_{ij} &\iff S_{U,ij} \neq 0, S_{L,ji} = 0; \\ L_{ij} < X_{ij} < U_{ij} &\iff S_{L,ij} = S_{U,ij} = 0. \end{aligned}$$

If we let $S := S_L - S_U$, then for each entry of S

$$\forall i \in [n], j \in [n], S_{ij} = \begin{cases} S_{L,ij}, & \text{if } S_{L,ij} \neq 0, \\ -S_{U,ij}, & \text{otherwise.} \end{cases}$$

Thus, in the dual objective function, we have

$$\langle L, S_L\rangle - \langle U, S_U\rangle = \sum_{S_{L,ij}\neq 0} L_{ij} S_{L,ij} - \sum_{S_{L,ij}=0} U_{ij} S_{U,ij}. \tag{13}$$

Expressing $\inf_W \{\langle S, W\rangle \mid L \leq W \leq U\}$ element-wisely gives

$$\inf_{W_{ij}} \{S_{ij} W_{ij} \mid L_{ij} \leq W_{ij} \leq U_{ij}\} = \begin{cases} L_{ij} S_{ij}, & \text{if } S_{ij} \geq 0, \\ U_{ij} S_{ij}, & \text{if } S_{ij} < 0. \end{cases} \tag{14}$$

Combining the observations above, we end up with

$$\langle L, S_L\rangle - \langle U, S_U\rangle = \inf_W \{\langle S, W\rangle \mid L \leq W \leq U\}. \tag{15}$$

Similarly, let $v := v_l - v_u$, then we have

$$l^\top v_l - u^\top v_u = \sum_{v_{l,k}\neq 0} l_k v_{l,k} - \sum_{v_{l,k}=0} u_k v_{u,k} = \inf_{\omega}\{\langle v, \omega\rangle \mid l \le \omega \le u\} \tag{16}$$

Hence, problem (10) is equivalent to (12) and it is the dual of (9). □

We now form the augmented Lagrangian function corresponding to (10).

$$\begin{aligned} \mathcal{L}(y, \bar{y}, Z, S, v; X, s) = \ & b^\top y + \mathcal{F}_1(S) + \mathcal{F}_2(v) \\ & - \langle \mathcal{A}^* y + \mathcal{B}^* \bar{y} + S + Z - C, X\rangle - \langle -\bar{y} + v, s\rangle \\ & - \frac{\sigma}{2}\|\mathcal{A}^* y + \mathcal{B}^* \bar{y} + S + Z - C\|_F^2 - \frac{\sigma}{2}\| - \bar{y} + v\|^2. \end{aligned} \tag{17}$$

The saddle point of this augmented Lagrangian function is

$$(y^*, \bar{y}^*, Z^*, S^*, v^*, X^*, s^*) := \operatorname{arg\,min}_{X,s} \operatorname{arg\,max}_{y,\bar{y},v,Z,S} \mathcal{L}(y, \bar{y}, Z, S, v; X, s), \tag{18}$$

which is also an optimal solution for the primal and dual problems. If both, the primal and the dual problem, have strictly feasible points, then a point $(X, s, y, \bar{y}, Z, S, v)$ is optimal if and only if

$$\mathcal{A}(X) = b, \quad \mathcal{B}(X) = s, \quad L \le X \le U, \quad l \le s \le u, \tag{19a}$$

$$\mathcal{A}^* y + \mathcal{B}^* \bar{y} + S + Z = C, \quad \bar{y} = v, \tag{19b}$$

$$X \succeq 0, \quad Z \succeq 0, \quad \langle X, Z\rangle = 0, \tag{19c}$$

$$(X_{ij} - L_{ij})(U_{ij} - X_{ij})S_{ij} = 0, \ \forall i \in [n] \ \forall j \in [n], \quad L \le X \le U, \tag{19d}$$

$$(s_k - l_k)(u_k - s_k)v_k = 0, \ \forall k \in [q], \quad l \le v \le u. \tag{19e}$$

Remark 1 (19d) and (19e) is derived from the optimality conditions for (11), namely from

$$\begin{aligned} & (X_{ij} - L_{ij})S_{L,ij} = 0, S_{L,ij} \ge 0, X_{ij} \ge L_{ij}, \forall i \in [n] \ \forall j \in [n], \\ & (U_{ij} - X_{ij})S_{U,ij} = 0, S_{U,ij} \ge 0, X_{ij} \le U_{ij}, \forall i \in [n] \ \forall j \in [n], \\ & (s_k - l_k)v_{l,k} = 0, v_{l,k} \ge 0, s_k \ge l_k, \forall k \in [q], \\ & (l_k - s_k)v_{l,k} = 0, v_{l,k} \ge 0, s_k \le u_k, \forall k \in [q]. \end{aligned} \tag{20}$$

With $S = S_L - S_U$ and $v = v_l - v_u$, we obtain (19d) and (19e).

We solve problem (18) coordinatewise, i.e., we optimize only over a block of variables at a time while keeping all other variables fixed. The procedure is outlined in Algorithm 1.

Algorithm 1: Extended ADMM for problem (10)

```
  Initialization: Select $\sigma^k > 0$, $\varepsilon_{tol} > 0$; $k = 0,\ X_0 = 0,\ Z_0 = 0,\ S_0 = 0$;
  while $\max\{\varepsilon_{pc}, \varepsilon_{dc}, \varepsilon_{opt_d}, \varepsilon_{opt_p}, \varepsilon_{pb}\} > \varepsilon_{tol}$ do
1     $(y^{k+1}, \bar{y}^{k+1}) = \arg\min_{(y,\bar{y})} -b^\top y - s^\top \bar{y} + \langle \mathcal{A}^* y + \mathcal{B}^* \bar{y}, X^k \rangle$
        $+ \frac{\sigma^k}{2} \|\mathcal{A}^* y + \mathcal{B}^* \bar{y} + S^k + Z^k - C\|^2 + \frac{\sigma^k}{2} \|v^k - \bar{y}\|^2$;
2     $S^{k+1} = \arg\min_S -\mathcal{F}_1(S) + \langle X^k, S \rangle + \frac{\sigma^k}{2} \|\mathcal{A}^* y^{k+1} + \mathcal{B}^* \bar{y}^{k+1} + S + Z^k - C\|_F^2$ ;
3     $Z^{k+1} = \arg\min_Z \langle X^k, Z \rangle + \frac{\sigma^k}{2} \|\mathcal{A}^* y^{k+1} + \mathcal{B}^* \bar{y}^{k+1} + S^{k+1} + Z - C\|_F^2$,
        $v^{k+1} = \arg\min_v -\mathcal{F}_2(v) + v^\top s^k + \frac{\sigma^k}{2} \|v - \bar{y}^{k+1}\|^2$ ;
4     $X^{k+1} = X^k + \sigma^k (\mathcal{A}^* y^{k+1} + \mathcal{B}^* \bar{y}^{k+1} + S^{k+1} + Z^{k+1} - C)$,
        $s^{k+1} = s^k + \sigma^k (v^{k+1} - \bar{y}^{k+1})$ ;
5     Update infeasibilities $\varepsilon_{dc}$, $\varepsilon_{pc}$, $\varepsilon_{pb}$ ,$\varepsilon_{opt_v}$, $\varepsilon_{opt_m}$;
6     Tune stepsize and obtain $\sigma^{k+1}$ ;
7     $k \leftarrow k + 1$;
8 end
  where $\varepsilon_{dc} := \frac{\|\mathcal{A}^* y + \mathcal{B}^* \bar{y} + Z + S - C\|_F}{1 + \|C\|_F} + \frac{\|-\bar{y} + v\|}{1 + \|y\|}$ , $\varepsilon_{pc} := \frac{\|\mathcal{A}(X) - b\|}{1 + \|b\|} + \frac{\|\mathcal{B}(X) - s\|}{1 + \|s\|}$,
  $\varepsilon_{pb} := \frac{\|X - \mathcal{P}_{[L,U]}(X)\|_F}{1 + \|X\|_F}$, $\varepsilon_{opt_m} := \frac{\|X - \mathcal{P}_{[L,U]}(X - S)\|_F}{1 + \|X\|_F + \|S\|_F}$, $\varepsilon_{opt_v} := \frac{\|v - \mathcal{P}_{[l,u]}(v - s)\|}{1 + \|v\| + \|s\|}$.
```

In Step 1, the minimization over $(y, \bar{y})$, we force the first order optimality conditions to hold, i.e., we set the gradient with respect to $(y, \bar{y})$ to zero and thereby obtain the explicit expression

$$\begin{pmatrix} y^{k+1} \\ \bar{y}^{k+1} \end{pmatrix} = \begin{pmatrix} \mathcal{A}\mathcal{A}^* & \mathcal{A}\mathcal{B}^* \\ \mathcal{B}\mathcal{A}^* & \mathcal{B}\mathcal{B}^* + I \end{pmatrix}^{-1} \begin{pmatrix} \frac{b}{\sigma^k} - \mathcal{A}(S^k + Z^k - C + \frac{1}{\sigma^k} X^k) \\ -\mathcal{B}(S^k + Z^k - C + \frac{1}{\sigma^k} X^k) + v^k + \frac{1}{\sigma^k} s^k \end{pmatrix}. \tag{21}$$

Note that the size of y^k is the number of equality constraints and the size of $\bar{y}^k$ the number of inequality constraints. By abuse of notation we write $\mathcal{A}\mathcal{A}^*$ for the matrix product formed by the system matrix underlying the operator $\mathcal{A}(\cdot)$. Similarly for $\mathcal{B}\mathcal{A}^*, \mathcal{B}\mathcal{B}^*$.

In practice, we solve (21) in the following way. First, we apply the Cholesky decomposition

$$RR^\top = \begin{pmatrix} \mathcal{A}\mathcal{A}^* & \mathcal{A}\mathcal{B}^* \\ \mathcal{B}\mathcal{A}^* & \mathcal{B}\mathcal{B}^* + I \end{pmatrix} =: Q. \tag{22}$$

Since $\mathcal{A}$ and $\mathcal{B}$ are row independent, the Cholesky decomposition exists. Moreover, the Cholesky decomposition only needs to be computed once since matrix Q remains the same in all iterations. Then, we update $(y, \tilde{y})$ as

$$RR^\top \begin{pmatrix} y^{k+1} \\ \bar{y}^{k+1} \end{pmatrix} = rhs \tag{23}$$

by solving two systems of equations subsequently, i.e., $R\mathbf{x} = rhs$ and then solve the system $R^\top \mathbf{y} = \mathbf{x}$ and thereby having solved $RR^\top \mathbf{y} = rhs$.

