SDP-based bounds for the Quadratic Cycle Cover Problem via
cutting plane augmented Lagrangian methods and
reinforcement learning

Frank de Meijer ∗ Renata Sotirov †

Abstract
We study the Quadratic Cycle Cover Problem (QCCP), which aims to find a node-disjoint cycle cover in a directed graph with minimum interaction cost between successive arcs. We derive several semidefinite programming (SDP) relaxations and use facial reduction to make these strictly feasible. We investigate a nontrivial relationship between the transformation matrix used in the reduction and the structure of the graph, which is exploited in an efficient algorithm that constructs this matrix for any instance of the problem. To solve our relaxations, we propose an algorithm that incorporates an augmented Lagrangian method into a cutting plane framework by utilizing Dykstra’s projection algorithm. Our algorithm is suitable for solving SDP relaxations with a large number of cutting planes. Computational results show that our SDP bounds and our efficient cutting plane algorithm outperform other QCCP bounding approaches from the literature. Finally, we provide several SDP-based upper bounding techniques, among which a sequential Q-learning method that exploits a solution of our SDP relaxation within a reinforcement learning environment.

Keywords quadratic cycle cover problem, semidefinite programming, facial reduction, cutting plane method, Dykstra’s projection algorithm, reinforcement learning.

1 Introduction
A disjoint cycle cover in a graph is a set of node-disjoint cycles such that every node is covered by exactly one cycle. The cycle cover problem (CCP) is the problem of finding a disjoint cycle cover such that the total arc weight is minimized. In this paper we focus on its quadratic version, which is known as the quadratic cycle cover problem (QCCP). The QCCP is the problem of finding a disjoint cycle cover in a graph such that the total sum of interaction costs between consecutive arcs is minimized. Although the problem can be defined for both directed and undirected graphs, we focus here on the asymmetric version which is defined on directed graphs.

The QCCP is introduced by Jäger and Molitor [42]. Fischer et al. [25] show that the problem is \( \mathcal{NP} \)-hard. This result is later on strengthened by De Meijer and Sotirov [43], who prove that the QCCP is strongly \( \mathcal{NP} \)-hard and not approximable within any constant factor.

There exist several special cases of the QCCP with respect to the objective function, for example, the angular metric cycle cover problem (ANGLE-CCP) [3] and the minimum reload cost cycle cover (MINRC3) problem [27]. In the ANGLE-CCP the quadratic costs represent the change of the direction induced by two consecutive arcs. The MINRC3 problem is the problem of finding a minimum disjoint cycle cover in an arc-colored graph under the reload cost model. These problems have applications in various fields, such as robotics [3], cargo and energy distribution networks [57]. We refer the interested reader to [43] for a more detailed overview of these variants of the QCCP and their applications. The QCCP may also be seen as a generalization of the minimum-turn cycle cover problem, which belongs to the class of covering-tour problems introduced in Arkin et al.
Covering tour problems play an important role in manufacturing, automatic inspection, spray painting operations, etc. For a detailed overview of cycle cover problems with turn costs and their applications, we refer the reader to [22].

The importance of the QCCP is also due to its close connection to the quadratic traveling salesman problem (QTSP). The goal of the QTSP is to find a Hamiltonian cycle in a graph minimizing the total quadratic costs between consecutive arcs. After removing the subtour elimination constraints, the QTSP boils down to the QCCP. Not surprisingly, the QTSP was introduced simultaneously with the QCCP in [12]. The QTSP is proven to be \( \mathcal{NP} \)-hard and not approximable within any constant factor [42] and is generally accepted to be one of the hardest combinatorial optimization problems nowadays. The QTSP has applications in robotics, bioinformatics and telecommunication, see e.g., Fischer et al. [24]. The QCCP plays an important role in obtaining strong lower and upper bounds for the QTSP [24, 42, 52].

Various papers study theoretical aspects as well as solution approaches for the QCCP and its variants. Fischer [23] studies the polyhedral structure of the quadratic cycle cover polytope and provides several inequalities that define the facets of this polytope. Bıyıkçöklak et al. [15] consider the MinRC3 on complete graphs with an equitable or nearly equitable 2-edge coloring and derive a polynomial time algorithm that constructs a monochromatic cycle cover. Jäger and Molitor [42] use approximated solutions of the QCCP as lower bounds in a branch-and-bound algorithm for the QTSP. Galbiati et al. [27] exploit a column generation approach to compute lower bounds for the MinRC3 problem. Stanek et al. [52] use the QCCP combined with a rounding procedure to construct heuristics for the QTSP. Several local search algorithms for the MinRC3 problem are given in [27]. Approximation algorithms for the QCCP and its variants are studied in [3, 4, 22].

The linearization problem of the QCCP is considered by De Meijer and Sotirov [43]. Several sufficient conditions for a QCCP instance to be linearizable are provided, which are used to construct strong linearization based bounds for any instance of the problem.

## Main results and outline

We first derive several SDP relaxations for the QCCP with increasing complexity. Our strongest SDP relaxation contains nonnegativity constraints and an additional subset of the facet-defining inequalities of the Boolean Quadric Polytope (BQP), which make it a powerful yet very difficult to solve relaxation. Since the derived relaxations are not Slater feasible, we study the geometry of their feasible sets. We identify the minimal face that contains the feasible set for one of the relaxations and show how to derive a sparse transformation matrix that performs the projection for all relaxations. It is nontrivial to obtain the transformation matrix since its construction is graph-specific. Therefore we propose a polynomial time algorithm that is based on the bipartite representation of the underlying graph, and provides a sparse transformation matrix.

To solve our SDP relaxation with nonnegativity constraints, we study the following two variants of the alternating direction augmented Lagrangian method; the (original) Alternating Direction Method of Multipliers (ADMM) and the Peaceman-Rachford splitting method (PRSM) that is also known as the symmetric ADMM. Although the ADMM is tested on SDP relaxations of different optimization problems, the PRSM with larger stepsizes was not implemented up to date for SDP relaxations. Our results show that the PRSM outperforms the classical ADMM. Therefore, we use the PRSM within our cutting plane augmented Lagrangian method (CP-ALM).

It is well-known that one can significantly improve an SDP model by adding the facet-defining inequalities of the BQP. It is also well-known that current SDP solvers have difficulties solving relaxations with those cuts. Here, we present a novel approach that overcomes the mentioned difficulties. In order to deal with the BQP inequalities, we exploit Dykstra’s projection algorithm that together with the PRSM forms the CP-ALM. We use Dykstra’s cyclic algorithm that is (partially) parallelized by clustering the set of BQP inequalities into subsets of nonoverlapping cuts. The CP-ALM also exploits warm starts each time new violated cuts are added. We present several other ingredients that improve the efficiency of the algorithm. Although it might seem that our algorithm is problem specific, all ingredients described in this paper can be easily extended for solving other optimization problems.
Finally, we present several upper bounding approaches that exploit matrices resulting from the CP-ALM. In our randomized undersampling algorithm we sample a partial solution and deterministically extend it to a full cycle cover. In randomized oversampling we iteratively draw a pair of successive arcs according to a distribution related to the SDP solution, until we obtain a cycle cover. Our final rounding approach is based on a distributed reinforcement learning technique, i.e., Q-learning. In particular, we let artificial agents learn how to find cycle covers by exploiting the SDP solution matrix such that the expected total reward is maximized. The latter approach provides the best upper bounds among all presented ones. Let us emphasize that it is challenging to find good feasible solutions for the QCCP, especially for large instances, since the considered graphs are not necessarily complete.

We provide extensive numerical tests on data sets used for the QCCP as well as data sets for the QTSP. Our bounds significantly outperform other bounds from the literature.

The paper is structured as follows. In Section 2 we formally introduce the QCCP and study the directed 2-factor polytope. In Section 3 we construct several SDP relaxations for the QCCP of increasing complexity. The Slater feasibility of the SDP relaxations is the topic of Section 4.1. Since transformation matrices used for projecting onto the minimal face are graph-specific, we provide a polynomial time algorithm for computing their sparse expressions in Section 4.2. In Section 5 we propose a new algorithm for solving the SDP relaxations that is based on a combination of the PRSM, Dykstra’s projection algorithm and a cutting plane method. Several upper bounding approaches are discussed in Section 6. Extended numerical results are given in Section 7.

Notation

A directed graph $G = (N,A)$ is given by a node set $N$ and an arc set $A \subseteq N \times N$, where $n = |N|$ and $m = |A|$. For all $i \in N$, let $\delta^+(i)$ and $\delta^-(i)$ denote the set of arcs leaving and entering node $i$, respectively. For all $S,T \subseteq N$, let $\delta^+(S,T)$ (resp. $\delta^-(S,T)$) denote the set of arcs going from $S$ to $T$ (resp. $T$ to $S$). The starting node of an arc $e \in A$ is denoted by $e^+$. Similarly, we denote by $e^-$ the ending node of $e$.

Let $0_n$ and $1_n$ be the $n \times 1$ vector of zeros and the $n \times 1$ vector of ones, respectively. Moreover, we denote the $i$-th elementary vector by $e_i$. Let $J_n$ and $I_n$ be the $n \times n$ matrix of all ones and the $n \times n$ identity matrix, respectively. In case the order of these vectors or matrices is clear, we omit the subscript to simplify notation.

In this paper we frequently work with extended matrices of the form $\begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix}$ for some $x \in \mathbb{R}^m$ and $X \in \mathbb{R}^{m \times m}$. We index the top row and top column of such extended matrix as row zero and column zero, respectively. Accordingly, we denote by $e_0$ a vector that has a one on the first position and all other elements zero in the context of extended matrices.

For any matrix $M \in \mathbb{R}^{m \times n}$, let $\text{Col}(M)$ be the linear space spanned by the columns of $M$. The null space of $M$ is denoted by $\text{Nul}(M)$. For all $M,N \in \mathbb{R}^{m \times n}$, the Hadamard product $M \circ N$ equals the entrywise product of $M$ and $N$, i.e., $(M \circ N)_{ij} = M_{ij}N_{ij}$. Moreover, for all square matrices $M$, the operator $\text{Diag} : \mathbb{R}^{n \times n} \to \mathbb{R}^n$ maps a matrix to a vector consisting of its diagonal elements. Its adjoint operator is denoted by $\text{Diag} : \mathbb{R}^n \to \mathbb{R}^{n \times n}$. The trace of a square matrix $M$ is given by $\text{tr}(M)$.

Let $S^n$ 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 $S^n_+ := \{ M \in S^n : M \succeq 0 \}$. Let $\langle \cdot, \cdot \rangle$ denote the trace inner product. That is, for any $M,N \in \mathbb{R}^{m \times m}$, we define $\langle M,N \rangle := \text{tr}(M^T N) = \sum_{i=1}^m \sum_{j=1}^m M_{ij}N_{ij}$. Its associated norm is the Frobenius norm, denoted by $\|M\|_F := \sqrt{\text{tr}(M^T M)}$. 

3
2 The Quadratic Cycle Cover Problem

In this section we formally introduce the asymmetric version of the quadratic cycle cover problem. Moreover, we introduce the directed 2-factor polytope and consider some of its properties.

The quadratic cycle cover problem (QCCP) is the problem of finding a set of node-disjoint cycles covering all the nodes such that the sum of interaction costs between successive arcs is minimized. Since we assume that all cycle covers in this paper are disjoint, we use the term cycle cover to denote this concept in the sequel. An instance of the QCCP is specified by the pair \((G, Q)\), where \(G = (N,A)\) is a simple directed graph with \(n = |N|\) nodes and \(m = |A|\) arcs and \(Q = (q_{ef}) \in \mathbb{R}^{m \times m}\) is a quadratic cost matrix. We assume that the entries of \(Q\) are such that \(q_{ef} = 0\) if arc \(f\) is not a successor of arc \(e\), i.e., if \(f \notin \delta^+(e^-)\).

Let \(x \in \{0,1\}^m\) represent the characteristic vector of a cycle cover. That is, \(x_e = 1\) if arc \(e\) belongs to the cycle cover and \(x_e = 0\) otherwise. Then the QCCP can be formulated as:

\[
OPT(Q) := \min \{x^\top Qx : x \in P\},
\]

where \(P\) denotes the set of all cycle covers in \(G\), i.e.,

\[
P := \left\{ x \in \{0,1\}^m : \sum_{e \in \delta^+(i)} x_e = \sum_{e \in \delta^-(i)} x_e = 1 \quad \forall i \in N \right\}.
\]

In the literature, a cycle cover in a directed graph is also called a directed 2-factor. For the existence of a directed 2-factor in a graph, see e.g., Chiba and Yamashita [17]. The QCCP is shown to be \(\mathcal{NP}\)-hard in the strong sense and not approximable within any constant factor [25, 43].

The linear problem corresponding to the QCCP is called the cycle cover problem (CCP). Given a linear arc-weight function, the CCP asks for a minimum weight cycle cover in \(G\). The CCP reduces to the well-known linear assignment problem, see e.g., [14], and is therefore polynomial time solvable.

Let \(\text{Conv}(P)\) be the convex hull of all characteristic vectors corresponding to directed 2-factors in \(G\). We call this set the directed 2-factor polytope. Let \(U \in \mathbb{R}^{n \times m}\) and \(V \in \mathbb{R}^{n \times m}\) be defined as

\[
U_{i,e} := \begin{cases} 1 & \text{if arc } e \text{ starts at node } i \\ 0 & \text{otherwise} \end{cases} \quad V_{i,e} := \begin{cases} 1 & \text{if arc } e \text{ ends at node } i \\ 0 & \text{otherwise}. \end{cases}
\]

Additionally, let \(u_i^\top\) and \(v_i^\top\) denote the \(i\)-th row of \(U\) and \(V\), respectively. Thus, \(P = \{x \in \{0,1\}^m : [U^\top, V^\top]^\top x = 1_{2n}\}\). It follows from the total unimodularity of \([U^\top, V^\top]^\top\) that the directed 2-factor polytope can be written explicitly as:

\[
\text{Conv}(P) = \left\{ x \in \mathbb{R}^m : x \geq 0_m, \begin{bmatrix} U \\ V \end{bmatrix} x = 1_{2n} \right\}.
\]

Observe that the arcs that are never used in a cycle cover are irrelevant for the QCCP. We define the set \(\mathcal{J}\) consisting of all arcs with this property, i.e.,

\[
\mathcal{J} := \{ f \in A : x_f = 0 \text{ for all } x \in P \}.
\]

The elements in \(\mathcal{J}\) can be obtained in polynomial time by solving for each \(f \in A\) the following CCP:

\[
z_f := \max \{ e_f^\top x : x \in P \}.
\]

The set \(\mathcal{J}\) consists of all arcs \(f \in A\) for which \(z_f = 0\). Without loss of generality, we can remove the arcs that are in \(\mathcal{J}\) from the given instance to simplify the problem. This leads to the following assumption that applies to the rest of this paper.

\[4\]
Taking the product of the linear constraints, we distinguish three types of squared linear constraints that are summarized in Table 1.

\[ \alpha := \text{rank} \left( \begin{bmatrix} U \end{bmatrix} \right) \tag{4} \]

In Section 4.2, we derive the value of \( \alpha \) in terms of the graph. For now, we note that \( n \leq \alpha \leq 2n - 1 \), provided that Assumption 1 holds.

It follows from the rank-nullity theorem that \( \text{dim}(\text{Nul}([U^T, V^T]^T)) = m - \alpha \). Let us prove the following lemma.

**Lemma 1.** Under Assumption 1, the dimension of the directed 2-factor polytope equals \( m - \alpha \).

**Proof.** It follows from (4) that

\[ \text{Conv}(P) = \{ x \in \mathbb{R}^m : [U^T, V^T]^T x = 1_{2n} \} \cap \mathbb{R}^m_+. \]

Obviously, the set \( \mathbb{R}^m_+ \) is full-dimensional, whereas the dimension of \( \{ x \in \mathbb{R}^m : [U^T, V^T]^T x = 1_{2n} \} \) equals \( \text{dim}(\text{Nul}([U^T, V^T]^T)) = m - \alpha \). Hence, we have \( \text{dim}(\text{Conv}(P)) \leq m - \alpha \), where strict inequality holds only if \( \text{Conv}(P) \) is fully contained in one of the facets of \( \mathbb{R}^m_+ \). Now assume for the sake of contradiction that there exists some arc \( e \) such that \( \text{Conv}(P) \subseteq \{ x \in \mathbb{R}^m_+ : x_e = 0 \} \). Consequently, we must have \( P \subseteq \{ x \in \mathbb{R}^m_+ : x_e = 0 \} \), which implies that \( e \in \mathcal{J} \). This contradicts Assumption 1. We conclude that \( \text{dim}(\text{Conv}(P)) = m - \alpha \). \( \square \)

## 3 SDP relaxations for the QCCP

In this section, we focus on constructing several semidefinite programming relaxations for the QCCP. These relaxations increase in strength and complexity.

Using the trace inner product, the objective function of (1) can be rewritten as \((Q, xx^\top) = \langle Q, X \rangle\), where we replace \( xx^\top \) by a matrix variable \( X \in S^m \). We now relax the equality \( X - xx^\top = 0 \) by replacing it by the SDP constraint \( X - xx^\top \succeq 0 \). It follows from the Schur complement that we can equivalently write \( \begin{bmatrix} 1 & x^\top \\ x & X \end{bmatrix} \succeq 0 \). Moreover, since \( x \in P \) is a binary vector, we have \( \text{diag}(X) = x \).

This leads to the following basic feasible set for an SDP relaxation of the QCCP:

\[ \mathcal{F}_{\text{basis}} = \left\{ \begin{bmatrix} 1 \\ x \\ X \end{bmatrix} \in S^{m+1} : \begin{bmatrix} u_i^\top x = v_i^\top x = 1 \ (\forall i \in N) \\ x = \text{diag}(X) \end{bmatrix}, \begin{bmatrix} 1 \\ x^\top \\ X \end{bmatrix} \succeq 0 \right\}. \tag{5} \]

We show below how to strengthen the feasible set by adding valid constraints.

Since each cycle cover consists of \( n \) arcs, we know that \( 1^\top_n x = n \) for all \( x \in P \). This can be written equivalently as \( \text{tr}(X) = n \), which we refer to as the trace constraint. Moreover, since \( xx^\top \) is replaced by \( X \), the constraint \( \langle J, X \rangle = n^2 \), which we call the all-ones constraint, is also valid.

One can also add to \( \mathcal{F}_{\text{basis}} \) the so-called squared linear constraints. These constraints result from taking the product of the linear constraints \( u_i^\top x = 1 \) and \( u_j^\top x = 1 \) for all \( i, j \in N \), which yield \( 1 = (u_i^\top x)(x^\top u_j) = \langle u_i u_j^\top, xx^\top \rangle \). Hence, the constraint \( \langle u_i u_j^\top, X \rangle = 1 \) is valid for \( \mathcal{F}_{\text{basis}} \). The same can be done by taking the products of the linear constraints \( v_i^\top x = 1 \) for all \( i \in N \), etc. In total, we distinguish three types of squared linear constraints that are summarized in Table 1.

| Type of squared linear constraint | Constraints on \( X \) |
|----------------------------------|--------------------------|
| Type I                           | \( \langle u_i u_i^\top, X \rangle = 1 \) and \( \langle v_i v_i^\top, X \rangle = 1 \) for all \( i \in N \); |
| Type II                          | \( \langle u_i u_j^\top, X \rangle = 1 \) and \( \langle v_i v_j^\top, X \rangle = 1 \) for all \( i, j \in N, i \neq j \); |
| Type III                         | \( \langle u_i v_j^\top, X \rangle = 1 \) for all \( i, j \in N \). |

Table 1: Three types of valid squared linear constraints for \( \mathcal{F}_{\text{basis}} \).
We show below how the above mentioned valid constraints relate. An interesting result is that the squared linear constraints of Type II and III turn out to be redundant when the Type I constraints and some other constraints are added to (5).

**Proposition 1.** Let \( x \in \mathbb{R}^m \) and \( X \in \mathcal{S}^m \) be such that \( \begin{pmatrix} 1 \\ x^T \\ X \end{pmatrix} \succeq 0, \ \text{diag}(X) = x, \ \text{tr}(X) = n \) and \( \langle J, X \rangle = n^2 \). If \( \langle u_i u_i^T, X \rangle = \langle v_i v_i^T, X \rangle = 1 \) for all \( i \in N \), then

(i) the squared linear constraints of Type II and III are redundant;

(ii) the linear constraints \( u_i^T x = u_j^T x = 1 \) for all \( i, j \in N \) are redundant.