In Step 2, the minimization amounts to a projection onto the non-negative orthant.

$$\begin{aligned}
S^{k+1} &= \arg\min{}_S - \mathcal{F}_1(S) + \langle X^k, S\rangle + \frac{\sigma^k}{2}\|\mathcal{A}^* y^{k+1} + \mathcal{B}^* \bar{y}^{k+1} + S + Z^k - C\|_F^2, \\
S_{ij}^{k+1} &= \begin{cases} \arg\min{}_{S_{ij}\geq 0}\|M_{ij}^{k+1} + S_{ij} - \frac{1}{\sigma^k}L_{ij}\|^2, S_{ij} \geq 0, \\ -\arg\min{}_{S_{ij}\leq 0}\|M_{ij}^{k+1} - S_{ij} + \frac{1}{\sigma^k}U_{ij}\|^2, S_{ij} \leq 0, \end{cases} \\
&= \begin{cases} \mathcal{P}_{\geq 0}(-M_{ij}^{k+1} + \frac{1}{\sigma^k}L_{ij}), S_{ij} \geq 0, \\ -\mathcal{P}_{\leq 0}(M_{ij}^{k+1} + \frac{1}{\sigma^k}U_{ij}), S_{ij} \leq 0, \end{cases} \\
&= \begin{cases} \frac{1}{\sigma^k}\mathcal{P}_{\geq L_{ij}}(\sigma^k M_{ij}^{k+1}) - M_{ij}^{k+1}, S_{ij} \geq 0, \\ \frac{1}{\sigma^k}\mathcal{P}_{\leq U_{ij}}(\sigma^k M_{ij}^{k+1}) - M_{ij}^{k+1}, S_{ij} \leq 0, \end{cases} \\
&= \frac{1}{\sigma^k}\mathcal{P}_{[L_{ij},U_{ij}]}(\sigma^k M_{ij}^{k+1}) - M_{ij}^{k+1},
\end{aligned} \tag{24}$$

where $M^{k+1} := \mathcal{A}^* y^{k+1} + \mathcal{B}^* \bar{y}^{k+1} + Z^k + \frac{1}{\sigma^k}X^k - C$. Hence,

$$S^{k+1} = \frac{1}{\sigma^k}\mathcal{P}_{[L,U]}(\sigma^k M^{k+1}) - M^{k+1}. \tag{25}$$

Similarly, in Step 3 for v^{k+1} we have

$$v^{k+1} = \frac{1}{\sigma^k}\mathcal{P}_{[l,u]}(\sigma^k \bar{y}^{k+1} - s^k) - (\bar{y}^{k+1} - \frac{1}{\sigma^k}s^k) \tag{26}$$

In Step 3, for $Z^{k+1} \succeq 0$, the minimizer is found via a projection onto the cone of positive semidefinite matrices

$$\begin{aligned}
Z^{k+1} &= \arg\min{}_{Z\succeq 0}\langle X^k, Z\rangle + \frac{\sigma^k}{2}\|\mathcal{A}^* y^{k+1} + \mathcal{B}^* \bar{y}^{k+1} + S^{k+1} + Z - C\|_F^2 \\
&= \arg\min{}_{Z\succeq 0}\|\mathcal{A}^* y^{k+1} + \mathcal{B}^* \bar{y}^{k+1} + S^{k+1} + Z + \frac{1}{\sigma^k}X^k - C\|_F^2 \\
&= -\mathcal{P}_{\preceq 0}(N^{k+1}),
\end{aligned} \tag{27}$$

where $N^{k+1} := \mathcal{A}^* y^{k+1} + \mathcal{B}^* \bar{y}^{k+1} + S^{k+1} + \frac{1}{\sigma^k}X^k - C$.

Finally, by substituting Z^{k+1} and v^{k+1} into Step 4 we obtain

$$\begin{aligned}
X^{k+1} &= \sigma^k \mathcal{P}_{\succeq 0}(N^{k+1}), \\
s^{k+1} &= \mathcal{P}_{[l,u]}(\sigma^k \bar{y}^{k+1} - s^k).
\end{aligned} \tag{28}$$

Remark 2 Throughout Algorithm 1, the complementary slackness condition for (X^{k+1}, Z^{k+1}) holds. This is since $\frac{1}{\sigma^k}X^{k+1}$ is the projection onto the positive semidefinite cone of matrix N^{k+1}, while $-Z^{k+1}$ is a projection onto the negative semidefinite cone of the same matrix N^{k+1}.

3.1 Stepsize adjustment

Previous numerical results showed that the practical performance of an ADMM is strongly influenced by the stepsize σ. The most common way is to adjust σ to balance primal and dual infeasibilities: if $\varepsilon_p/\varepsilon_d < c$ for some constant c, then increase σ; if $\varepsilon_p/\varepsilon_d > \frac{1}{c}$, then decrease σ.

Lorenz and Tran-Dinh [17] derived an adaptive stepsize for the Douglas-Rachford Splitting (DRS) scheme. In the setting of the 2-block ADMM, this translates to the ratio between norms of the primal and dual variables $\frac{\|X_k\|}{\|Z_k\|}$ in the k-th iteration. In general, for the 2-block ADMM this update rule yields a better performance than the former one.

In this paper, we use either of these update rules, depending on the type of problems we solve. For SDP problems with equations and nonnegativity constraints only, we apply the adaptive stepsize method from Lorenz and Tran-Dinh [17] since it works very well in practice. However, the situation is different for SDP problems with inequalities different than nonnegativity constraints. In this case, we use the classic method to adjust the stepsize σ according to the ratio between the primal and dual infeasibilities.

4 Lower bound post-processing algorithms

We relax the original graph partition problem to an SDP problem, thereby generating a lower bound on the original problem. However, when solving the SDP by a first-order method, it is hard to reach a solution to high precision in reasonable computational time. Therefore, we stop the ADMM already when a medium precision is reached. In this way, however, the solution obtained by the ADMM is not always a safe underestimate for the optimal solution of the SDP problem. Hence, we need a post-processing algorithm that produces a safe underestimate for the SDP relaxation, which is then also a lower bound for the graph partition problem.

Theorem 1 leads to the first post-processing algorithm. Before stating it, we rewrite Lemma 3.1 from [13] in our context.

Lemma 2 *Let $X, Z \in \mathcal{S}_n$, and $0 \le \lambda(X) \le \bar{x}$, where $\lambda(\cdot)$ indicates the operation of getting the eigenvalues of the respective matrix. Then*

$$\langle X, Z\rangle \ge \sum_{\lambda(Z)<0} \bar{x} \cdot \lambda(Z).$$

Theorem 1 *Given $Z \in \mathcal{S}^n$, $y \in \mathbb{R}^m$, $\tilde{y} \in \mathbb{R}^q$, $\tilde{v} \in \mathbb{R}^q$, $\tilde{S} \in \mathcal{S}^n$, and let $X \in \mathcal{S}^n_+$ be an optimal solution for (9) and $\bar{x} \ge \lambda_{\max}(X)$, then we have a safe lower bound for the optimal value p^**

$$\mathrm{lb} := b^\top \tilde{y} + \mathcal{F}_1(\tilde{S}) + \mathcal{F}_2(\tilde{v}) + \sum_{\lambda(Z)<0} \bar{x}\lambda(Z). \tag{29}$$

Proof We recall the alternative formulation of (9),

$$\begin{aligned} \min\ & \langle C, X\rangle \\ \text{s.t. } & \mathcal{A}(X) = b, \\ & \mathcal{B}(X) - s = 0, \\ & X \succeq 0, \\ & X \geq L, \\ & -X \geq -U, \\ & s \geq l, \\ & -s \geq -u, \end{aligned} \tag{11}$$

and the corresponding dual problem

$$\begin{aligned} \max\quad & b^\top y + \mathbf{0}_q^\top \bar{y} + \langle \mathbf{0}_{n\times n}, Z\rangle + \langle L, S_L\rangle - \langle U, S_U\rangle + l^\top v_l - u^\top v_u \\ \text{s.t.}\quad & \mathcal{A}^* y + \mathcal{B}^* \bar{y} + Z + S_L - S_U = C, \\ & -\bar{y} + v_l - v_u = 0, \\ & Z \succeq 0, \\ & S_L, S_U, v_l, v_u \geq 0. \end{aligned} \tag{12}$$

Given an optimal solution X^* from (11) and the free variable $\tilde{y}$ and nonnegative variables $(\tilde{v}_l, \tilde{v}_u, \tilde{S}_L, \tilde{S}_U)$, we define $Z := C - \mathcal{A}^*\tilde{y} - \mathcal{B}^*\tilde{v}_l + \mathcal{B}^*\tilde{v}_u - S_L + S_U$ and have

$$\langle C, X^*\rangle - (b^\top \tilde{y} + l^\top \tilde{v}_l - u^\top \tilde{v}_u + \langle L, \tilde{S}_L\rangle - \langle U, \tilde{S}_U\rangle), \tag{30a}$$

$$= \langle C, X^*\rangle - \langle \mathcal{A}^*\tilde{y}, X^*\rangle - (l^\top \tilde{v}_l - u^\top \tilde{v}_u + \langle L, \tilde{S}_L\rangle - \langle U, \tilde{S}_U\rangle), \tag{30b}$$

$$\geq \langle C, X^*\rangle - \langle \mathcal{A}^*\tilde{y}, X^*\rangle - \langle \mathcal{B}^*\tilde{v}_l, X^*\rangle + \langle \mathcal{B}^*\tilde{v}_u, X^*\rangle - \langle X, \tilde{S}_L\rangle + \langle U, \tilde{S}_U\rangle, \tag{30c}$$

$$= \langle C - \mathcal{A}^*\tilde{y} - \mathcal{B}^*\tilde{v}_l + \mathcal{B}^*\tilde{v}_u - S_L + S_U, X^*\rangle, \tag{30d}$$

$$= \langle Z, X^*\rangle, \tag{30e}$$

$$\geq \sum_{\lambda(Z)<0} \bar{x}\lambda(Z), \tag{30f}$$

where inequality (30c) holds because $\tilde{v}_l, \tilde{v}_u, \tilde{S}_U$ and $\tilde{S}_L$ are nonnegative. This gives us a lower bound for the problem (11) as

$$\text{lb} := b^\top \tilde{y} + l^\top \tilde{v}_l - u^\top \tilde{v}_u + \langle L, \tilde{S}_L\rangle - U \cdot \tilde{S}_U + \sum_{\lambda(Z)<0} \bar{x}\lambda(Z). \tag{31}$$

On substituting $S := S_L - S_U$ and $v := v_l - v_u$ into the objective function we have

$$\begin{aligned}\langle L, S_L\rangle - \langle U, S_U\rangle &= \inf_W\{\langle S, W\rangle \mid L \le W \le U\},\\ l^\top v_l - u^\top v_u &= \inf_\omega\{\langle v, \omega\rangle \mid l \le \omega \le u\}.\end{aligned} \tag{32}$$

Consequently, we can rewrite (31) as

$$\text{lb} := b^\top \tilde{y} + \mathcal{F}_1(\tilde{S}) + \mathcal{F}_2(\tilde{v}) + \sum_{\lambda(Z)<0} \bar{x}\lambda(Z). \tag{29}$$

□

For specifically structured SDP problems, a value of $\bar{x}$ might be known. Otherwise, without any information about an upper bound $\bar{x}$ in (29) for the maximal eigenvalue $\lambda_{\max}(X)$, we approximate $\bar{x}$ as $\lambda_{\max}(\tilde{X})$ where the output from the extended ADMM is $(\tilde{X}, \tilde{y}, \tilde{v}, \tilde{S})$. Then, we scale it with $\mu > 1$, e.g., $\mu = 1.1$, to have a safe bound $\mu\bar{x}$. Note that this requires that the solution of the extended ADMM, i.e., Algorithm 1, is satisfied with reasonable accuracy, say $\varepsilon = 10^{-5}$.

The complete post-processing algorithm is summarized in Algorithm 2.

Algorithm 2: Rigorous lower bound for (9)

```
Input: Data $P = (\mathcal{A}, \mathcal{B}, b, l, u, L, U, C)$, approximate primal and dual optimal
       solution $(\tilde{X}, \tilde{y}, \tilde{v}, \tilde{S})$ for $P$ and $\bar{x}$ with $\max(\lambda(\tilde{X})) \le \bar{x}$.
Output: Lower bound $d^*$
1 Compute $d_0^* := b^\top \tilde{y} + \mathcal{F}_1(\tilde{S}) + \mathcal{F}_2(\tilde{v})$;
2 if $\min(\lambda(Z)) < 0$ then
3     perturbation $= \bar{x} \cdot \sum_{\lambda(Z)<0} \lambda(Z)$;
4 else
5     perturbation $= 0$;
6 end
7 $d^* := d_0^*$ + perturbation;
8 where $\mathcal{F}_1(S) = \inf_W\{\langle S, W\rangle \mid L \le W \le U\}$, $\mathcal{F}_2(v) = \inf_\omega\{\langle v, \omega\rangle \mid l \le \omega \le u\}$.
```

As for the k-equipartition problem (3), we have $X \preceq m \cdot I$ for any feasible solution X. Hence, we let $\bar{x} = m$ when applying post-processing Algorithm 2 for k-equipartition problems. As for the GPKC, we have no value $\bar{x}$ at hand.

Another way to get a safe lower bound for (9) is to tune the output results and get a feasible solution for its dual problem (10). This is outlined as Algorithm 3. The brief idea is to build a feasible solution $(y_{new}, v_{new}, Z_{new}, S_{new})$ from an approximate solution $(\tilde{y}, \tilde{v}, \tilde{Z}, \tilde{S})$. To guarantee feasibility of $(y_{new}, v_{new}, Z_{new}, S_{new})$, we first get a Z_{new} by projecting $\tilde{Z}$ on the cone of positive semidefinite matrices. We then keep Z_{new} fixed and hence have a linear problem. The final step is to find the optimal solution for this linear programming problem.

In Algorithm 1, the condition $Z \succeq 0$ is guaranteed by the projection operation onto the cone of positive semidefinite matrices. Hence, we can skip Step 1 in Algorithm 3.

We would like to remark that the linear program can be infeasible, but this algorithm works well when the input solution has a good precision. The comparisons of numerical results of these two post processing algorithms are given in Sect. 6.2.

Algorithm 3: Adjusted lower bound for (11)

```
  Input: Data $P = (\mathcal{A}, \mathcal{B}, b, l, u, L, U, C)$, approximate primal and dual optimal
         solution $\tilde{Z}$ for $P$.
  Output: Lower bound $d^*$
1 Update $\tilde{Z}$: $\tilde{Z} \to \mathcal{P}_{\succeq 0}(\tilde{Z})$ ;
2 if LP problem
      $$LP(P) := \max_{v_l, v_u, S_L, S_U \geq 0, y} \{b^\top y + l^\top v_l - u^\top v_u + \langle L, S_L \rangle - \langle U, S_U \rangle \mid \mathcal{A}^* y + \mathcal{B}^* v_l - \mathcal{B}^* v_u + S_L - S_U = C - \tilde{Z}\}$$
   is feasible;
3 then
4     return lower bound $d^* = LP(P)$;
5 else
6     return $d^* = -\infty$;
7 end
```

5 Building upper bounds from the SDP solutions

Computing upper bounds of a minimization problem is typically done via finding feasible solutions of the original problem by heuristics.

A k-equipartition problem can be transformed into a quadratic assignment problem (QAP), and we can find feasible solutions for a QAP by simulated annealing (SA), see, e.g., [22]. However, this method comes with a high computational expense for large graphs. Moreover, it cannot be generalized to GPKC problems.

Here we consider building upper bounds from the optimizer of the SDP relaxations. We apply different rounding strategies to the solution X of the SDP relaxations presented in Sect. 2.

5.1 Randomized algorithm for *k*-equipartition

The first heuristic is a hyperplane rounding algorithm that is inspired by the Goemans and Williamson algorithm for the max-cut problem [10] and Frieze and Jerrum [7]'s improved randomized rounding algorithm for k-cut problems.

Note that the Goemans and Williamson algorithm as well as the Frieze and Jerrum algorithm are designed for cut-problems formed as models on variables in $\{-1/(k-1), 1\}^n$, while our graph partition problems are formed on $\{0, 1\}^n$. Therefore, we need to transform the SDP solutions of problems (3) and (7) before applying the hyperplane rounding procedure. Our hyperplane rounding algorithm for k-equipartition is given in Algorithm 4.

Algorithm 4: Hyperplane rounding algorithm (Hyp) for k-equipartition problem

```
Data: number of partitions $k$, cluster cardinality $m$, number of sampling $M$,
      objective matrix $C$;
Input: Optimal solution $X \in \mathcal{S}^n$ from the SDP relaxation;
Output: $X^* \in \{0,1\}^{n\times n}$, partition $\mathcal{P} := \{P_t \mid t = 1,\dots,k\}$, $b^*_{up}$;
1  Initialization: $P_t \leftarrow \emptyset \ \forall t = 1,\dots,k$, $b^*_{up} = +\infty$;
2  Transformation: $X \leftarrow (kX - ee^\top)/(k-1)$;
3  Get $V$ by matrix decomposition such that $VV^\top = X$;
4  for $iter = 1,\dots,M$ do
5      $r := [r_{ij}] \sim U(0,1)$ ,$\forall i \in [n]$ $j \in [k]$ ;
6      for $t = 1,\dots,k$ do
7          $P_t \leftarrow \text{argmaxk}_{i\in[n]/\cup_{S\in\mathcal{P}}S}(v_i^\top r_t, m)$;
8          for $i \in P_t$ do
9              $X'_{ij} = \begin{cases}1, & j \in P_t,\\ 0, & j \notin P_t;\end{cases}$, $X'_{ji} = \begin{cases}1, & j \in P_t,\\ 0, & j \notin P_t.\end{cases}$
10         end
11     end
12     $b'_{up} = \langle C, X'\rangle$;
13     if $b'_{up} < b^*_{up}$ then
14         $X^* \leftarrow X'$;
15         $b^*_{up} \leftarrow b'_{up}$;
16     end
17 end
18 where $\text{argmaxk}_{i\in I}(a_i, s)$ returns the index set of $s$ largest elements in $a_i$, $\forall i \in I$.
```

5.2 Vector clustering algorithm for *k*-equipartition

We next propose a heuristic via the idea of vector clustering. Given a feasible solution X of (3), we can get $V \in \mathbb{R}^{n\times n}$ with $VV^\top = X$. Let v_i be the i-th row of V and associate it with vertex i in the graph. The problem of building a feasible solution from X can then be interpreted as the problem of clustering vectors $v_1,\dots,v_n$ into k groups. This can be done heuristically as follows.

1. Form a new group with an unassigned vector.
2. Select its $m-1$ closest unassigned neighbors and add them in the same group.
3. Update the status of those vectors as assigned.

This process is repeated $k-1$ times until all vectors are assigned in a group, yielding a k-equipartition for the vertices in V. The details are given in Algorithm 5.

Algorithm 5: Vector clustering algorithm (Vc) for k-equipartition problem

```
   Data: number of partitions $k$, cluster cardinality $m$, maximum iteration number $M$;
   Input: SDP relaxation optimal solution $X \in \mathcal{S}^n$;
   Output: $X^*$, partition $\mathcal{P} := \{P_t \mid t = 1, \ldots, k\}$, $b^*_{up}$;
1  Initialization: $P_t \leftarrow \emptyset \ \forall t = 1, \ldots, k$, $b^*_{up} = +\infty$ .
2  for $iter = 1, \cdots, M$ do
3      for $t = 1, \ldots, k$ do
4          $i = \text{random}\{i \in [n] \mid i \notin \cup_{S \in \mathcal{P}} S\}$,
5          $P_t \leftarrow \{i\}$;
6          $P_t \leftarrow P_t \cup \text{argmaxk}_{j \in [n]/\cup_{S \in \mathcal{P}} S}(x_i^\top x_j, m-1)$ ;
7          for $i \in P_t$ do
8              $X'_{ij} = \begin{cases} 1, & j \in P_t, \\ 0, & j \notin P_t; \end{cases}, \quad X'_{ji} = \begin{cases} 1, & j \in P_t, \\ 0, & j \notin P_t. \end{cases}$
9          end
10     end
11     $b'_{up} = \langle C, X' \rangle$;
12     if $b'_{up} < b^*_{up}$ then
13         $X^* \leftarrow X'$;
14         $b^*_{up} \leftarrow b'_{up}$;
15     end
16 end
17 where $\text{argmaxk}_{i \in I}(a_i, s)$ returns the index set of $s$ largest elements in $a_i$, $\forall i \in I$.
```

5.2.1 Measure closeness between vertices

We explain in this section how we determine the closest neighbor for a vector. The idea of vector clustering is to have vectors with more similarities in the same group. In our setting, we need a measure to define the similarity between two vectors according to the SDP solution.

For a pair of unit vectors v_i, v_j, using the relationship $\cos \measuredangle(v_i, v_j) = v_i^\top v_j$ one can measure the angle between v_i and v_j.

By the setting $VV^\top = X$, we have for any $i \in [n]$

$$x_i = \begin{pmatrix} v_i^\top v_1 \\ \vdots \\ v_i^\top v_n \end{pmatrix} = \begin{pmatrix} \cos \measuredangle(v_i, v_1) \\ \vdots \\ \cos \measuredangle(v_i, v_n) \end{pmatrix}, \tag{33}$$

where x_i is the i-th row vector in X.

Hence, x_i consists of the cosines of the angle between v_i and other vectors. We define $\text{sim}(v_i, v_j) := \sum_{k=1}^n \cos \measuredangle(v_i, v_k) \cos \measuredangle(v_j, v_k) = x_i^\top x_j$ and use this as a measure in Algorithm 5. In other words, we measures the closeness between v_i and v_j by their geometric relationships with other vectors.

In Algorithm 5, we choose a vector as the center of its group and then find vectors surrounding it and assign them to this group.

In each iteration we randomly choose one vector to be the center.

5.3 Vector clustering algorithms for GPKC

Using similar ideas as in Algorithm 5, we construct a rounding algorithm (see Algorithm 6) for GPKC as follows.

1. In each iteration, randomly choose an unassigned vector v_i to start with.
2. Add vectors in the group of v_i in the order according to $\mathrm{sim}(v_i, v_j)$, $\forall j \neq i \in [n]$, until the capacity constraint is violated.
3. If no more vector fits into the group, then this group is completed and we start forming a new group.

Algorithm 6: Vector clustering algorithm (Vc) for GPKC problem

```
Data: vertex weight $a$, knapsack bound $W$, maximum iteration number $M$;
Input: SDP relaxation optimal solution $X$;
Output: $X^* \in \{0,1\}^{n\times n}$, partition $\mathcal{P} := \{P_t \mid t = 1,\dots,n\}$, $b^*_{up}$;
Initialization:$P_t \leftarrow \emptyset\ \forall t = 1,\dots,n$, $V_0 \leftarrow [n]$, $t \leftarrow 1$.
for $iter = 1,\cdots,M$ do
    while $V_0 \neq \emptyset$ do
        $i = \text{random}\{i \in [n] \mid i \notin \cup_{S\in\mathcal{P}} S\}$;
        $P_t \leftarrow \{i\}$;
        $w_t \leftarrow a_i$;
        $w_0 \leftarrow 0$;
        $I \leftarrow \{i \in [n] \mid i \notin \cup_{S\in\mathcal{P}} S\}$;
        for $iter = 1,\dots,|I|$ do
            $j \leftarrow \arg\max_{j\in I} \langle x_j, x_i\rangle$ ;
            $w_0 \leftarrow w_t + a_j$;
            $I \leftarrow I/\{j\}$;
            if $w_0 \leq W$ then
                $w_t \leftarrow w_0$;
                $P_t \leftarrow P_t \cup \{j\}$;
            else
                continue.
            end
        end
        $V_0 \leftarrow V_0 / \cup_{S\in\mathcal{P}} S$;
        for $i \in P_t$ do
            $$X'_{ij} = \begin{cases} 1, & j \in P_t, \\ 0, & j \notin P_t; \end{cases}, \quad X'_{ji} = \begin{cases} 1, & j \in P_t, \\ 0, & j \notin P_t. \end{cases}$$
        end
        $t \leftarrow t + 1$ ;
    end
    $b'_{up} = \langle C, X'\rangle$;
    if $b'_{up} < b^*_{up}$ then
        $X^* \leftarrow X'$;
        $b^*_{up} \leftarrow b'_{up}$;
    end
end
```

5.4 2-opt for graph partition problems

2-opt heuristics are used to boost solution qualities for various combinatorial problems, e.g., TSP [15]. We apply this method after running our rounding algorithms for the graph partition problems to improve the upper bounds. According to the rounding method we choose, the hybrid strategies are named as Hyperplane+2opt (also short as Hyp+2opt) and Vc+2opt for Algorithms 4 and 5, respectively.

The 2-opt heuristic for bisection problems is outlined in Algorithm 7. Given a partition with more than two groups, we apply 2-opt on a pair of groups (P_s, P_t), which is randomly chosen from all groups in the partition, and repeat it on a different pair of groups until no more improvement can be found.

For GPKC, some adjustments are needed because of the capacity constraints. We only traverse among swaps of vertices that still give feasible solutions to find the best swap that improves the objective function value.