**Proof.**

(i) Let \( i, j \in N \) with \( i \neq j \), then we have,

\[
\langle (u_i - u_j)(u_i - u_j)^T, X \rangle = \langle u_i u_i^T + u_j u_j^T - 2u_i u_j^T, X \rangle = \langle u_i u_i^T, X \rangle + \langle u_j u_j^T, X \rangle - 2\langle u_i u_j^T, X \rangle = 2 - 2\langle u_i u_j^T, X \rangle \geq 0,
\]

since \( \langle u_i u_j^T, X \rangle = \langle u_i u_j^T, X \rangle = 1 \) and \( X \succeq 0 \). From this it follows that \( \langle u_i u_j^T, X \rangle \leq 1 \).

Conversely, as all arcs have exactly one starting node, we have \( \mathbf{1}_m = \sum_{i \in N} u_i \). Using this, we can rewrite the matrix \( J \) as \( J = (\sum_{i \in N} u_i)(\sum_{i \in N} u_i)^T \) and the constraint \( \langle J, X \rangle = n^2 \) as follows:

\[
n^2 = \langle J, X \rangle = \langle \left( \sum_{i \in N} u_i \right) \left( \sum_{i \in N} u_i \right)^T, X \rangle = \sum_{i \in N} \sum_{j \in N} \langle u_i u_j^T, X \rangle.
\]

The right-hand side expression is a sum of \( n^2 \) elements for which \( \langle u_i u_j^T, X \rangle \leq 1 \) for all \( i, j \in N \). Since the sum has to be equal to \( n^2 \), it follows that \( \langle u_i u_j^T, X \rangle = 1 \) for all \( i, j \in N, i \neq j \). The other equalities can be proven in a similar fashion.

(ii) Let \( Y := X - xx^T \). By the Schur complement, we know that \( Y \succeq 0 \). Now,

\[
1 = \langle u_i u_i^T, X \rangle = \langle u_i u_i^T, Y \rangle + \langle u_i u_i^T, xx^T \rangle = \langle u_i u_i^T, Y \rangle + \langle u_i x \rangle^2.
\]

Since \( \langle u_i u_i^T, Y \rangle \succeq 0 \), it follows that \( \langle u_i x \rangle^2 \leq 1 \) and, consequently, \( u_i^T x \leq 1 \) for all \( i \in N \).

Now we rewrite the trace constraint \( \mathbf{1}_m^T x = n \) as

\[
\mathbf{1}_m^T x = \left( \sum_{i \in N} u_i \right)^T x = \sum_{i \in N} u_i^T x = n.
\]

Since each term \( u_i^T x \) is bounded by \( 1 \), equality is established only when \( u_i^T x = 1 \) for all \( i \in N \). In a similar way one can show that \( v_i^T x = 1 \) for all \( i \in N \).

Observe that there exist \( 2n \) constraints of Type I. We show how to merge these constraints to obtain a more compact formulation. To this end, we define the matrices \( \tilde{U}, \tilde{V} \in \mathbb{R}^{m \times m} \) as follows:

\[
\tilde{U} := \sum_{i \in N} \begin{pmatrix} -1 \\ u_i \end{pmatrix} \begin{pmatrix} -1 \\ u_i \end{pmatrix}^T \quad \text{and} \quad \tilde{V} := \sum_{i \in N} \begin{pmatrix} -1 \\ v_i \end{pmatrix} \begin{pmatrix} -1 \\ v_i \end{pmatrix}^T.
\]

We establish the following result.

**Proposition 2.** Let \( x \in \mathbb{R}^m \) and \( X \in \mathcal{S}^m \) be such that \( \begin{pmatrix} 1 \\ x^T \\ X \end{pmatrix} \succeq 0 \) and \( \text{diag}(X) = x \). Then the following statements are equivalent:

(i) \( \text{tr}(X) = n \) and \( \langle u_i u_i^T, X \rangle = \langle v_i v_i^T, X \rangle = 1 \) for all \( i \in N \);
(ii) \( \langle \tilde{U}, \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \rangle = \langle \tilde{V}, \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \rangle = 0. \)

Proof. It is not difficult to see that (i) \( \Rightarrow \) (ii). We now show the reverse statement. We have

\[
\langle \tilde{U}, \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \rangle = \left\langle \sum_{i \in N} (-1) \begin{pmatrix} -1 \\ u_i \end{pmatrix}^T, \begin{pmatrix} 1 \\ x \end{pmatrix}^T \right\rangle = \sum_{i \in N} (u_i^T X u_i - 2u_i^T x + 1) = 0.
\]

Since \( \begin{pmatrix} -1 \\ u_i \end{pmatrix} \begin{pmatrix} -1 \\ u_i \end{pmatrix}^T \geq 0 \), it follows that \( u_i^T X u_i - 2u_i^T x + 1 \geq 0 \) for all \( i \in N \). Combining this with the equality above, we conclude that \( u_i^T X u_i - 2u_i^T x + 1 = 0 \) for all \( i \in N \).

Now define \( Y := X - xx^T \geq 0 \). Then \( u_i^T X u_i - 2u_i^T x + 1 = 0 \) can be rewritten as

\[
u_i^T (Y + xx^T) u_i - 2u_i^T x + 1 = 0 \iff u_i^T Y u_i + (u_i^T x - 1)^2 = 0.
\]

Since \( u_i^T Y u_i \geq 0 \) and \( (u_i^T x - 1)^2 \geq 0 \), it follows that \( u_i^T x = 1 \), which in turn implies that \( u_i^T X u_i = 2u_i^T x - 1 = 1 \). Similarly, one can prove that \( \langle v_i u_i^T, X \rangle = 1 \) for all \( i \in N \).

Finally, since \( 1_m = \sum_{i \in N} u_i \), we have

\[
\text{tr}(X) = 1_m^T x = \sum_{i \in N} u_i^T x = n.
\]

We conclude that (ii) \( \Rightarrow \) (i). \( \square \)

Proposition 2 shows that instead of the trace constraint and the squared linear constraints, we can equivalently include the merged squared linear constraints. Let us now define the following set:

\[
\mathcal{F}_1 := \left\{ \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \in \mathcal{S}^{n+1} : \langle J, X \rangle = n^2, \quad \langle \tilde{U}, \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \rangle = \langle \tilde{V}, \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \rangle = 0, \right. \]

\[
\text{diag}(X) = x, \quad \begin{pmatrix} 1 \\ x^T \\ X \end{pmatrix} \succeq 0 \}.
\]

(6)

From the above discussion it follows that \( \mathcal{F}_1 \subseteq \mathcal{F}_\text{basis} \). Let us now introduce our first SDP relaxation:

\[
\text{(SDP}_1) \quad \min \left\{ \langle Q, X \rangle : \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \in \mathcal{F}_1 \right\}.
\]

(7)

In the sequel we show how to improve SDP relaxation \( \text{(SDP}_1) \). Let us exploit the structure of a cycle cover to identify a zero pattern in \( X \). For each \( i \in N \), we know that there is exactly one arc \( e \in \delta^+(i) \) with \( x_e = 1 \) and \( x_f = 0 \) for all other arcs \( f \) leaving \( i \). Hence, for each pair of distinct arcs \( e, f \in \delta^+(i) \) we have \( x_e x_f = 0 \). This leads to the valid constraint \( X_{ef} = 0 \) for all \( e, f \in \delta^+(i), e \neq f \). The same holds for the incoming arcs. We call these type of equalities the zero-structure constraints.

We define:

\[
\mathcal{Z} := \{ (e, f) \in A \times A : e \text{ and } f \text{ start or end at the same node, } e \neq f \}.
\]

(8)

Then the zero-structure constraints read that \( X_{ef} = 0 \) for all \( (e, f) \in \mathcal{Z} \).

Note that one may also add the nonnegativity constraints on matrix variables in \( \text{(SDP}_1) \). For that purpose, we define the cone of nonnegative symmetric matrices, i.e.,

\[
\mathcal{N}_+^m := \{ X \in \mathcal{S}^m : X \geq 0 \}.
\]

We show next that after adding nonnegativity constraints to the feasible set of \( \text{(SDP}_1) \), the zero-structure constraints turn out to be redundant.

**Proposition 3.** Let \( x \in \mathbb{R}^m \) and \( X \in \mathcal{S}^m \) be feasible for \( \text{(SDP}_1) \). If \( X \in \mathcal{N}_+^m \), then \( X_{ef} = 0 \) for all \( (e, f) \in \mathcal{Z} \).
Proof. We prove the statement for the outgoing arcs. The proof for the incoming arcs is similar. Using Proposition 2, we know that \( \langle u_i u_i^T, X \rangle = 1 \) for all \( i \in N \). We rewrite this equality as:

\[
1 = \langle u_i u_i^T, X \rangle = \sum_{e \in A} \sum_{f \in A} (u_i)_e (u_i)_f X_{ef} = \sum_{e \in \delta^+(i)} X_{ee} + \sum_{e, f \in \delta^+(i), e \neq f} X_{ef}.
\]

Since \( \text{diag}(X) = x \), we have \( \sum_{e \in \delta^+(i)} X_{ee} = \sum_{e \in \delta^+(i)} x_e = u_i^T x = 1 \), where the last equality follows from Proposition 1. Thus, we have \( \sum_{e, f \in \delta^+(i), e \neq f} X_{ef} = 0 \), from where it follows that \( X_{ef} = 0 \) for all \( e, f \in \delta^+(i) \) with \( e \neq f \).

Let us now define our next, tighter SDP relaxation:

\[
\text{(SDP}_2) \quad \min \left\{ \langle Q, X \rangle : \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \in F_1 \cap \mathcal{N}_+^{m+1} \right\}.
\]  

To further strengthen (SDP\(_2\)), we consider an additional set of valid inequalities. Namely, we consider cuts that are related to the well-known Boolean Quadric Polytope introduced by Padberg. The BQP of order \( m \) is defined as

\[
\text{BQ}^m := \text{Conv} \left\{ (x, X) \in \mathbb{R}^m \times \mathbb{R}^{m(m-1)/2} : x \in \{0,1\}^m, X_{ij} = x_i x_j \forall 1 \leq i < j \leq m \right\}.
\]

Since the matrix \( X \) in our previous relaxations is such that \( X_{ef} \) represents \( x_e x_f \), the inequalities that are valid for \( \text{BQ}^m \) are also valid for our SDP relaxations. In [10] it is proven that the following triangle inequalities (written in our QCCP notation) define facets of \( \text{BQ}^m \):

\[
X_{ef} + X_{eg} \leq x_e + X_{fg} \quad \text{for all } e, f, g \in A, e \neq f, f \neq g, e \neq g.
\]

Although there are more facet-defining inequalities for the BQP, we consider only the above mentioned ones in this paper. Namely, our preliminary tests show that the triangle inequalities lead to the largest improvement of the SDP bounds. Note that there are \( \mathcal{O}(m^3) \) triangle inequalities, and that it is challenging to solve even medium-size SDPs that include all triangle inequalities.