Algorithm 7: 2-opt method for bisection problems

```
  Data: Lapacian matrix $L$, threshold $\varepsilon$;
  Input: A feasible bisection $\mathcal{P}_0 = \{P_1, P_2\}$ for graph $G$;
  Output: New bisection $\mathcal{P}^*$ ;
1 $(s,t) \leftarrow \arg\max_{i \in P_1, j \in P_2} \sum_{k \neq i, k \in P_1} L_{jk} - \sum_{k \neq j, k \in P_2} L_{jk} + \sum_{k \neq j, k \in P_2} L_{ik} - \sum_{k \neq i, k \in P_1} L_{ik}$ ;
2 $\Delta_{cost} \leftarrow \sum_{k \neq s, k \in P_1} L_{tk} - \sum_{k \neq t, k \in P_2} L_{tk} + \sum_{k \neq t, k \in P_2} L_{sk} - \sum_{k \neq s, k \in P_1} L_{sk}$;
3 while $\Delta_{cost} > \varepsilon$ do
4     $P_1 \leftarrow P_1 - \{s\} + \{t\}$;
5     $P_2 \leftarrow P_2 - \{t\} + \{s\}$;
6     $(s,t) \leftarrow \arg\max_{i \in P_1, j \in P_2} \sum_{k \neq i, k \in P_1} L_{jk} - \sum_{k \neq j, k \in P_2} L_{jk} + \sum_{k \neq j, k \in P_2} L_{ik} - \sum_{k \neq i, k \in P_1} L_{ik}$;
7     $\Delta_{cost} \leftarrow \sum_{k \neq s, k \in P_1} L_{tk} - \sum_{k \neq t, k \in P_2} L_{tk} + \sum_{k \neq t, k \in P_2} L_{sk} - \sum_{k \neq s, k \in P_1} L_{sk}$;
8 end
9 $\mathcal{P}^* \leftarrow \{P_1, P_2\}$;
```

6 Numerical results

We implemented all the algorithms in MATLAB and run the numerical experiments on a ThinkPad-X1-Carbon-6th with 8 Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz. The maximum iterations for extended ADMM is set to be 20 000 and the stopping tolerance ε_{tol} is set to be 10^{-5} by default.

The code can be downloaded from https://github.com/shudianzhao/ADMM-GP.

6.1 Instances

In order to evaluate the performance of our algorithms, we run numerical experiments on several classes of instances. All instances can be downloaded from https://github.com/shudianzhao/ADMM-GP. The first set of instances for the k-equipartition problem are described in [16], the construction is as follows.

1. Choose edges of a complete graph randomly with probability 20%, 50% and 80%.
2. The nonzero edge weights are integers in the interval (0, 100].
3. Choose the partition numbers as divisors of the graph size n.

We name those three groups of instances rand20, rand50 and rand80, respectively.

Furthermore, we consider instances that have been used in [2]. These are constructed in the following way.

- $G_{|V|,|V|_p}$: Graphs $G(V, E)$, with $|V| \in \{124, 250, 500, 1000\}$ and four individual edge probabilities p. These probabilities were chosen depending on $|V|$, so that the average expected degree of each node was approximately $|V|_p = 2.5, 5, 10, 20$ [14].
- $U_{|V|,|V|_{\pi d^2}}$: For a graph $G(V, E)$, first choose $2|V|$ independent numbers uniformly from the interval (0, 1) and view them as coordinates of $|V|$ nodes on the unit square. Then, an edge is inserted between two vertices if and only if their Euclidian distance is less or equal to some pre-specified value d [14]. Here $|V| \in \{500, 1000\}$ and $|V|_{\pi d^2} \in \{5, 10, 20, 40\}$.
- *mesh*: Instances from finite element meshes; all edge weights are equal to one [5].

For GPKC we generate instances as described in [19]. This is done by the following steps.

1. Generate a random matrix with 20%, 50% and 80% of nonzeroes edge weights between 0 and 100, as vertex weights choose integers from the interval (0, 1000].
2. Determine a feasible solution for this instance for a k-equipartition problem by some heuristic method.
3. Produce 1000 permutations of the vertices in this k-equipartition.
4. Calculate the capacity bound for each instance and select the one such that only 10% of instances are feasible.

We name those three groups of instances GPKCrand20, GPKCrand50 and GPKCrand80, respectively.

6.2 Comparison of Post-processing methods

Our first numerical comparisons evaluate the different post-processing methods used to produce safe lower bounds for the graph partition problems. Recall that in Sect. 4, we introduced Algorithms 2 and 3.

Figure 1 shows how the lower bounds from the post-processing methods evolve as the number of iterations of the extended ADMM increases. We used the DNN relaxation on an instance of the k-equipartition problem of size $n = 100$ and $k = 2$. There are three lines: EB_eADMM represents the lower bounds obtained by the rigorous lower bound method given in Algorithm 3, LpB_eADMM represents the linear programming bound given in Algorithm 2 and dualOfv_eADMM displays the approximate dual objective function value obtained by our extended ADMM. Figure 1a shows that the rigorous error bound method gives tighter bounds in general, while the linear programming bound method is more stable and less affected by the quality of the dual objective function value. The other figures indicate that for small

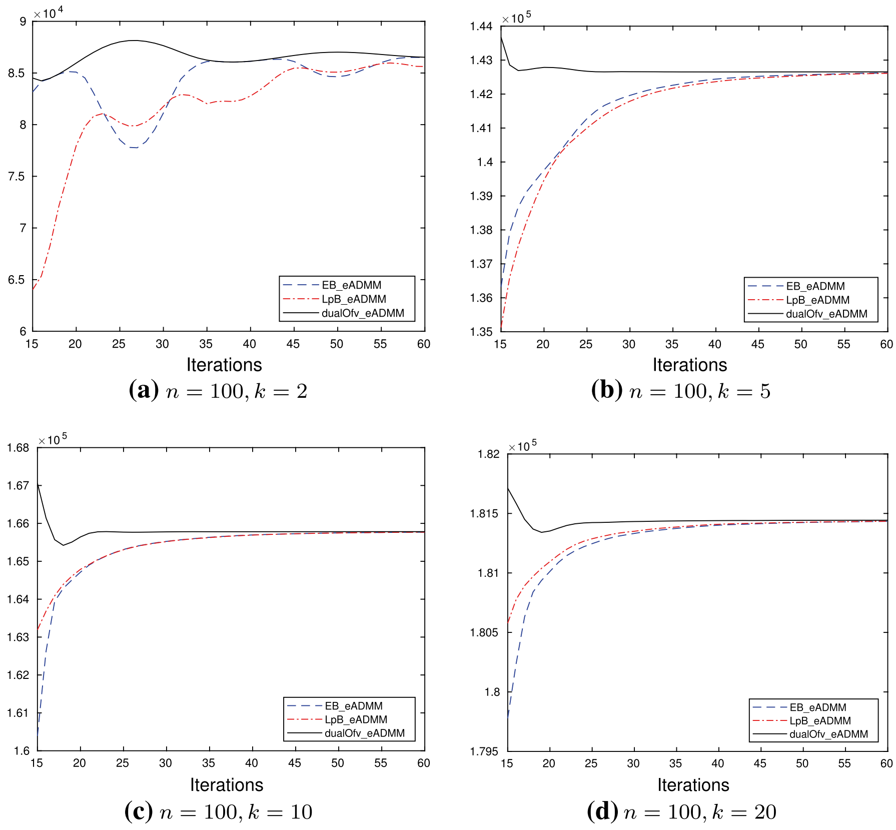


Fig. 1 Lower bounds obtained with post processing

k, the rigorous error bound method gives tighter bounds (Fig. 1b), but as k increases, the linear programming bound method dominates (Fig. 1c, d).

Remark 3 We choose Algorithm 2 in all following experiments as post-processing for k-equipartition problems because this method is more stable for varying k. For GPKC we use Algorithm 3 for the post-processing since we have no information on the eigenvalue of an optimal solution.

6.3 Results for *k*-equipartition

6.3.1 Comparison of the Lower Bounds using SDP, DNN and Transitivity Constraints

In this section we want to highlight the improvement of the bounds obtained from the relaxations introduced in Sect. 2. Note that the timings for computing these bounds are discussed later in Sect. 6.3.2.

In practice, adding all the transitivity constraints is computationally too expensive, we run a DNN-based loop instead. The idea is as follows.

1. Solve the DNN (3) to obtain the solution X^{DNN}.
2. Add m_{met} transitivity constraints that are most violated by X^{DNN} to the relaxation.
3. Solve the resulting relaxation and repeat adding newly violated constraints until the maximum number of iterations is reached or no more violated constraints are found.

Tables 1, 2 and 3 compare the lower bounds obtained from the relaxations for the k-equipartition problem. The improvements are calculated as $(d_{DNN} - d_{SDP})/d_{SDP}$

Table 1 k-equipartitioning lower bounds on rand80

n	k	lb_{SDP}	lb_{DNN}	Imp%	$lb_{DNN+MET}$	Imp%
100	2	86,605.32	86,616.54	0.01	87,900.20	1.50
	4	129,938.45	132,491.14	1.96	132,777.00	2.18
	5	138,603.13	142,647.17	2.92	142,736.00	2.98
	10	155,933.11	165,781.01	6.32	165,782.00	6.32
	20	164,598.46	181,444.03	10.23	–	–
	25	166,331.65	185,340.17	11.43	–	–
200	2	362,347.86	362,366.87	0.01	365,512.00	0.87
	4	543,624.41	550,178.14	1.21	550,616.00	1.29
	5	579,880.79	590,123.76	1.77	590,281.00	1.79
	10	652,392.81	677,024.59	3.78	677,048.00	3.78
	20	688,652.15	730,635.43	6.10	730,637.00	6.10
	40	706,783.98	766,592.49	8.46	–	–
⋮	⋮	⋮	⋮	⋮	⋮	⋮
900	2	7,667,181.82	7,667,181.82	0.00	7,685,601.68	0.24
	5	12,277,076.08	12,326,530.46	0.40	12,331,888.86	0.45
	10	13,813,797.53	13,957,505.42	1.04	–	–
	20	14,582,084.03	14,854,713.50	1.87	–	–
	30	14,838,190.47	15,197,038.17	2.42	–	–
	50	15,043,065.60	15,518,811.79	3.16	–	–
	100	15,196,404.75	15,821,635.34	4.11	–	–
	300	15,299,130.76	16,068,306.79	5.03	–	–
1,000	2	9,512,467.50	9,520,746.69	0.09	9,534,980.44	0.24
	5	15,233,252.24	15,287,792.55	0.36	15,295,612.25	0.41
	10	17,139,974.48	17,302,696.39	0.95	–	–
	20	18,093,221.41	18,404,248.61	1.72	–	–
	40	18,569,845.83	19,056,214.07	2.62	–	–
	50	18,665,166.60	19,211,689.51	2.93	–	–
	100	18,855,894.81	19,578,435.97	3.83	–	–
	200	18,951,231.87	19,800,076.60	4.48	–	–

Table 2 k-equipartitioning lower bounds on rand50

n	k	lb_{SDP}	lb_{DNN}	Imp%	$lb_{DNN+MET}$	Imp%
100	2	47, 928.97	47, 928.23	0.00	49, 300.26	2.86
	4	71, 902.51	74, 349.78	3.40	74, 714.81	3.91
	5	76, 697.24	80, 404.43	4.83	80, 566.86	5.05
	10	86, 286.59	94, 925.78	10.01	94, 931.82	10.02
	20	91, 081.36	105, 704.48	16.06	105, 705.40	16.06
	25	105, 704.48	108, 816.55	2.94	108, 817.97	2.95
200	2	209, 582.35	209, 592.40	0.00	212, 563.49	1.42
	4	314, 422.58	320, 218.87	1.84	320, 740.01	2.01
	5	335, 391.78	344, 366.46	2.68	344, 545.59	2.73
	10	377, 332.46	398, 653.05	5.65	398, 686.12	5.66
	20	398, 306.21	434, 293.16	9.03	–	–
	40	408, 794.19	462, 004.09	13.02	–	–
⋮	⋮	⋮	⋮	⋮	⋮	⋮
900	2	4, 641, 598.87	4, 644, 349.58	0.06	4, 657, 296.91	0.34
	5	7, 431, 923.29	7, 481, 538.62	0.67	7, 484, 865.68	0.71
	10	8, 362, 037.01	8, 499, 452.66	1.64	–	–
	20	8, 827, 102.41	9, 078, 072.19	2.84	–	–
	30	8, 982, 122.95	9, 308, 248.74	3.63	–	–
	50	9, 106, 135.45	9, 533, 366.09	4.69	–	–
	100	9, 199, 147.87	9, 776, 472.27	6.28	–	–
	300	9, 261, 139.26	10, 009, 909.74	8.09	–	–
1, 000	2	5, 760, 088.00	5, 760, 088.00	0.00	5, 778, 737.72	0.32
	5	9, 224, 420.72	9, 278, 976.84	0.59	9, 283, 474.07	0.64
	10	10, 378, 975.72	10, 534, 126.08	1.49	–	–
	20	10, 956, 261.72	11, 242, 920.84	2.62	–	–
	40	11, 244, 891.69	11, 684, 089.99	3.91	–	–
	50	11, 302, 617.95	11, 794, 006.69	4.35	–	–
	100	11, 418, 071.85	12, 084, 361.06	5.84	–	–
	200	11, 475, 814.32	12, 289, 919.20	7.09	–	–

and $(d_{DNN+MET} - d_{SDP})/d_{SDP}$, respectively; a '–' indicates that no transitivity constraints violated by the SDP solution of problem (3) have been found. In [21] it has been observed that the violation of the transitivity constraints is small and the nonnegativity constraints $X \geq 0$ are more important than $X \in$ MET when the partition number k increases. In our experiments we also observe that the improvement due to the nonnegativity constraints gets even better as k increases.

Table 3 *k*-equipartitioning lower bounds on rand20

n	k	lb_{SDP}	lb_{DNN}	Imp%	$lb_{DNN+MET}$	Imp%
100	2	14, 747.59	14, 747.59	0.00	15, 762.93	6.88
	4	22, 127.04	23, 466.71	6.05	23, 980.80	8.38
	5	23, 602.89	25, 695.45	8.87	26, 002.67	10.17
	10	26, 554.54	31, 618.12	19.07	31, 684.77	19.32
	20	28, 030.30	36, 793.39	31.26	36, 803.50	31.30
	25	28, 325.43	38, 432.17	35.68	38, 436.72	35.70
200	2	70, 569.62	70, 569.62	0.00	72, 897.33	3.30
	4	105, 873.90	109, 373.17	3.31	110, 128.59	4.02
	5	112, 935.88	118, 429.95	4.86	118, 775.68	5.17
	10	127, 058.52	140, 350.63	10.46	140, 393.14	10.49
	20	134, 120.25	156, 812.40	16.92	156, 815.89	16.92
	40	137, 652.36	170, 920.84	24.17	170, 921.06	24.17
⋮	⋮	⋮	⋮	⋮	⋮	⋮
900	2	1, 715, 687.43	1, 715, 987.61	0.02	1, 726, 605.37	0.64
	5	2, 747, 123.67	2, 782, 017.73	1.27	2, 783, 340.64	1.32
	10	3, 090, 931.90	3, 184, 081.52	3.01	–	–
	20	3, 262, 832.49	3, 430, 495.43	5.14	–	–
	30	3, 320, 134.28	3, 535, 568.53	6.49	–	–
	50	3, 365, 975.14	3, 644, 361.83	8.27	–	–
	100	3, 400, 355.41	3, 767, 595.30	10.80	–	–
	300	3, 423, 251.06	3, 945, 849.47	15.27	–	–
1, 000	2	2, 140, 975.92	2, 141, 688.49	0.03	2, 152, 422.29	0.53
	5	3, 427, 722.39	3, 465, 886.87	1.11	3, 467, 615.21	1.16
	10	3, 856, 783.95	3, 960, 838.71	2.70	–	–
	20	4, 071, 315.84	4, 260, 408.03	4.64	–	–
	40	4, 178, 581.29	4, 463, 023.58	6.81	–	–
	50	4, 200, 033.96	4, 516, 439.34	7.53	–	–
	100	4, 242, 940.30	4, 661, 565.65	9.87	–	–
	200	4, 264, 393.83	4, 801, 394.97	12.59	–	–

6.3.2 Comparisons between extended ADMM and Interior Point Methods (IPMs) on *k*-equipartition

In this section we want to demonstrate the advantage of our extended ADMM over interior point methods. For our comparisons we use Mosek [1], one of the currently best performing interior point solvers.

Note that computing an equipartition for these graphs using commercial solvers is out of reach. For instance, Gurobi obtains for a graph with $n = 100$ vertices and $k \in \{2, 4, 5, 10, 20, 25\}$ after 120 s a gap of at least 80 %, whereas we obtain gap up to at most 7 %.

We list the results for solving the SDP (2) using an ADMM, and the results when solving the DNN relaxation (3) by our extended ADMM and by Mosek. We run the experiments on randomly generated graphs with 80% density, the results are given in Table 4.

Table 4 shows that the convergence behavior of the extended ADMM is not worse than the 2-block ADMM for SDP problems, and we can get a tighter lower bound by forcing nonnegativity constraints in the model without higher computational expense.

The results for Mosek solving problem (3) clearly show that these problems are out of reach for interior point solvers. A "−" indicates that Mosek failed to solve this instance due to memory requirements.

6.3.3 Heuristics on *k*-equipartition Problems

We now compare the heuristics introduced in Sect. 5 to get upper bounds for the graph partition problems.

We use the solutions obtained from the DNN relaxation to build upper bounds for the k-equipartition problem since the experimental results in Sect. 6.4.1 showed that the DNN relaxation has a good tradeoff between quality of the bound and solution time.

We compare to the best know primal solution given in [2]; we set the time limit for our heuristics to 5 s. The gaps between the upper bounds by Vc+2opt (resp. Hyp+2opt) and the best know solution are shown in Table 5. The primal bounds that are proved to be optimal are marked with "*".

Table 5 shows that on small instances our heuristics can find upper bounds not worse than the best known upper bounds. For large instances, Vc+2opt performs better than Hyp+2opt. The corresponding upper bounds are less than 10 % away from the best known upper bounds, some of them computed using 5 h.

We next compare the upper bounds for the instances rand80, rand50, and rand20. Figure 2 shows that, for small instances (i.e., $n = 100$), our hybrid methods (eg. Vc+2opt and Hyp+2opt) can find tight upper bounds quickly while simulated annealing (SA) needs a longer burning down time to achieve an upper bound of good quality.

Figure 3 shows how the heuristics behave for large-scale instances (i.e, $n = 1000$). The time limit is set to 5 s. Compared to Fig. 2, Vc+2opt and Hyp+2opt take more time to generate the first upper bounds but these upper bounds are much tighter than the one found by SA. Also, when the time limit is reached, the upper bounds found by Vc+2-opt and Hyp+2opt are much tighter than those from SA.

Tables 6, 7 and 8 give a detailed comparison of the upper bounds for the instances rand80, rand50, and rand20, respectively. We display the gap between the lower bounds obtained from the DNN relaxation (3) and the upper bounds built by varied heuristics. The time limit for the heuristics is set to 1 s for $n \in \{100, 200\}$ and 3 s for $n \in \{900, 1000\}$ for rand80. For rand50 and rand20 we set the limit to 5 s. The best upper bounds are typeset in bold.

Table 4 Computation times for k-equipartitioning problems

		ADMM				Mosek
		SDP (2)		DNN (3)		DNN (3)
n	k	Iter	CPU time(s)	Iter	CPU time(s)	CPU time (s)
100	2	220	1.59	152	1.03	40.12
	4	302	2.16	79	0.48	41.13
	5	309	2.07	76	0.48	42.22
	10	291	2.17	88	0.81	37.97
	20	285	2.09	231	1.56	39.50
	25	286	1.81	209	1.42	35.31
200	2	223	3.85	162	3.71	2, 414.38
	4	339	5.92	81	1.43	2, 279.18
	5	361	5.63	75	1.50	2, 266.13
	10	382	6.05	69	1.24	2, 028.00
	20	369	5.75	104	2.05	1, 816.87
	40	379	6.01	324	7.05	2, 187.09
300	2	245	8.15	173	7.82	–
	3	307	10.65	114	4.06	–
	5	339	11.25	79	3.22	–
	6	346	12.53	72	2.73	–
	10	365	12.68	65	2.62	–
	30	413	14.72	130	5.82	–
	50	466	17.07	286	12.13	–
	100	580	21.99	1, 158	50.96	–
400	2	267	15.45	183	10.42	–
	4	345	20.87	90	6.62	–
	5	360	21.55	80	5.19	–
	8	382	25.72	64	4.57	–
	10	392	23.83	60	4.31	–
	20	440	28.03	68	6.24	–
	40	471	32.72	154	11.36	–
	100	616	43.61	734	55.43	–
500	2	256	20.87	187	16.14	–
	5	430	34.47	80	7.39	–
	10	485	39.49	59	5.06	–
	20	535	44.57	62	6.28	–
	25	546	47.44	71	6.58	–
	50	565	46.12	183	16.99	–
	100	683	54.59	559	51.17	–

Table 4 (continued)

		ADMM				Mosek
		SDP (2)		DNN (3)		DNN (3)
n	k	Iter	CPU time(s)	Iter	CPU time(s)	CPU time (s)
600	2	274	26.85	179	18.68	–
	3	348	38.90	112	12.48	–
	5	422	46.77	81	8.91	–
	6	437	48.41	73	8.11	–
	10	482	53.68	58	6.69	–
	20	536	59.20	59	6.96	–
	30	552	61.78	74	8.75	–
	50	565	64.16	155	18.91	–
	100	714	82.19	442	56.02	–
	200	923	106.55	890	111.57	–
700	2	266	39.92	209	35.69	–
	5	452	70.57	80	12.74	–
	10	535	85.25	57	9.14	–
	20	590	103.52	57	9.53	–
	35	612	97.37	75	13.13	–
	50	620	99.19	132	21.64	–
	70	658	104.88	222	37.70	–
	100	776	124.72	380	64.98	–
800	2	253	51.79	211	48.68	–
	5	451	96.57	79	17.60	–
	10	537	116.06	57	12.71	–
	20	595	128.97	55	12.55	–
	40	620	134.88	75	17.59	–
	50	625	136.16	118	28.53	–
	100	759	166.67	333	78.13	–
	200	1, 087	240.91	776	186.33	–
900	2	246	67.94	218	65.91	–
	5	434	124.60	80	23.52	–
	10	523	151.21	57	16.92	–
	20	583	169.88	54	16.58	–
	30	601	174.28	63	19.73	–
	50	613	178.47	104	35.40	–
	100	711	207.95	305	95.00	–
	300	1, 644	489.24	515	162.40	–

Table 4 (continued)

		ADMM				Mosek
		SDP (2)		DNN (3)		DNN (3)
n	k	Iter	CPU time(s)	Iter	CPU time(s)	CPU time (s)
1, 000	2	246	88.62	200	77.91	–
	5	449	166.91	80	30.85	–
	10	549	206.41	57	21.88	–
	20	612	232.10	52	20.45	–
	40	637	249.78	72	30.02	–
	50	642	241.78	91	38.88	–
	100	705	265.93	278	114.78	–
	200	1, 306	496.70	682	277.84	–

The numbers confirm that Vc+2opt and Hyp+2opt can build tighter upper bounds than SA, in particular for the dense graphs rand80. Overall, Vc+2opt has the best performance.

Comparing lower and upper bounds, the numerical results show that our methods perform very well on dense graphs; for rand80 the largest gap is less than 4%, for rand50 the largest gap is less than 6%. As the randomly generated graph gets sparser, the gap between lower bounds and upper bounds increases, for rand20 the gap is bounded by 12%.

Comparing Tables 6, 7 and 8 , it can be observed that the gaps get larger as the graph gets sparser. We conjecture that this is due to less tightness of the lower bound which is supported by the following experiment.

We regard the best upper bounds obtained from all three heuristics within an increased time limit of 10 s. In this way, we should have an upper bound that approximates the optimal solution well enough for all densities. As an example, in Table 9 we report for a graph on 100 vertices and three different densities the lower and these upper bounds. We can clearly see that when the graph gets sparser, adding the nonnegativity constraints to the SDP relaxation (2) gains more improvement. However, the gap between lower and upper bound gets worse as the graph gets sparser.