Let \( T \subseteq A \times A \times A \) denote the set of arc triples corresponding to the triangle inequalities, and \( \mathcal{C}(T) \) be the polyhedron induced by these cuts, i.e.,

\[
\mathcal{C}(T) := \left\{ \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \in \mathcal{N}^{m+1} : X_{ef} + X_{eg} \leq X_{ee} + X_{fg} \forall (e, f, g) \in T \right\},
\]

where we incorporated the fact that \( x_e = X_{ee} \) for all \( e \in A \) in our relaxations. Then our strongest SDP relaxation is:

\[
\text{(SDP}_3) \quad \min \left\{ \langle Q, X \rangle : \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \in F_1 \cap \mathcal{N}_+^{m+1} \cap \mathcal{C}(T) \right\}.
\]

By abuse of notation, we will also use \( T \) to denote a subset of the set of arc triples corresponding to the triangle inequalities within a cutting plane environment.

## 4 Graph-dependent facial reduction

In this section we investigate the Slater feasibility of the relaxations constructed in Section 3. We prove that the relaxations are not Slater feasible and show how to obtain facially reduced relaxations. We conclude this section by providing an algorithm that computes a sparse transformation matrix required for the facial reduction. Each transformation matrix is graph specific, and the algorithm exploits the bipartite representation of the underlying graph. Our algorithm can be downloaded and used whenever one needs to compute a basis for the flow space of the bipartite representation of a directed graph.

\[\text{[The code can be downloaded from www.frankdemeijer.org]}\]
4.1 Strict feasibility by facial reduction

Recall that Slater’s constraint qualification holds for an SDP relaxation if there exists a feasible solution that is also positive definite. The following lemma shows that Slater’s constraint qualification does not hold for the SDP relaxation (7), and consequently, neither for (9) and (10).

**Lemma 2.** Let \( Y := \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \in S^{m+1}_+ \) be feasible for (SDP1). Then

\[
\text{Span}\left\{\begin{pmatrix} -1 \\ u_i \\ v_i \end{pmatrix} : i \in N \right\} \subseteq \text{Nul}(Y).
\]

**Proof.** It follows directly from the fact that \( \langle \tilde{U}, Y \rangle = 0 \) and positive semidefinite matrices having a nonnegative inner product that we have

\[
(-1) u_i \begin{pmatrix} 1 \\ x^T \\ X \end{pmatrix} (-1) u_i = 0,
\]

for all \( i \in N \). Since \( Y \succeq 0 \), this implies that \( \begin{pmatrix} 1 \\ x^T \\ X \end{pmatrix} (-1) u_i = 0 \) for all \( i \in N \). Thus, \( (-1) u_i \in \text{Nul}(Y) \) for all \( i \in N \). Similarly, one can prove that \( (-1) v_i \in \text{Nul}(Y) \) for all \( i \in N \). \( \square \)

Lemma 2 shows that our SDP relaxations are not Slater feasible. Thus, the feasible sets of the SDP relaxations are fully contained in one of the faces of \( S^{m+1}_+ \). For now, we only focus on the relaxation \( (\text{SDP1}) \). In order to find an equivalent relaxation for \( (\text{SDP1}) \) that is Slater feasible, we project the problem onto the minimal face containing the feasible set, i.e., apply facial reduction, see e.g., [11, 20, 54].

To find the minimal face containing the feasible set of the SDP relaxation, one needs to find its exposing vectors, i.e., the vectors orthogonal to the feasible set of the SDP relaxation. It follows from Lemma 2 that the following matrices satisfy that property:

\[
\begin{pmatrix} -1 \\ u_i \\ u_i \end{pmatrix} \begin{pmatrix} -1 \\ u_i \\ u_i \end{pmatrix}^T \quad \text{and} \quad \begin{pmatrix} -1 \\ v_i \\ v_i \end{pmatrix} \begin{pmatrix} -1 \\ v_i \\ v_i \end{pmatrix}^T \quad \text{for all } i \in N.
\] (11)

Now, let \( R \) be defined as follows:

\[
R := \left( \text{Span}\left\{\begin{pmatrix} -1 \\ u_i \\ v_i \end{pmatrix} : i \in N \right\} \right)^\perp = \text{Nul}\left( \begin{bmatrix} -1_n & U \\ -1_n & V \end{bmatrix} \right).
\] (12)

Observe that under Assumption 1, the rank of \( \begin{bmatrix} -1_n & U \\ -1_n & V \end{bmatrix} \) equals the rank of \( [U^T, V^T]^T \) which we defined to be \( \alpha \), see (4). From this it follows that \( \dim(R) = m + 1 - \alpha \). We now define \( F_R \) to be the subset of \( S^{m+1}_+ \) that is orthogonal to the exposing vectors (11), i.e.,

\[
F_R := \{ X \in S^{m+1}_+ : \text{Col}(X) \subseteq R \}.
\]

Since faces of \( S^{m+1}_+ \) are known to be in correspondence with linear subspaces of \( \mathbb{R}^{m+1} \), \( F_R \) is a face of \( S^{m+1}_+ \) containing the feasible set of (SDP1). Later on we show that \( F_R \) is actually the minimal face with this property, see Theorem 2.

In order to derive an explicit expression of \( F_R \), let \( W \in \mathbb{R}^{(m+1) \times (m+1-\alpha)} \) be a matrix whose columns form a basis for \( R \). Then the face \( F_R \) can be equivalently written as:

\[
F_R = W S^{m+1-\alpha}_+ W^T.
\] (13)

This implies that any \( Y \in S^{m+1}_+ \) that is feasible for (SDP1) can be written as \( Y = W Z W^T \) for some \( Z \in S^{m+1-\alpha}_+ \). By substituting this term into (SDP1), we obtain an equivalent relaxation in a lower
dimensional relaxation. As a direct byproduct, some of the original constraints become redundant. The resulting relaxation is as follows:

\[
(SDP_{S1}) \quad \min \left\{ (W^TQW, Z) : \text{diag}(ZW^T) = WZW^Te_0, e_0^TWZW^Te_0 = 1, Z \succeq 0 \right\},
\]

where \( \hat{Q} := \begin{pmatrix} 0 \\ 0_m \\ 0_m^T/Q \end{pmatrix} \). Let us define the feasible set of the above relaxation for future reference:

\[
\mathcal{F}_{S1} := \{ Z \in S^{m+1-\alpha}_+ : \text{diag}(ZW^T) = WZW^Te_0, e_0^TWZW^Te_0 = 1, Z \succeq 0 \}.
\]

We show below that the SDP relaxations (14) and (7) are equivalent.

**Theorem 1.** The SDP relaxation (SDP_{S1}) is equivalent to the SDP relaxation (SDP_1).

**Proof.** Let \( Z \) be feasible for (SDP_{S1}) and define \( Y := WZW^T, X := (0_m, I_m)Y(0_m, I_m)^T \) and \( x := \text{diag}(X) \). Our goal is to show that \( x \) and \( X \) are feasible for (SDP_1).

Note that the SDP constraint is trivially satisfied. Therefore, it remains to prove that the all-ones constraint and the merged squared linear constraints hold. Observe that

\[
\begin{pmatrix} n \\ 1_m \end{pmatrix}^T \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \begin{pmatrix} n \\ 1_m \end{pmatrix} = \begin{pmatrix} n \\ 1_m \end{pmatrix}^T WZW^T \begin{pmatrix} n \\ 1_m \end{pmatrix} = \sum_{i \in N} \begin{pmatrix} n \\ 1_m \end{pmatrix}^T WZW^T \begin{pmatrix} -1 \\ u_i \end{pmatrix} = 0,
\]

where the last equality follows from the construction of \( W \). Since the most left term in the expression above equals \(-n^2 + \sum_{i \in N} X_{1i} \), it follows that \( \langle J, X \rangle = n^2 \).

Next, we have

\[
\langle \hat{U}, Y \rangle = \left\langle \sum_{i \in N} \begin{pmatrix} -1 \\ u_i \end{pmatrix} \begin{pmatrix} -1 \\ u_i \end{pmatrix}^T, WZW^T \right\rangle = \sum_{i \in N} \left\langle W^T \begin{pmatrix} -1 \\ u_i \end{pmatrix} \begin{pmatrix} -1 \\ u_i \end{pmatrix}^T, W, Z \right\rangle = 0,
\]

since the columns of \( W \) are orthogonal to \((-1, u_i^T)^T\) for all \( i \in N \). In a similar fashion we can show that \( \langle \tilde{V}, Y \rangle = 0 \) for all \( i \in N \). We conclude that the matrix \( X \) and vector \( x \) obtained from (SDP_{S1}) are feasible for (SDP_1).

Conversely, let \( Y \) be feasible for (SDP_1). Then it follows from (13) that there exists a matrix \( Z \succeq 0 \) such that \( Y = WZW^T \). Since the objective functions of (SDP_1) and (SDP_{S1}) coincide, we conclude that the two relaxations are equivalent.

\( \square \)

We now prove that (SDP_{S1}) is indeed Slater feasible, see also [54].

**Theorem 2.** The relaxations (SDP_{S1}) contains a Slater feasible point.

**Proof.** Since Conv(\( P \)) has dimension \( m - \alpha \), see Lemma [4] it follows that there exists an affinely independent set of vectors \( \{x_1, ..., x_{m+1-\alpha}\} \subseteq P \). Because of the affinely independence of these vectors, the set

\[
\left\{ \begin{pmatrix} 1 \\ x_1 \end{pmatrix}, \begin{pmatrix} 1 \\ x_2 \end{pmatrix}, ..., \begin{pmatrix} 1 \\ x_{m+1-\alpha} \end{pmatrix} \right\}
\]

is linearly independent in \( \mathbb{R}^{m+1} \). Since \((1, x_i^T) \in \mathcal{R} \) for all \( i = 1, ..., m+1-\alpha \) and \( W \) forms a basis for \( \mathcal{R} \), there exist vectors \( y_1, ..., y_{m+1-\alpha} \in \mathbb{R}^{m+1-\alpha} \) such that \( W^T y = (1, x_i^T)^T \) for all \( i = 1, ..., m+1-\alpha \). Moreover, the vectors \( y_i \) are linearly independent in \( \mathbb{R}^{m+1-\alpha} \) because of the linear independence of the vectors \( W^T y_i \).