6.4 Results for GPKC

6.4.1 Comparison of the Lower Bounds using SDP, DNN and Transitivity Constraints

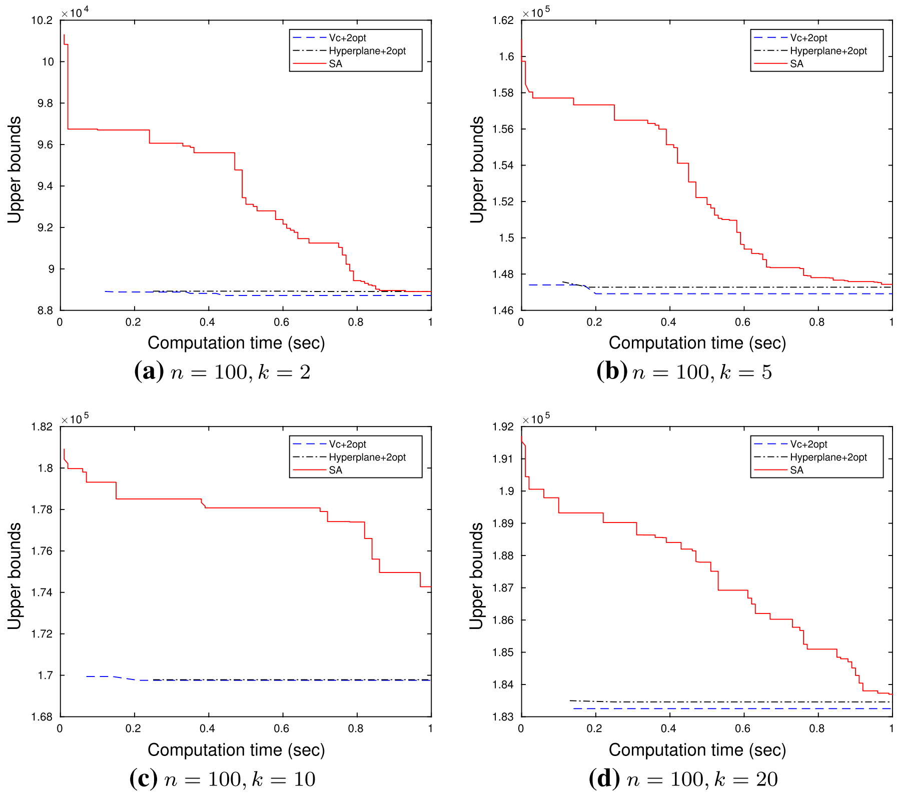
We now turn our attention to the GPKC problem. We run experiments similar to those presented in Sect. 6.3.1, i.e., we solve DNN (7) to obtain the solution X^{DNN}. Table 10 shows the lower bounds for GPKC problems on the randomly generated graphs rand80. The improvements are calculated in the same

Table 5 Feasible solutions for the graphs from [2] (the time limit is 5 s, optimal solutions are indicated by a “*”)

Graph	n	k	Vc+2opt	Gap%	Hyp+2opt	Gap%	Best Bound
$U_{500,5}$	500	2	2	0.00	4	100.00	2*
$U_{500,10}$	500	2	26	0.00	30	15.38	26*
$U_{500,20}$	500	2	178	0.00	179	0.56	178
$U_{500,40}$	500	2	412	0.00	412	0.00	412
$U_{1000,5}$	1000	2	1	0.00	3	200.00	1*
$U_{1000,10}$	1000	2	39	0.00	65	66.67	39*
$U_{1000,20}$	1000	2	242	9.01	303	36.49	222
$U_{1000,40}$	1000	2	737	0.00	921	24.97	737
$G_{124,2.5}$	124	2	13	0.00	13	0.00	13*
$G_{124,5}$	124	2	63	0.00	64	1.59	63*
$G_{124,10}$	124	2	178	0.00	179	0.56	178*
$G_{124,20}$	124	2	449	0.00	449	0.00	449*
$G_{250,2.5}$	250	2	30	3.45	30	3.45	29*
$G_{250,5}$	250	2	116	1.75	117	2.63	114
$G_{250,10}$	250	2	362	1.40	362	1.40	357
$G_{250,25}$	250	2	831	0.36	832	0.48	828
$G_{500,2.5}$	500	2	52	6.12	52	6.12	49
$G_{500,5}$	500	2	226	3.67	237	8.72	218
$G_{500,10}$	500	2	638	1.92	654	4.47	626
$G_{500,20}$	500	2	1, 765	1.20	1, 779	2.01	1, 744
$G_{1000,2.5}$	1000	2	106	3.92	109	6.86	102
$G_{1000,5}$	1000	2	479	6.21	492	9.09	451
$G_{1000,10}$	1000	2	1, 404	2.71	1, 431	4.68	1, 367
$G_{1000,20}$	1000	2	3, 437	1.42	3, 450	1.80	3, 389
mesh.138.232	138	2	8	0.00	8	0.00	8*
mesh.148.265	148	2	7	0.00	7	0.00	7*
mesh.274.469	274	2	7	0.00	7	0.00	7*
mesh.70.120	70	2	7	0.00	7	0.00	7*
mesh.74.129	74	2	8	0.00	8	0.00	8*

way as in the previous section. The experimental results on GPKCrand50 and GPKCrand20 are omitted since they have a similar behavior.

The lower bounds obtained from different SDP relaxations show that when the capacity bound W decreases (and thus the number of groups increases), the improvement of the nonnegativity constraints gets more significant. This is in line with the results for k-equipartition. And, also similar to k-equipartition, for GPKC the improvement due to the transitivity constraints is only minor.



(a) $n = 100, k = 2$ **(b)** $n = 100, k = 5$ **(c)** $n = 100, k = 10$ **(d)** $n = 100, k = 20$

Fig. 2 Upper bounds for k-equipartition problems on rand80 with $n = 100$

6.4.2 Comparisons between extended ADMM and IPMs for GPKC

Table 11 compares the computation times when solving the DNN relaxations for the GPKC (7) by the extended ADMM and Mosek, respectively. A "–" indicates for extended ADMM that the maximum number of iterations is reached, and for Mosek that the instance could not be solved due to memory requirements. The results of the SDP relaxation (6) in Table 10 are computed using Mosek, hence we omit these timings in Table 11.

In the thesis [19], numerical results on the GPKC are presented using an LP relaxation. However, the method therein is capable of getting bounds either for very sparse graphs of density at most 6 % (up to 2000 vertices) or for graphs with up to 140 vertices and density of at most 50 %. We clearly outperform these results in terms of the density of the graphs that can be considered.

While for instances of size $n = 100$, the timings of the extended ADMM and Mosek are comparable, the picture rapidly changes as n increases. For $n \geq 300$, Mosek cannot solve any instance while the extended ADMM manages to obtain bounds for instances with $n = 500$ within 1 h.

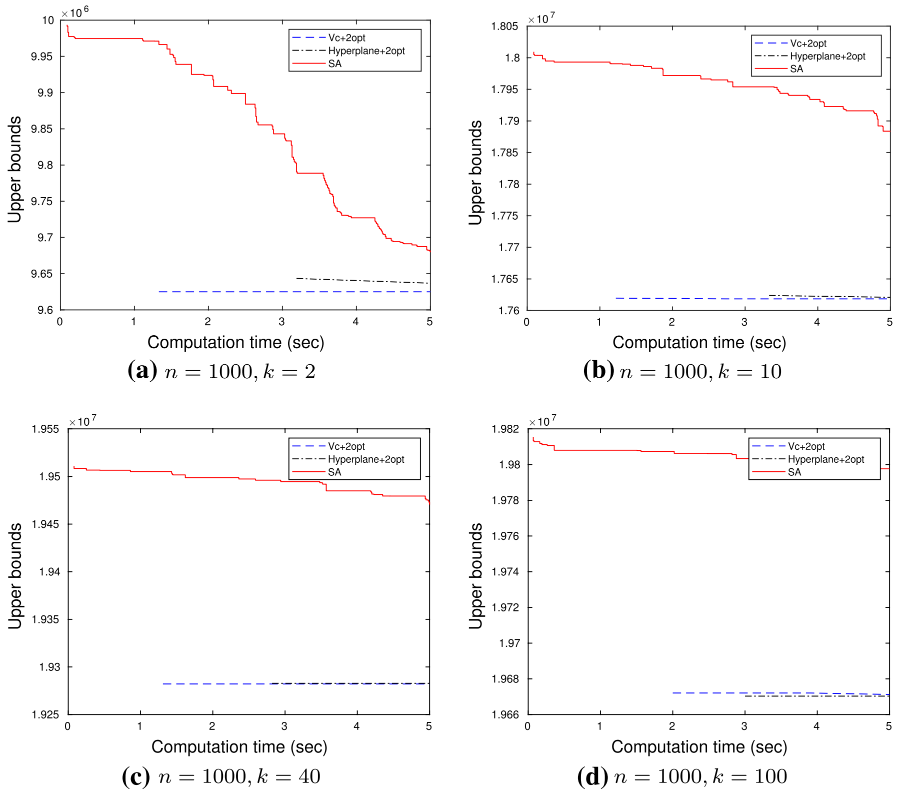


Fig. 3 Upper bounds for k-equipartition problems on rand80 with $n = 1000$

6.4.3 Heuristics on GPKC problems

As mentioned in Sect. 5, the simulated annealing heuristic for the QAP cannot be applied to the GPKC, because there is no equivalence between the GPKC and the QAP. Therefore, we compare the upper bounds for the GPKC from the heuristic introduced in Sect. 5.3 with the lower bounds given by the DNN relaxation (7). We set a time limit of 5 s.

Also, we set the maximum number of iterations to be 50,000 for the sparse graph GPKCrand20, while the maximum numbers of iterations for GPKCrand50 and GPKCrand80 are 20,000. In Tables 12, 13 and 14, a * indicates for the extended ADMM that the maximum number of iterations is reached.

Table 12 shows that the gaps between the lower and upper bounds are less than 3% for GPKCrand80, they are less than 7% for GPKCrand50, see Table 13, and for GPKCrand20, the gaps are less than 15%, see Table 14. Similar to the k-equipartition problem, we note that computing the lower bound on the sparse instances is harder. The maximum number of iterations is reached for rand20 much more often than for rand80 or rand50.

Table 6 Feasible solutions for randomly generated graphs rand80 (for instances with $n \in \{100, 200\}$, the time limit is 1 s; for instances with $n \in \{900, 1000\}$, the limit is 3 s)

n	k	lb_{DNN}	Vc+2opt	Gap%	Hyp+2opt	Gap%	SA	Gap%
100	2	86, 605.32	**88, 717**	2.44	88, 910	2.66	88, 958	2.72
	4	132, 491.14	136, 591	3.09	**136, 458**	2.99	137, 372	3.68
	5	142, 647.17	**146, 915**	2.99	147, 276	3.24	147, 265	3.24
	10	165, 781.01	**169, 756**	2.40	169, 786	2.42	177, 396	7.01
	20	181, 444.03	183, 251	1.00	**183, 460**	1.11	183, 676	1.23
	25	185, 340.17	**186, 715**	0.74	186, 682	0.72	189, 530	2.26
200	2	362, 366.87	**369, 966**	2.10	370, 311	2.19	371, 930	2.64
	4	550, 178.14	**564, 627**	2.63	564, 831	2.66	570, 125	3.63
	5	590, 123.76	605, 614	2.62	**605, 316**	2.57	622, 799	5.54
	10	677, 024.59	**692, 776**	2.33	692, 994	2.36	711, 926	5.16
	20	730, 635.43	**742, 377**	1.61	742, 386	1.61	758, 429	3.80
	40	766, 592.49	**771, 494**	0.64	771, 889	0.69	784, 773	2.37
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
900	2	7, 667, 181.82	**7, 765, 356**	1.28	7, 774, 223	1.40	8, 049, 595	4.99
	5	12, 326, 530.46	**12, 550, 665**	1.82	12, 551, 486	1.82	12, 909, 228	4.73
	10	13, 957, 505.42	**14, 221, 471**	1.89	14, 225, 896	1.92	14, 535, 127	4.14
	20	14, 854, 713.50	15, 103, 466	1.67	**15, 101, 121**	1.66	15, 354, 017	3.36
	30	15, 197, 038.17	15, 411, 096	1.41	**15, 409, 817**	1.40	15, 622, 960	2.80
	50	15, 518, 811.79	15, 674, 587	1.00	**15, 673, 142**	0.99	15, 841, 316	2.08
	100	15, 821, 635.34	15, 892, 607	0.45	**15, 891, 366**	0.44	16, 007, 565	1.18
	300	16, 068, 306.79	**16, 073, 556**	0.03	16, 073, 569	0.03	16, 118, 899	0.31
1000	2	9, 520, 746.69	**9, 624, 982**	1.09	9, 643, 410	1.29	9, 966, 328	4.68
	5	15, 287, 792.55	**15, 556, 241**	1.76	15, 567, 063	1.83	15, 976, 113	4.50
	10	17, 302, 696.39	**17, 619, 665**	1.83	17, 623, 783	1.86	17, 993, 155	3.99
	20	18, 404, 248.61	**18, 699, 789**	1.61	18, 703, 350	1.63	19, 000, 174	3.24
	40	19, 056, 214.07	**19, 282, 129**	1.19	19, 282, 815	1.19	19, 506, 580	2.36
	50	19, 211, 689.51	**19, 402, 983**	1.00	19, 405, 492	1.01	19, 606, 983	2.06
	100	19, 578, 435.97	19, 672, 071	0.48	**19, 670, 328**	0.47	19, 808, 050	1.17
	200	19, 800, 076.60	19, 826, 975	0.14	**19, 826, 442**	0.13	19, 911, 225	0.56

7 Conclusions

In this paper we first introduce different SDP relaxations for k-equipartition problems and GPKC problems. Our tightest SDP relaxations, problems (4) and (8), contain all nonnegativity constraints and transitivity constraints, which bring $O(n^3)$ constraints in total. Another kind of tight SDP relaxation, (3) and (7), has only nonnegativity constraints. While it is straight forward to consider the constraint $X \geq 0$ in a 3-block ADMM, including all the transitivity constraints is impractical. Therefore, our strategy is to solve (3) and (7) and then adding violated transitivity constraints in loops to tighten both SDP relaxations.

Table 7 Feasible solutions for randomly generated graphs rand50 (time limit 5 s)

n	k	lb_{DNN}	Vc+2opt	Gap%	Hyp+2opt	Gap%	SA	Gap%
100	2	47, 928.23	**50, 108**	4.55	**50, 108**	4.55	50, 256	4.86
	4	74, 349.78	**77, 861**	4.72	77, 904	4.78	77, 939	4.83
	5	80, 404.43	**84, 224**	4.75	84, 334	4.89	84, 391	4.96
	10	94, 925.78	98, 649	3.92	**98, 344**	3.60	98, 546	3.81
	20	105, 704.48	108, 081	2.25	**108, 016**	2.19	108, 028	2.20
	25	108, 816.55	110, 550	1.59	110, 443	1.49	**110, 397**	1.45
200	2	209, 592.40	**216, 781**	3.43	216, 884	3.48	217, 758	3.90
	4	320, 218.87	**333, 509**	4.15	333, 828	4.25	334, 519	4.47
	5	344, 366.46	**358, 788**	4.19	359, 029	4.26	359, 623	4.43
	10	398, 653.05	414, 227	3.91	414, 248	3.91	**412, 837**	3.56
	20	434, 293.38	447, 107	2.95	447, 410	3.02	**446, 754**	2.87
	40	462, 004.09	468, 072	1.31	468, 126	1.33	**467, 900**	1.28
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
900	2	4, 644, 349.58	**4, 729, 597**	1.84	4, 739, 736	2.05	4, 751, 219	2.30
	5	7, 481, 538.62	**7, 696, 505**	2.87	7, 700, 566	2.93	7, 755, 369	3.66
	10	8, 499, 452.66	**8, 754, 562**	3.00	8, 755, 697	3.01	8, 862, 487	4.27
	20	9, 078, 072.19	**9, 325, 687**	2.73	9, 326, 356	2.73	9, 469, 542	4.31
	30	9, 308, 248.74	**9, 535, 446**	2.44	9, 536, 350	2.45	9, 672, 001	3.91
	50	9, 533, 366.09	**9, 716, 248**	1.92	9, 716, 489	1.92	9, 848, 828	3.31
1000	2	5, 760, 088.00	**5, 870, 879**	1.92	5, 871, 853	1.94	5, 899, 773	2.43
	5	9, 278, 976.84	**9, 540, 363**	2.82	9, 540, 969	2.82	9, 658, 512	4.09
	10	10, 534, 126.08	**10, 840, 528**	2.91	10, 845, 227	2.95	11, 041, 031	4.81
	20	11, 242, 920.84	11, 548, 172	2.72	**11, 545, 943**	2.70	11, 764, 487	4.64
	40	11, 684, 089.99	**11, 934, 463**	2.14	11, 934, 795	2.15	12, 122, 767	3.75
	50	11, 794, 006.69	**12, 020, 096**	1.92	12, 022, 058	1.93	12, 200, 503	3.45

In order to deal with the SDP problems with inequality and bound constraints, we extend the classical 2-block ADMM, which only deals with equations, to the extended ADMM for general SDP problems. This algorithm is designed to solve large instances that interior point methods fail to solve. We also introduce heuristics that build upper bounds from the solutions of the SDP relaxations. The heuristics include two parts, first we round the SDP solutions to get a feasible solution for the original graph partition problem, then we apply 2-opt methods to locally improve this feasible solution. In the procedure of rounding SDP solutions, we introduce two algorithms, the vector clustering method and the generalized hyperplane rounding method. Both methods perform well with the 2-opt method.

The extended ADMM can solve general SDP problems efficiently. For SDP problems with bound constraints, the extended ADMM deals with them separately from inequalities and equations, thereby solving the problems more efficiently. Mosek fails to solve the DNN relaxations of problems with $n \geq 300$ due to memory requirements while the extended ADMM can solve the DNN relaxations for k-equipartition

Table 8 Feasible solutions for randomly generated graphs rand20 (time limit 5 s)

n	k	lb_{DNN}	Vc+2opt	Gap%	Hyp+2opt	Gap%	SA	Gap%
100	2	14, 747.21	**16, 152**	9.53	**16, 152**	9.53	16, 182	9.73
	4	23, 468.28	**26, 029**	10.91	26, 088	11.16	26, 046	10.98
	5	25, 696.90	28, 443	10.69	28, 465	10.77	**28, 320**	10.21
	10	31, 618.43	**34, 180**	8.10	34, 196	8.15	34, 263	8.36
	20	36, 793.39	**38, 709**	5.21	38, 712	5.21	38, 832	5.54
	25	38, 432.16	39, 836	3.65	40, 009	4.10	**39, 671**	3.22
200	2	70, 566.72	75, 667	7.23	**75, 490**	6.98	76, 591	8.54
	4	109, 373.17	118, 849	8.66	**118, 546**	8.39	119, 244	9.02
	5	118, 429.95	**128, 529**	8.53	128, 778	8.74	129, 850	9.64
	10	140, 350.63	**151, 485**	7.93	152, 110	8.38	152, 101	8.37
	20	156, 812.40	166, 711	6.31	167, 022	6.51	**166, 371**	6.10
	40	170, 920.84	**177, 531**	3.87	177, 628	3.92	177, 696	3.96
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
900	2	1, 715, 987.61	**1, 782, 307**	3.86	1, 786, 114	4.09	1, 796, 268	4.68
	5	2, 782, 017.73	**2, 932, 461**	5.41	2, 938, 334	5.62	2, 969, 129	6.73
	10	3, 184, 081.52	**3, 366, 215**	5.72	3, 366, 617	5.73	3, 409, 458	7.08
	20	3, 430, 495.43	**3, 613, 787**	5.34	3, 615, 197	5.38	3, 674, 447	7.11
	30	3, 535, 568.53	**3, 707, 785**	4.87	3, 709, 718	4.93	3, 768, 366	6.58
	50	3, 644, 361.83	**3, 792, 021**	4.05	3, 794, 172	4.11	3, 862, 982	6.00
1000	2	2, 141, 688.49	**2, 219, 733**	3.64	2, 222, 627	3.78	2, 235, 931	4.40
	5	3, 465, 886.87	**3, 643, 532**	5.13	3, 653, 694	5.42	3, 692, 085	6.53
	10	3, 960, 838.71	4, 182, 733	5.60	**4, 179, 458**	5.52	4, 253, 642	7.39
	20	4, 260, 408.03	**4, 482, 570**	5.21	4, 483, 451	5.24	4, 593, 686	7.82
	40	4, 463, 023.58	**4, 659, 444**	4.40	4, 660, 450	4.42	4, 783, 289	7.18
	50	4, 516, 439.34	**4, 699, 391**	4.05	4, 699, 495	4.05	4, 825, 571	6.84

Table 9 Feasible solutions for the randomly generated graphs rand80, rand50, rand20 ($n = 100$ $k = 5$, with an increased time limit for heuristics of 10 s)

Density%	lb_{SDP}	lb_{DNN}	Improvement%	ub	Gap%
80	138, 603.13	142, 647.17	2.92	146, 565	2.75
50	76, 697.24	80, 404.43	4.83	84, 334	4.89
20	23, 602.89	25, 696.90	8.87	28, 320	10.21

Table 10 GPKC lower bounds on GPKCrand80

n	W	lb_{SDP}	lb_{DNN}	Imp%	$lb_{DNN+MET}$	Imp%
100	26536	79, 801.90	79, 863.89	0.08	80, 641.38	1.05
	14023	117, 886.58	121, 989.72	3.48	122, 471.69	3.89
	11434	125, 769.08	132, 094.08	5.03	132, 441.03	5.30
	6321	141, 338.32	154, 560.71	9.36	154, 733.56	9.48
	3668	149, 417.80	169, 471.95	13.42	169, 667.97	13.55
	3043	151, 321.28	173, 607.35	14.73	173, 792.45	14.85
200	50084	334, 496.25	334, 909.68	0.12	336, 293.72	0.54
	25787	489, 900.72	511, 836.11	4.48	512, 694.32	4.65
	21741	515, 782.13	544, 827.75	5.63	545, 325.23	5.73
	11404	581, 909.21	640, 300.76	10.03	640, 580.64	10.08
	6686	612, 092.40	692, 857.83	13.19	693, 182.87	13.25
	3419	632, 993.42	737, 482.61	16.51	737, 654.49	16.53

problems on large instances up to $n = 1000$ within as few as 5 min and for GPKC problems up to $n = 500$ within as little as 1 h.

We run numerical tests on instances from the literature and on randomly generated graphs with different densities. The results show that SDP relaxations can produce tighter bounds for dense graphs than sparse graphs. In general, the results show that nonnegativity constraints give more improvement when k increases.

We compare our heuristics with a simulated annealing method in the generation of upper bounds for k-equipartition problems. Our heuristics obtain upper bounds displaying better quality within a short time limit, especially for large instances. Our methods show better performance on dense graphs, where the final gaps are less than 4% for graphs with 80% density, while the gaps between lower and upper bounds for sparse graphs with 20% density are bounded by 12%. This is mainly due to the tighter lower bounds for dense graphs.