We define

\[
Z_\lambda := \sum_{i=1}^{m+1-\alpha} \lambda_i y_i^T,
\]

10
where \( \lambda_i \geq 0 \) for all \( i = 1, \ldots, m+1-\alpha \) and \( 1^\top \lambda = 1 \), and rewrite \( WZ_\lambda W^\top \) as follows:

\[
WZ_\lambda W^\top = \sum_{i=1}^{m+1-\alpha} \lambda_i Wy_i(Wy_i)^\top = \sum_{i=1}^{m+1-\alpha} \lambda_i \begin{pmatrix} 1 \\ x_i \end{pmatrix} \begin{pmatrix} 1 \\ x_i \end{pmatrix}^\top.
\]

It is not difficult to see that \( Z_\lambda \) is feasible for \((SDP_{S_1})\). By taking \( \lambda_i > 0 \) for all \( i = 1, \ldots, m+1-\alpha \), the resulting matrix \( Z_\lambda \) is non-singular, which implies that \( Z_\lambda > 0 \). Hence, \((SDP_{S_1})\) contains a Slater feasible point.

Note that the key in the proof of Theorem 2 is the known dimension of \( \text{Conv}(P) \).

Continuing in the same vein, one can show that the following SDP relaxation is equivalent to the SDP relaxation \((9)\):

\[
(SDP_{S_2}) \quad \min \left\{ \langle W^\top \hat{Q}W, Z \rangle : \; Z \in \mathcal{F}_{S_1}, \; WZW^\top \in \mathcal{N}^{m+1}_+ \right\},
\]

and the following relaxation equivalent to the SDP relaxation \((10)\):

\[
(SDP_{S_3}) \quad \min \left\{ \langle W^\top \hat{Q}W, Z \rangle : \; Z \in \mathcal{F}_{S_1}, \; WZW^\top \in \mathcal{N}^{m+1}_+ \cap \mathcal{C}(T) \right\}.
\]

### 4.2 A polynomial time algorithm for the transformation matrix

Although the subspace \( \mathcal{R} \) has been defined algebraically in Section 4.1, we now focus on its relation with the graph \( G \). This leads to a polynomial time algorithm for computing a sparse transformation matrix \( W \) that depends on the considered graph. Although one can compute \( W \) numerically, we require its sparse expression for efficient implementation of our cutting plane algorithm, see Section 5.

Recall that the columns of \( W \) form a basis for the subspace \( \mathcal{R} \), see \((12)\). A natural way to construct \( W \) is as follows: let \( \bar{x} \in P \) be the characteristic vector of any cycle cover in \( G \). Moreover, let \( \bar{W} \in \mathbb{R}^{m \times (m-\alpha)} \) be a matrix whose columns form a basis for \( \text{Nul}([U^\top, V^\top]^\top) \). Then, the matrix

\[
W := \begin{bmatrix} 1 & 0^\top_{m-\alpha} \\ \bar{x} & \bar{W} \end{bmatrix}
\]

(18)

forms a basis for the subspace \( \mathcal{R} \). Finding a sparse expression for \( W \) now boils down to finding a sparse expression for \( \bar{W} \). For that purpose, we focus on a graph \( B(G) \) that is induced by \( G \), the so-called bipartite representation of \( G \), which is introduced by Bang-Jensen and Gutin \cite{7}. The graph \( B(G) = (V_1 \cup V_2, E) \) is an undirected bipartite graph where \( V_1 \) and \( V_2 \) are copies of the set \( N \) and the edge set \( E \) is defined as:

\[
E = \{ (i, j) \in V_1 \times V_2 : \; (i, j) \in A \}.
\]

By construction, each arc in \( G \) corresponds to exactly one edge in \( B(G) \), where the orientation in \( G \) determines the configuration of the edges in \( B(G) \). Figure 1 shows an example of \( G \) and its corresponding bipartite representation \( B(G) \). Observe that a cycle cover in \( G \) corresponds to a perfect matching in \( B(G) \) and vice versa.

The matrix \( [U^\top, V^\top]^\top \) equals the incidence matrix of \( B(G) \). Suppose we orient all edges of \( B(G) \) from \( V_1 \) to \( V_2 \). The incidence matrix with respect to this orientation equals \([U^\top, -V^\top]^\top\). Clearly, we have \( \text{Nul}([U^\top, V^\top]^\top) = \text{Nul}([U^\top, -V^\top]^\top) \). The null space of the incidence matrix of a directed graph is in the literature known as the flow space of a graph. Hence, it follows that the columns of \( \bar{W} \) form a basis for the flow space of the bipartite representation of \( G \) (with respect to the orientation from \( V_1 \) to \( V_2 \)).

Let \( C \) be a cycle in \( B(G) \). Since \( B(G) \) is a bipartite graph, \( C \) consists of an even number of edges. Let \( z \in \mathbb{R}^m \) denote its signed characteristic vector, i.e., we alternately assign values +1 and −1 to the edges on \( C \) and assign value 0 otherwise. It is well-known that the flow space of a graph
is spanned by the signed characteristic vectors of all its cycles. For more information about the flow space of a graph, we refer to e.g., [31].

Hence, $\mathcal{R}$ is related to the cycles of the bipartite representation of $G$. A natural question is how do the cycles of $B(G)$ relate to the original graph $G$? To answer this question, we exploit the notion of a closed antidirected trail, which is introduced in [6]. Recall that a trail is a walk in a graph that does not contain repeated arcs, but is allowed to contain repeated nodes. A closed trail is a trail that has the same start and ending node.

**Definition 1.** A closed antidirected trail (CAT) in a directed graph $G$ is a closed trail of even length with arcs oriented alternately.

A cycle in $B(G)$ corresponds to a CAT in $G$. To verify this, let $\phi : E \to A$ be the bijection between the edges of $B(G)$ and the arcs of $G$ in the natural way. Then, $C$ equals a cycle in $B(G)$ if and only if $\phi(C)$ equals a CAT in $G$. Obviously, since $C$ starts and ends at the same vertex in $B(G)$, $\phi(C)$ also starts and ends at the same node in $G$. Moreover, since $B(G)$ is bipartite, $C$ and thus $\phi(C)$ must be of even length. Finally, each two consecutive edges of $C$ have one common vertex in $V_1$ (resp. $V_2$) and the other vertex in $V_2$ (resp. $V_1$). By construction of $B(G)$, it follows that two consecutive edges of $C$ correspond to alternately oriented arcs in $G$. Thus $\phi(C)$ is a CAT. The reverse statement can be shown in the same fashion. This leads to the following proposition.

**Proposition 4.** The flow space of $B(G)$ equals the subspace spanned by the closed antidirected trails in $G$.

We now have two interpretations of the column space of $\overline{W}$, one with respect to $G$ and the other with respect to $B(G)$. The latter one is more suitable for finding a sparse expression for $\overline{W}$.

Since $\overline{W}$ has $m - \alpha$ columns and the flow space of $B(G)$ has dimension $|E| - |V_1 \cup V_2| + c_{B(G)}$, where $c_{B(G)}$ equals the number of connected components in $B(G)$, it follows that:

$$\alpha = |V_1 \cup V_2| - c_{B(G)} = 2n - c_{B(G)}.$$  \hspace{1cm} \hspace{1cm} (19)

Observe that the extreme cases are established by the directed cycle and the complete digraph on $n$ nodes, which yield $\alpha = n$ and $\alpha = 2n - 1$, respectively.

There exist several natural bases for the flow space of a graph, see e.g., [31]. We use the following construction: Let $T$ be a spanning forest of $B(G)$ and let $E(T) \subseteq E$ denote its corresponding edge set. Then, for all $e \in E \setminus E(T)$, we know that $T \cup \{e\}$ contains a cycle. By alternately assigning values +1 and −1 to the edges of the cycle and assigning value 0 to all remaining edges, we obtain a signed characteristic vector of the cycle. By repeating this construction for all edges in $E \setminus E(T)$, we obtain $m - \alpha$ linearly independent vectors in $\text{Nul}([U^\top, V^\top]^\top)$, which form a basis for this subspace. Finding a spanning forest and detecting a cycle in $T \cup \{e\}$ can both be done by a breadth first search.

The pseudo-code for the computation of a sparse $W$ is given in Algorithm 1. This algorithm applies to all QCCP instances.
Algorithm 1 Computation of Transformation Matrix $W$

**Input:** $G = (N, A)$

1. Construct the bipartite representation $B(G) = (V_1 \cup V_2, E)$ of $G$.
2. Find a spanning forest $T$ of $B(G)$.
3. for $e \in E \setminus E(T)$ do
   4. Find the unique cycle $C$ in $T \cup \{e\}$.
   5. Alternately assign values $+1$ and $-1$ to edges on $C$.
   6. Construct vector $w_e \in \mathbb{R}^m$ by $w_e[f] = \begin{cases} \pm 1 & \text{if } f \in C \text{ (according to step 5)}, \\ 0 & \text{otherwise}. \end{cases}$
4. end for
5. Find a cycle cover $\bar{x} \in P$.
6. Let $W \in \mathbb{R}^{(m+1) \times (m+1-\alpha)}$ be the matrix whose columns are $(\begin{smallmatrix} 1 \\ \bar{x} \end{smallmatrix}) \cup \left\{ \left( \begin{smallmatrix} 0 \\ w_e \end{smallmatrix} \right) : e \in E \setminus E(T) \right\}$.

**Output:** $W$

**Remark 1.** Although Algorithm 1 uses $B(G)$ to compute $W$, it is possible to perform the same construction using the original graph $G$. This follows from the fact that the CATs of $G$ form the circuits of a matroid $(A, F)$ where $F := \{ F \subseteq A : \text{subgraph } (N(F), F) \text{ does not contain a CAT} \}$, see [6]. Step 2 of Algorithm 1 then reduces to finding a maximal basis of $(A, F)$ using a greedy algorithm, while step 4 boils down to finding the unique CAT in $T \cup \{e\}$ using a breadth first search.