Table 11 Computation times for GPKC problems

		extended ADMM		Mosek
		DNN (6)		DNN (6)
n	W	Iterations	CPU time (s)	CPU time (s)
100	26, 536	–	–	53.52
	14, 023	2167	36.59	51.07
	11, 434	2241	39.73	56.48
	6, 321	2369	42.02	47.00
	3, 668	2760	47.80	47.82
	3, 043	3198	55.30	45.55
200	50, 084	4918	315.40	3, 576.52
	25, 787	3257	207.93	3, 541.04
	21, 741	3431	217.94	3, 065.87
	11, 404	3760	241.98	2, 657.11
	6, 686	3564	225.52	2, 620.47
	3, 419	3538	220.82	2, 155.30
300	73, 485	–	–	–
	49, 920	4837	557.46	–
	31, 134	5798	678.43	–
	25, 752	5820	676.88	–
	16, 767	5191	602.28	–
	6, 647	4010	459.82	–
	4, 210	4728	541.19	–
	2, 454	5355	606.81	–
400	73, 485	–	–	–
	49, 920	4837	557.46	–
	31, 134	5798	678.43	–
	25, 752	5820	676.88	–
	16, 767	5191	602.28	–
	6, 647	4010	459.82	–
	4, 210	4728	541.19	–
	2, 454	5355	606.81	–
500	132, 135	–	–	–
	53, 186	10061	3, 492.13	–
	26, 965	8710	3, 237.27	–
	15, 049	8982	3, 146.47	–
	12, 806	9834	4, 426.28	–
	7, 312	7054	3, 101.58	–
	4, 071	6858	2, 442.62	–

Table 12 Feasible solutions for randomly generated graphs on GPKC problems GPKCrand80 (the maximum number of iterations for eADMM is 20 000, a * indicates that the maximum number of iterations is reached)

n	W	lb_{DNN}	VC+2opt	Gap%
100	26, 536	79, 861.23*	81, 191	1.67
	14, 023	121, 989.22	124, 781	2.29
	11, 434	132, 093.71	135, 681	2.72
	6, 321	154, 560.56	157, 773	2.08
	3, 668	169, 471.89	172, 343	1.69
	3, 043	173, 607.32	176, 385	1.60
200	50, 084	334, 909.34	339, 970	1.51
	25, 787	511, 835.53	517, 093	1.03
	21, 741	544, 827.45	553, 988	1.68
	11, 404	640, 300.82	651, 955	1.82
	6, 686	692, 857.79	704, 127	1.63
	3, 419	737, 482.45	745, 057	1.03
300	73, 485	723, 712.97*	729, 625	0.82
	49, 920	992, 357.12	1, 004, 965	1.27
	31, 134	1, 229, 623.39	1, 249, 143	1.59
	25, 752	1, 304, 759.48	1, 325, 865	1.62
	16, 767	1, 440, 209.56	1, 462, 605	1.56
	6, 647	1, 618, 112.76	1, 638, 290	1.25
	4, 210	1, 672, 195.76	1, 687, 602	0.92
	2, 454	1, 719, 157.25	1, 729, 916	0.63
400	99, 348	1, 338, 105.41	1, 351, 240	0.98
	51, 033	2, 071, 963.02	2, 093, 269	1.03
	41, 130	2, 237, 953.68	2, 264, 663	1.19
	28, 718	2, 462, 598.44	2, 494, 382	1.29
	22, 740	2, 578, 443.43	2, 614, 666	1.40
	11, 772	2, 812, 875.09	2, 848, 815	1.28
	6, 132	2, 961, 922.93	2, 991, 521	1.00
500	132, 135	1, 936, 655.66*	1, 986, 415	2.57
	53, 186	3, 467, 473.90	3, 521, 678	1.56
	26, 965	4, 073, 265.28	4, 130, 236	1.40
	15, 049	4, 396, 228.10	4, 452, 115	1.27
	12, 806	4, 464, 071.03	4, 523, 570	1.33
	7, 312	4, 651, 268.18	4, 697, 755	1.00
	4, 071	4, 783, 543.51	4, 821, 216	0.79

Table 13 Feasible solutions for randomly generated graphs on GPKC problems GPKCrand50 (the maximum number of iterations for eADMM is 20 000, a * indicates that the maximum number of iterations is reached)

n	W	lb_{DNN}	VC+2opt	Gap%
100	27, 572	43, 862.45*	46, 578	6.19
	14, 639	69, 143.36*	73, 148	5.79
	11, 686	75, 761.37	80, 014	5.61
	6, 526	89, 183.50	93, 252	4.56
	4, 040	97, 547.91	101, 998	4.56
	3, 179	101, 184.12	105, 129	3.90
200	48, 935	193, 293.88*	196, 369	1.59
	25, 685	300, 462.90*	310, 368	3.30
	21, 504	322, 166.70*	333, 111	3.40
	10, 742	384, 833.05	397, 883	3.39
	6, 034	418, 763.84*	432, 026	3.17
	3, 940	438, 101.15*	449, 402	2.58
300	76, 829	433, 450.12*	449, 273	3.65
	49, 664	615, 471.29	626, 291	1.76
	32, 370	741, 306.33	763, 853	3.04
	27, 123	782, 740.17	804, 648	2.80
	16, 670	873, 377.63	898, 159	2.84
	6, 371	985, 628.12	1, 011, 695	2.64
	4, 428	1, 014, 899.09	1, 038, 083	2.28
	2, 649	1, 049, 198.04	1, 066, 520	1.65
400	97, 431	780, 703.56*	790, 271	1.23
	48, 514	1, 242, 772.57	1, 261, 840	1.53
	44, 456	1, 285, 533.56	1, 317, 925	2.52
	25, 558	1, 501, 960.49	1, 535, 376	2.22
	22, 264	1, 544, 147.97	1, 578, 729	2.24
	11, 477	1, 700, 761.30	1, 740, 466	2.33
	6, 800	1, 786, 701.78	1, 825, 146	2.15
500	131, 286	1, 289, 330.60*	1, 301, 301	0.93
	56, 101	2, 127, 940.95	2, 185, 271	2.69
	29, 244	2, 476, 801.01	2, 534, 746	2.34
	15, 747	2, 685, 115.43	2, 750, 968	2.45
	13, 176	2, 730, 501.04	2, 793, 996	2.33
	7, 920	2, 836, 940.86	2, 893, 907	2.01
	4, 249	2, 937, 039.59	2, 979, 181	1.43

Table 14 Feasible solutions for randomly generated graphs on GPKC problems GPKCrand20 (the maximum number of iterations for eADMM is 50 000, a * indicates that the maximum number of iterations is reached)

n	W	lb_{DNN}	VC+2opt	Gap%
100	26, 472	14, 332.54*	16, 215	13.13
	14, 310	22, 340.23*	25, 531	14.28
	11, 397	24, 666.72*	27, 764	12.56
	6, 349	29, 856.16*	33, 447	12.03
	3, 973	33, 419.32	36, 909	10.44
	3, 082	35, 224.71*	38, 693	9.85
200	49, 329	66, 505.12*	72, 779	9.43
	26, 138	104, 505.04*	114, 208	9.28
	21, 312	113, 363.64*	124, 094	9.47
	11, 665	133, 889.10*	146, 638	9.52
	7, 007	147, 185.28	160, 241	8.87
	3, 747	160, 643.07*	171, 998	7.07
300	73, 762	163, 996.97*	172, 803	5.37
	49, 938	222, 399.96*	236, 916	6.53
	31, 051	272, 370.40*	293, 840	7.88
	25, 979	287, 194.07*	309, 615	7.81
	16, 089	319, 402.57*	344, 106	7.73
	6, 616	361, 919.99	386, 588	6.82
	4, 277	378, 355.72	400, 646	5.89
	2, 829	392, 682.34	411, 978	4.91
400	96, 066	295, 262.50*	306, 214	3.71
	49, 622	458, 045.11*	481, 621	5.15
	40, 512	493, 274.52*	520, 661	5.55
	26, 495	553, 117.75*	583, 029	5.41
	22, 338	572, 857.55*	606, 241	5.83
	11, 274	635, 081.65*	672, 230	5.85
	7, 300	664, 905.86	702, 511	5.66
500	133, 561	470, 275.89*	490, 915	4.39
	54, 084	801, 360.88	847, 836	5.80
	28, 866	925, 818.14	980, 463	5.90
	14, 840	1, 012, 359.76	1, 073, 619	6.05
	12, 923	1, 026, 775.88	1, 086, 881	5.85
	7, 468	1, 076, 226.50	1, 132, 106	5.19
	3, 975	1, 123, 555.87	1, 170, 459	4.17

Acknowledgements We thank Kim-Chuan Toh for bringing our attention to [13] and for providing an implementation of the method therein. We would like to thank the reviewers for their thoughtful comments and efforts towards improving this paper.

Funding Open access funding provided by University of Klagenfurt. This study was funded by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement MINOA No 764759.

Declarations

Conflicts of interest/Competing interests The authors have no conflicts of interest to declare that are relevant to the content of this article.

Availability of data and material All data can be downloaded from https://github.com/shudianzhao/ADMM-GP.

Code availability The codes can be downloaded from https://github.com/shudianzhao/ADMM-GP.

Ethics approval Not applicable

Consent to participate Not applicable

Consent for publication Not applicable

References

1. MOSEK ApS.: The MOSEK optimization toolbox for MATLAB manual. Version 9.1.13, 2020. http://docs.mosek.com/9.1.13/toolbox/index.html
2. Dipl.-Math Armbruster.: Branch-and-Cut for a Semidefinite relaxation of large-scale minimum bisection problems. (2007)
3. Chen, C., He, B., Ye, Y., Yuan, X.: The direct extension of ADMM for multi-block convex minimization problems is not necessarily convergent. Math. Programm. **155**(1–2), 57–79 (2016)
4. De Santis, M., Rendl, F., Wiegele, A.: Using a factored dual in augmented Lagrangian methods for semidefinite programming. Oper. Res. Lett. **46**(5), 523–528 (2018)
5. de Souza, C.C.: The graph equipartition problem: Optimal solutions, extensions and applications. PhD thesis, PhD-Thesis, Université Catholique de Louvain, Louvain-la-Neuve, Belgium (1993)
6. Fan, N., Pardalos, P.M.: Linear and quadratic programming approaches for the general graph partitioning problem. J. Glob. Optim. **48**(1), 57–71 (2010)
7. Frieze, A., Jerrum, M.: Improved approximation algorithms for MAX *k*-CUT and MAX bisection. In: International conference on integer programming and combinatorial optimization, pp. 1–13. Springer (1995)

8. Garey, M.R., Johnson, D.S., Stockmeyer, L.: Some simplified NP-complete problems. In: Proceedings of the sixth annual ACM symposium on theory of computing, pp. 47–63 (1974)
9. Ghaddar, B., Anjos, M.F., Liers, F.: A branch-and-cut algorithm based on semidefinite programming for the minimum *k*-partition problem. Ann. Oper. Res. **188**(1), 155–174 (2011)
10. Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J. ACM (JACM) **42**(6), 1115–1145 (1995)
11. Helmberg, C., Rendl, F., Weismantel, R.: Quadratic knapsack relaxations using cutting planes and semidefinite programming. In: International conference on integer programming and combinatorial optimization, pp. 175–189 (1996)
12. Hendrickson, B., Kolda, T.G.: Graph partitioning models for parallel computing. Parallel Comput. **26**(12), 1519–1534 (2000)
13. Jansson, C., Chaykin, D., Keil, C.: Rigorous error bounds for the optimal value in semidefinite programming. SIAM J. Numer. Anal. **46**(1), 180–200 (2007)
14. Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C.: Optimization by simulated annealing: an experimental evaluation; part i, graph partitioning. Oper. Res. **37**(6), 865–892 (1989)
15. Lin, S.: Computer solutions of the traveling salesman problem. Bell Syst. Tech. J. **44**(10), 2245–2269 (1965)
16. Lisser, A., Rendl, F.: Graph partitioning using linear and semidefinite programming. Math. Programm. **95**(1), 91–101 (2003)
17. Lorenz, D.A., Tran-Dinh, Q.: Non-stationary Douglas-Rachford and alternating direction method of multipliers: adaptive stepsizes and convergence. Comput. Optim. Appl. **74**(1), 67–92 (2019)
18. Malick, J., Povh, J., Rendl, F., Wiegele, A.: Regularization methods for semidefinite programming. SIAM J. Optim. **20**(1), 336–356 (2009)
19. Nguyen, D.P.: Contributions to graph partitioning problems under resource constraints. Doctoral dissertation, Université Pierre et Marie Curie-Paris VI (2016)
20. Nishihara, R., Lessard, L., Recht, B., Packard, A., Jordan, M.: A general analysis of the convergence of ADMM. In: International conference on machine learning, pp. 343–352 (2015)
21. Rendl, F.: Semidefinite programming and combinatorial optimization. Appl. Numer. Math. **29**(3), 255–281 (1999)
22. Sotirov, R.: SDP relaxations for some combinatorial optimization problems. In: Anjos, M.F. (ed.) Handbook on Semidefinite, Conic and Polynomial Optimization, pp. 795–819. Springer, Berlin (2012)
23. Sun, D., Toh, K.-C., Yuan, Y., Zhao, X.-Y.: SDPNAL+: a matlab software for semidefinite programming with bound constraints (version 1.0). Optimization Methods and Software, pp. 1–29 (2019)