5 A cutting plane augmented Lagrangian approach

It is known that SDP solvers based on interior point methods exhibit problems in terms of both time and memory for solving even medium-size SDPs. Moreover, interior point methods have difficulties with handling additional cutting planes such as nonnegativity constraints and triangle inequalities. Therefore, solving strong SDP models remains a challenging task.

Recently, a promising alternative for solving large-scale SDP relaxations based on alternating direction augmented Lagrangian methods has been investigated, see [13, 48, 56, 58, 51]. There exist several variants of alternating direction augmented Lagrangian method for solving SDPs, see e.g., [43, 55, 34, 35, 45, 40, 39]. Here, we first consider two variants known as the (original) Alternating Direction Method of Multipliers (ADMM) and the Peaceman–Rachford splitting method (PRSM), also called the symmetric ADMM. Then, we present a novel approach that puts these alternating direction augmented Lagrangian methods into a cutting plane framework. In particular, we show how to efficiently combine the PRSM with Dykstra’s projection algorithm [19] within a cutting plane approach.

5.1 The Alternating Direction Method of Multipliers and the Peaceman-Rachford Splitting Method

The ADMM is a first-order method that is introduced in the 1970s to solve large-scale convex optimization problems. Starting from the augmented Lagrangian function, it decomposes the problem into various subproblems that are relatively easy to solve. In [45], the authors use the ADMM to solve an SDP relaxation for the quadratic assignment problem and in [39] the similar approach is used to compute strong SDP bounds for the quadratic shortest path problem. Their approaches allow for inexpensive iterations and cheap ways for obtaining lower and upper bounds. In this section we first show how to exploit the approach from [45, 39] to solve $(SDP_{S2})$ by the ADMM. Then, we present the PRSM for our problem.

Let us first rewrite $(SDP_{S2})$ by introducing the constraint $Y = WZW^\top$. The purpose of adding this equality is to split the remaining set of constraints into the SDP constraint on $Z$ and the linear
The efficiency of the ADMM constraints on $Y$. To deal with the latter type, we introduce the following set:

$$
\mathcal{Y} := \left\{ Y \in S^{m+1} : \begin{array}{l}
Y_{00} = 1, \\
\text{diag}(Y) = Ye_0, \\
Y_{ef} \leq 1 \ \forall e \neq f \\
Y \geq 0, \\
\text{tr}(Y) = n + 1, \\
Y_{ef} = 0 \ \forall (e, f) \in Z
\end{array} \right\},
$$

(20)

where $Z$ is given in \([8]\). Observe that $\mathcal{Y}$ also contains the constraints that are redundant for (SDP$_2$), see Section 3. We do this to fasten the convergence of the ADMM as suggested in \([39]\).

Remark 2. Observe that $\mathcal{Y}$ does not contain the redundant constraint $\langle J, Y \rangle = (n+1)^2$. Namely, our preliminary experiments show that the gain in convergence after adding that constraint is not worth the additional computation effort caused by adding it to $\mathcal{Y}$.

Now, the starting point of the algorithm is the following relaxation:

$$
\min \left\{ \langle \hat{Q}, Y \rangle : \ Y = WZW^T, \ Y \in \mathcal{Y}, \ Z \succeq 0 \right\},
$$

(21)

that is equivalent to (SDP$_2$). We assume that the transformation matrix $W$ is normalized such that $W^TW = I$. Observe that the sparse $W$ resulting from Algorithm \([1]\) does not have orthogonal columns. Therefore, we apply a QR-decomposition on the matrix obtained from Algorithm \([1]\).

Let $S \in S^{m+1}$ denote the Lagrange multiplier for the linear constraint $Y = WZW^T$. We consider the augmented Lagrangian function of (21) w.r.t. this constraint for a fixed penalty parameter $\beta > 0$:

$$
L_\beta(Z,Y,S) := \langle \hat{Q}, Y \rangle + \langle S, Y - WZW^T \rangle + \frac{\beta}{2} \|Y - WZW^T\|^2_F.
$$

The ADMM aims to minimize $L_\beta(Z,Y,S)$ subject to $Y \in \mathcal{Y}$ and $Z \succeq 0$ while iteratively updating $S$. This problem can be decomposed into subproblems, where we only minimize with respect to one of the matrix variables while keeping the other fixed.

Suppose that $(Z^k,Y^k,S^k)$ denotes the $k$-th iterate of the ADMM. Then the new iterate $(Z^{k+1},Y^{k+1},S^{k+1})$ can be obtained by the following updates:

$$
\begin{align*}
(ADMM) \quad Z^{k+1} &:= \arg\min_{Z \succeq 0} L_\beta(Z,Y^k,S^k), \quad (22) \\
Y^{k+1} &:= \arg\min_{Y \in \mathcal{Y}} L_\beta(Z^{k+1},Y,S^k), \quad (23) \\
S^{k+1} &:= S^k + \gamma \cdot \beta \cdot (Y^{k+1} - WZ^{k+1}W^T). \quad (24)
\end{align*}
$$

Here $\gamma \in (0, \frac{1+\sqrt{5}}{2})$ is the stepsize parameter for updating the Lagrange multiplier $S$, see e.g., \([56]\). The efficiency of the ADMM depends on the difficulty of solving the subproblems \((22)\) and \((23)\).

The $Z$-subproblem can be solved as follows, see also \([15, 39]\):

$$
Z^{k+1} = \arg\min_{Z \succeq 0} \left[ \langle \hat{Q}, Y^k \rangle - \frac{1}{2\beta} \|S^k\|^2_F + \frac{\beta}{2} \left\|WZW^T - \left( Y^k + \frac{1}{\beta} S^k \right) \right\|^2_F \right] = \mathcal{P}_{S^{m+1}-0} \left( W^T \left( Y^k + \frac{1}{\beta} S^k \right) W \right),
$$

where $\mathcal{P}_{S^m}(\cdot)$ denotes the orthogonal projection onto the cone of positive semidefinite matrices of order $m$, which can be performed explicitly, see e.g., \([38]\).

The $Y$-subproblem can be rewritten as follows:

$$
Y^{k+1} = \arg\min_{Y \in \mathcal{Y}} \left[ \langle \hat{Q}, WZ^{k+1}W^T \rangle - \frac{\beta}{2} \left\| \frac{\hat{Q} + S^k}{\beta} \right\|^2_F + \frac{\beta}{2} \left\| Y - WZ^{k+1}W^T + \frac{\hat{Q} + S^k}{\beta} \right\|^2_F \right]
$$

14
\[ \begin{align*}
&= \arg \min_{y \in \mathbb{R}^m} \| y - T_{\text{arrow}}(M) \|^2_2 : \quad 1^\top y = n, \quad y \geq 0 \end{align*} \]

where \( P_Y(\cdot) \) denotes the orthogonal projection onto the polyhedral set \( Y \).

We now show how to project a matrix \( M \in S^{m+1} \) onto \( Y \). For that purpose, we define several operators, see Table 2.

Let \( \tilde{M} \) denote the projection of a matrix \( M \) onto \( Y \). The projection can be split into two parts: the projection of the so-called arrow of \( M \), i.e., the zeroth row, zeroth column and diagonal of \( M \), and the projection of the remaining entries. We specify details below.

We clearly have \( M_{00} = 1 \). The remaining entries of the arrow of \( \tilde{M} \) are obtained as the solution to the following minimization problem:

\[ \min_{y \in \mathbb{R}^m} \left\{ \| y - T_{\text{arrow}}(M) \|^2_2 : \quad 1^\top y = n, \quad y \geq 0 \right\} . \]

Observe that the problem above boils down to a projection of a vector onto the simplex \( \Delta(n) \), where \( \Delta(a) := \{ x \in \mathbb{R}^m : \quad 1^\top x = a, \quad x \geq 0 \} \) for all nonnegative \( a \in \mathbb{R} \). The projection onto \( \Delta(a) \), denoted by \( P_{\Delta(a)}(\cdot) \), can be performed explicitly in \( O(m \log m) \), see [36]. The projection of the remaining entries of \( M \) is trivial.

We conclude that the explicit projection of \( M \) onto \( Y \) equals:

\[ P_Y(M) = E_{00} + T_{\text{box}}(T_{\text{inner}}(M)) + T_{\text{arrow}}(3 \cdot P_{\Delta(n)}(T_{\text{arrow}}(M))) , \]

where \( E_{00} = e_0 e_0^\top \in S^{m+1} \).

In the ADMM the Lagrange multiplier is only updated after both primal variables have been updated. We present below the Peaceman–Rachford splitting method (PRSM) or the symmetric ADMM with larger stepsize [35]. This method consists of two dual updates per iteration. Let \((Z^k, Y^k, S^k)\) denote the \( k \)-th iterate of the PRSM. Then the following iterative scheme is applied:

\[
\begin{align*}
(Z^{k+1}) &:= \arg \min_{Z \geq 0} L_\beta(Z, Y^k, S^k), \quad (25) \\
S^{k+\frac{1}{2}} &:= S^k + \gamma_1 \cdot \beta \cdot (Y^k - W Z^{k+1} W^\top), \quad (26) \\
Y^{k+1} &:= \arg \min_{Y \in \mathcal{Y}} L_\beta(Z^{k+1}, Y, S^{k+\frac{1}{2}}), \quad (27) \\
S^{k+1} &:= S^{k+\frac{1}{2}} + \gamma_2 \cdot \beta \cdot (Y^{k+1} - W Z^{k+1} W^\top). \quad (28)
\end{align*}
\]

Here \( \gamma_1 \) and \( \gamma_2 \) are parameters that must be carefully chosen in order to guarantee convergence. The PRSM is known for accelerated speed of convergence in comparison with other ADMM-like algorithms, see [35].

Table 2: Overview of operators and their definitions.

| Operator       | Description                                                                 |
|---------------|----------------------------------------------------------------------------|
| \( T_{\text{arrow}} \) | \( S^{m+1} \rightarrow \mathbb{R}^m \) \quad \( T_{\text{arrow}} \left( \begin{pmatrix} x_0 & x^\top \end{pmatrix} \right) = \frac{1}{2} \left( \text{diag}(X) + 2x \right). \) |
| \( T_{\text{arrow}}^* \) | \( \mathbb{R}^m \rightarrow S^{m+1} \) \quad \( T_{\text{arrow}}^*(x) = \begin{pmatrix} 0 & \frac{1}{2} x^\top \\ \frac{1}{2} x & \text{Diag}(\frac{1}{2} x) \end{pmatrix} \) |
| \( T_{\text{inner}} \) | \( S^{m+1} \rightarrow S^{m+1} \) \quad \( T_{\text{inner}} \left( \begin{pmatrix} x_0 & x^\top \end{pmatrix} \right) = \begin{pmatrix} 0 & \text{Diag}((\frac{1}{2} x)^\top) \end{pmatrix} \quad \text{where } X \in S^m \text{ is s.t. } X_{ef} = 0 \text{ if } (e, f) \in Z \text{ and } X_{ef} = X_{fe} \text{ otherwise.} \) |
| \( T_{\text{box}} \) | \( S^{m+1} \rightarrow S^{m+1} \) \quad \( T_{\text{box}}(X)_{ef} = \min(\max(X_{ef}, 0), 1) \text{ for all } (e, f). \) |

15
5.2 ADMM versus PRSM: Preliminary Results

In Section 5.1 we present two methods for solving $(SDP_{S2})$: the ADMM and the PRSM. Both approaches can be incorporated within the cutting plane augmented Lagrangian method that we present later. We here provide some preliminary experiments to present the behaviour of both methods in terms of convergence.

We consider a test set of 10 Erdős-Rényi instances with $m$ ranging from 250 to 750, see Section 7 for a specification of these instances. For each instance, we use the ADMM and the PRSM to compute $(SDP_{S2})$ under the same parameter settings as will be explained in Section 7. We compute true lower bounds obtained from the methods, see Section 5.5.3, and scale them such that the final bound is indexed to 100. Figure 2a shows these scaled bounds for all instances, while Figure 2b shows their average over all instances with respect to the number of iterations performed.

![Figure 2a: True lower bounds for the ADMM (dashed) and the PRSM (solid) for full test set.](image1)

![Figure 2b: True lower bounds for the ADMM (dashed) and the PRSM (solid) on average.](image2)

Figure 2 shows that although both methods converge, the PRSM in general produces strong lower bounds faster than the ADMM. This is in line with the accelerated numerical performance of the PRSM in contrast to the ADMM presented in [35]. Because we desire a fast convergence when iteratively adding cuts, we incorporate the PRSM in the cutting plane augmented Lagrangian approach introduced in Section 5.5.

5.3 Projection onto a single BQP Cut

The implementation of the ADMM and the PRSM discussed in the previous section can be used to solve $(SDP_{S1})$ and $(SDP_{S2})$. In order to solve $(SDP_{S3})$, the constraints $Y \in C(T)$ are added to
the set of polyhedral constraints, which significantly increases the complexity of the \( Y \)-subproblem \([23]\). To project onto \( \mathcal{Y} \cap \mathcal{C}(\mathcal{T}) \), we use an iterative projection framework, see Section 5.4. In this section, we first show how to project onto the polyhedron induced by a single triangle inequality.

We assume that \( \mathcal{T} \) contains the arc triples \((e,f,g)\) with \( e \neq f, f \neq g, e \neq g \) that correspond to (possibly violated) triangle inequalities, e.g., resulting from a cutting plane framework. For each \((e,f,g) \in \mathcal{T}\) we let \( \mathcal{H}_{efg} \) be the following polyhedron:

\[
\mathcal{H}_{efg} := \{ Y \in S^{m+1} : Y_{ef} + Y_{eg} \leq Y_{ee} + Y_{fg}, \ \text{diag}(Y) = Ye_0 \}.
\]

Let \( \mathcal{P}_{\mathcal{H}_{efg}}(M) \) denote the projection of a matrix \( M \in S^{m+1} \) onto \( \mathcal{H}_{efg} \). This projection can be written explicitly as stated in the following lemma.

**Lemma 3.** Let \( \hat{M} := \mathcal{P}_{\mathcal{H}_{efg}}(M) \) be the projection of a matrix \( M \in S^{m+1} \) onto \( \mathcal{H}_{efg} \). If \( M_{ef} + M_{eg} \leq \frac{M_{ee} + 2M_{0c}}{3} + M_{fg} \), then

\[
\hat{M}_{st} = \begin{cases} \frac{1}{6} M_{ee} + \frac{2}{3} M_{0c} & \text{if } (s,t) \in \{(e,e), (e,0), (e,e)\}, \\ M_{st} & \text{otherwise}. \end{cases}
\]

If \( M_{ef} + M_{eg} > \frac{M_{ee} + 2M_{0c}}{3} + M_{fg} \), then the projection \( \hat{M} \) can be written explicitly as:

\[
\hat{M}_{st} = \begin{cases} \frac{1}{11} M_{ee} + \frac{2}{11} M_{0c} + \frac{3}{11} M_{fg} + \frac{8}{11} M_{ef} - \frac{3}{11} M_{eg} & \text{if } (s,t) \in \{(e,f), (f,e)\}, \\ \frac{1}{11} M_{ee} + \frac{2}{11} M_{0c} + \frac{4}{11} M_{fg} - \frac{3}{11} M_{ef} + \frac{8}{11} M_{eg} & \text{if } (s,t) \in \{(e,g), (g,e)\}, \\ \frac{1}{11} M_{ee} - \frac{2}{11} M_{0c} + \frac{8}{11} M_{fg} + \frac{3}{11} M_{ef} + \frac{8}{11} M_{eg} & \text{if } (s,t) \in \{(f,g), (g,f)\}, \\ \frac{2}{11} M_{ee} + \frac{6}{11} M_{0c} - \frac{2}{11} M_{fg} + \frac{7}{11} M_{ef} + \frac{2}{11} M_{eg} & \text{if } (s,t) \in \{(0,e), (e,0), (e,e)\}, \\ M_{st} & \text{otherwise}. \end{cases}
\]

**Proof.** See Appendix \([\mathcal{A}]\). \(\square\)

### 5.4 Semi-Parallel Dykstra’s projection algorithm

A reasonable argument for the fact that a cutting plane technique in an alternating direction augmented Lagrangian approach has never been considered before, is the increasing complexity of the involved projections. In our case, it requires a projection onto the intersection of \( \mathcal{Y} \), see \([20]\), and a finite collection of polyhedra \( \mathcal{H}_{efg} \). This can be performed in an iterative approach based on Dykstra’s projection algorithm \([19, 12]\). Although there exist some similarities between the ADMM and Dykstra’s algorithm, see \([53]\), we are the first that combine both methods to compute SDP bounds.

Finding the projection onto the intersection of polyhedra or general convex sets is a well-known problem for which multiple algorithms have been proposed. For a detailed background on projection methods, we refer the reader to \([10, 16]\). Bauschke and Koch \([16]\) compare several projection algorithms for problems motivated by road design and conclude that Dykstra’s cyclic algorithm performs best for projections onto the intersection of convex sets. The idea behind Dykstra’s algorithm is to iteratively project a deflected version of the previous iterate onto the individual sets. This method was first proposed by Dykstra \([19]\) for closed convex cones in finite-dimensional Euclidean spaces and later generalized to closed convex sets in Hilbert spaces by Boyle and Dykstra \([12]\).

We are interested in the following best approximation problem:

\[
\min \| \hat{M} - M \|_F^2 \quad \text{s.t.} \quad \hat{M} \in \mathcal{Y}_T := \left( \bigcap_{(e,f,g) \in \mathcal{T}} \mathcal{H}_{efg} \right) \cap \mathcal{Y},
\]

where \( M \) is the matrix that we project onto \( \mathcal{Y}_T \). Observe that \( \mathcal{Y}_T = \mathcal{C}(\mathcal{T}) \cap \mathcal{Y} \).
Dykstra’s algorithm starts by initializing the so-called normal matrices $R^0_{efg} = \mathbf{0}$ and $R^0_{efg} = \mathbf{0}$ for all $(e, f, g) \in \mathcal{T}$. Now, we set $X^0 = M$ and iterate for $k \geq 1$:

$$
\begin{align*}
X^k &:= \mathcal{P}_Y (X^{k-1} + R_{Y}^{k-1}) \\
R_{Y}^{k} &:= X^{k-1} + R_{Y}^{k-1} - X^k \\
L_{efg} &:= X^k + R_{efg}^{k-1} \\
X^k &:= \mathcal{P}_{\mathcal{H}_{efg}} (L_{efg}) \\
R_{efg}^k &:= L_{efg} - X^k
\end{align*}
$$

\hspace{1cm} (\text{CycDyk})

Several authors have shown that the sequence $\{X^k\}_{k \geq 1}$ strongly converges to the solution of the best approximation problem (29), see [12, 33, 26]. Since the polyhedra are considered in a cyclic order, the iterates (CycDyk) are referred to as Dykstra’s cyclic algorithm. Observe that if $\mathcal{T} = \emptyset$, then (CycDyk) boils down to a single projection onto $\mathcal{Y}$.

Instead of projecting on each polyhedron one after another, it is also possible to project on all polyhedra simultaneously. This method is referred to as Dykstra’s parallel algorithm. We refer the interested reader to Appendix B for an implementation and some details of this parallel version. Although the parallel version takes longer to converge in our case, the projections can be done simultaneously, which might be beneficial if used on parallel machines. Preliminary experiments show that in our cutting plane setting the parallel version, not implemented on parallel machines, is not able to improve on the cyclic version.

To make (CycDyk) efficient, we (partly) parallelize the algorithm. Note that a projection onto $\mathcal{H}_{efg}$ only concerns the entries $(e, f), (e, g), (f, g), (e, e)$ and $(0, e)$. Hence, if two projections onto $\mathcal{H}_{e1f1g1}$ and $\mathcal{H}_{e2f2g2}$ take place one after another and $\{e_1, f_1, g_1\} \cap \{e_2, f_2, g_2\} = \emptyset$, they can in fact be performed simultaneously. We partition the triples in $\mathcal{T}$ into $r$ clusters $C_i$, $i = 1, ..., r$, such that $C_1 \cup ... \cup C_r = \mathcal{T}$ and $C_i \cap C_j = \emptyset$ for all $i, j$. By doing so, an iterate of (CycDyk) is performed in $r + 1$ consecutive steps, instead of $|\mathcal{T}| + 1$ consecutive steps. More details about this clustering step are given in Section 5.5.2. This provides a semi-parallel implementation of (CycDyk).

We take the following actions to further accelerate the algorithm:

- All matrices in (CycDyk) are symmetric, hence we save memory by only working with the upper triangular part of the matrices;
- The normal vectors $R_{efg}^k$ for all $(e, f, g) \in \mathcal{T}$ are very sparse, i.e., only the entries $(e, f), (e, g), (f, g), (e, e)$ and $(0, e)$ may be nonzero. Therefore, we work with normal vectors corresponding to the nonzero elements in $R_{efg}^k$ instead of using full $(m+1) \times (m+1)$ matrices. This has the additional advantage that the memory needed does not increase with the size of the instance;
- The projection onto $\mathcal{Y}$ is considerably more costly than the projection onto the triangle inequalities in terms of computation time. Instead of performing all separate projections exactly once and iterate, numerical tests show that the convergence is accelerated if we perform the projection onto $\mathcal{Y}$ only occasionally. That is, after the projection onto $\mathcal{Y}$ we perform the $|\mathcal{T}|$ triangle inequality projections $K$ times in a cyclic order before we again project onto $\mathcal{Y}$.

5.5 A cutting plane augmented Lagrangian method

In this section we combine the PRSM discussed in Sections 5.1 and 5.2 with the projection method discussed in Sections 5.3 and 5.4. This leads to a cutting plane augmented Lagrangian method (CP-ALM). To the best of our knowledge, no such algorithm exists for solving SDP problems.

In the CP-ALM, we iteratively solve $(\text{SDP}_{\mathcal{T}})$ for a set of cuts $\mathcal{T}$ using the PRSM. Each time the PRSM has converged up to some precision, we evaluate the solution for violated cuts and add
the \( \text{numCuts} \) most violated ones to \( T \), where \( \text{numCuts} \) is a predefined parameter, and repeat. An advantage of using the PRSM in a cutting plane approach, as opposed to an interior point method, is that after the addition of new cuts we can start the new PRSM loop from the last obtained triple \( (Z^k, Y^k, S^k) \). In other words, we exploit the use of warm starts, which speeds up the convergence.

The CP-ALM is provided in Algorithm 2. In the sequel, we explain several ingredients of the algorithm in more detail.

**Algorithm 2 CP-ALM**

**Input:** \( \epsilon_{\text{PRSM}}, \epsilon_{\text{stags}}, \epsilon_{\text{proj}}, \text{maxIter}, \text{maxTotalIter}, \text{maxStagIter} \)

1. Compute \( \hat{W} \) by Algorithm 1 and perform a QR-decomposition on \( \hat{W} \) to obtain \( W \).
2. Set \( Y^0 = 0, Z^0 = 0, S^0 = 0, k = 0 \) and \( T = \emptyset \).
3. while stopping criteria not met do
   4. while stopping criteria not met do
      5. Update \( Z^{k+1} := \mathcal{P}_{\mathcal{G}^{k+1}} \left( W^T \left( Y^k + \frac{1}{\beta} S^k \right) W \right) \).
      6. Update \( S^{k+1} := S^k + \gamma_1 \cdot \beta \cdot (Y^k - WZ^{k+1}W^T) \).
      7. Update \( Y^{k+1} := \mathcal{P}_{\mathcal{X}_k} \left( WZ^{k+1}W^T - \frac{k+1}{\beta} S^{k+1} \right) \) by solving (29) using semi-parallel \( \text{CycDyk} \).
      8. Update \( S^{k+1} := S^k + \gamma_2 \cdot \beta \cdot (Y^{k+1} - WZ^{k+1}W^T) \).
      9. \( k \leftarrow k + 1 \).
   10. end while
11. Identify the violated triangle inequalities and add the \( \text{numCuts} \) most violated cuts to \( T \).
12. Cluster the cuts in \( T \) into sets \( C_1, ..., C_r \).
13. end while
14. Compute \( \text{LB}(S^k) \) using the final dual variable \( S^k \).

**Output:** \( \text{LB}(S^k) \) using the final dual variable \( S^k \).

**5.5.1 Stopping criteria**

The inner while-loop of Algorithm 2 constructs a PRSM sequence for a fixed \( T \). Experiments show that the algorithm is stabilized if, as opposed to adding many cuts at once, we add cuts smoothly in order to keep the residuals small. Hence, we want the inner PRSM sequence to converge before adding new cuts to \( T \). We consider three types of stopping criteria for the inner while-loop:

1. Let \( \epsilon_{\text{PRSM}} > 0 \) be a predefined tolerance parameter. The inner while-loop is terminated after iteration \( k \) if
   \[
   \min \left( \| Y^{k+1} - WZ^{k+1}W^T\|_F, \|W^T (Y^{k+1} - Y^k) W\|_F \right) < \epsilon_{\text{PRSM}}
   \]
   The first term on the left hand side measures primal feasibility, while the second term measures dual feasibility.
2. We stop when a fixed number of iterations \( \text{maxIter} \) is reached.
3. We add a stagnation criterion. Let \( \epsilon_{\text{stags}} > 0 \) be a tolerance parameter. We introduce a variable \( \text{stagIter} \) that is increased by one each time we have \( \| (Y^{k+1}, Q) - (Y^k, Q) \|_F < \epsilon_{\text{stags}} \). We stop the inner while-loop whenever \( \text{stagIter} > \text{maxStagIter} \) for some predefined integer \( \text{maxStagIter} \).

The cyclic Dykstra algorithm in line 7 of Algorithm 2 is stopped whenever \( \|X^{k+1} - X^k\|_F < \epsilon_{\text{proj}} \) for some predefined \( \epsilon_{\text{proj}} > 0 \).

Finally, the outer while-loop, i.e., the cutting plane part, is stopped whenever no more violated cuts can be found or after a predefined number of iterations \( \text{maxTotalIter} > \text{maxIter} \) has been reached.

**5.5.2 Clustering**

As explained in Section 5.3, the cyclic Dykstra algorithm can be partially parallelized by partitioning the set \( T \) into \( r \) clusters of non-overlapping cuts. We explain here how this clustering is done.
Let $H = (V, E)$ denote a graph where each node $i \in V$ represents a cut in $T$ and two nodes are connected by an edge whenever the corresponding cuts are overlapping. Clustering $T$ into the smallest number of non-overlapping sets is then equivalent to finding a minimum coloring in $H$. This problem is known to be $NP$-hard. Galinier and Hertz [29] provide an overview of graph coloring heuristics, where it is concluded that the Tabucol algorithm of Hertz and De Werra [37] is overall very successful. We implement here the improved Tabucol algorithm provided in [28].

5.5.3 Lower bound

After each CP-ALM iterate $k$, we obtain a triple $(Z^k, Y^k, S^k)$ which allows us to compute $\langle \hat{Q}, Y^k \rangle$. Although this value converges to the true SDP bound, the convergence is typically not monotonic, which implies that this value does not necessarily provide a true lower bound for the QCCP instance. We can still use the output of the CP-ALM to obtain a true lower bound by a procedure introduced in [45].

Let $W^\top SW := \{S : W^\top SW \preceq 0\}$.

Then, a true lower bound is obtained by solving:

$$\text{LB}(S^k) := \min_{Y \in Y_T} \langle \hat{Q} + P_{W^\top SW}(S^k), Y \rangle, \quad (30)$$

where $P_{W^\top SW}(S^k)$ is the projection of $S^k$ onto the set $W^\top SW$. This projection can be performed efficiently, see [45]. Moreover, note that (30) is a linear programming problem.

6 Upper bounds

The matrices resulting from the CP-ALM can be used to construct upper bounds for the QCCP. In this section we derive several upper bounding approaches, among which a deterministic method, two randomized algorithms and a Q-learning algorithm that is based on reinforcement learning. We are not aware of other SDP-based rounding algorithms that make use of reinforcement learning. We end the section by providing a hybrid approach that combines all aforementioned heuristics.

6.1 Best Euclidean approximation

Let $(Z^{\text{out}}, Y^{\text{out}}, S^{\text{out}})$ be the outcome of the CP-ALM. Throughout the entire section we assume that the CP-ALM is solved up to high precision in order for the utilized results to be valid. Let $x^{\text{out}}$ be the vector consisting of the diagonal elements of $Y^{\text{out}}$ excluding the first entry. As $x^{\text{out}}$ is an approximation of the optimal cycle cover, one can search for the vector $x \in P$ that is closest to $x^{\text{out}}$ in Euclidean norm. This vector can be obtained as follows:

$$x^* := \arg \max \{x^\top x^{\text{out}} : x \in \text{Conv}(P)\}. \quad (31)$$

The corresponding upper bound is $UB_{EB} := (x^*)^\top Q x^*$.

6.2 Randomized Undersampling

Randomized SDP-based heuristics have proven to be successful for various optimization problems, mainly sparked by the seminal work of Goemans and Williamson [32]. A widely used procedure in the design of approximation algorithms is randomized rounding [49], which rounds a relaxed solution to a solution for the original problem that is close to optimal in expectation. We present an SDP-based randomized rounding algorithm that we refer to as randomized undersampling.

Let $x^{\text{out}} \in \mathbb{R}^m$ be as discussed in Section 6.1. Observe that since all entries of $x^{\text{out}}$ are non-negative and $\sum_{e \in \delta^+(i)} x^{\text{out}}_e = 1$, see (2), we can view $\{x_e\}_{e \in \delta^+(i)}$ as a probability distribution on all arcs leaving node $i$. Similarly, $\{x_e\}_{e \in \delta^-(i)}$ represents a probability distribution on the set of arcs entering node $i$. Hence, for each node $i$ we can draw exactly one arc from $\delta^+(i)$ according to the distribution $\{x_e\}_{e \in \delta^+(i)}$. Let $y_i \in \{0, 1\}^m$ denote the characteristic vector of the outcome of these
n trials. We do the same for the incoming arcs, yielding a vector $y_2 \in \{0,1\}^m$. By construction we have $Uy_1 = V y_2 = 1_n$, but not necessarily $V y_1 = U y_2 = 1_n$.

The vector $y = y_1 \circ y_2$ denotes a partial cycle cover that satisfies $U y \leq 1_n$ and $V y \leq 1_n$. Observe that the probability of including arc $e$ in $y$ equals $x_e^2$. To extend $y$ to a feasible cycle cover, we define:

$$N^+ := \{ i \in N : y_e = 0 \ \forall e \in \delta^+(i) \} \quad \text{and} \quad N^- := \{ i \in N : y_e = 0 \ \forall e \in \delta^-(i) \}. \quad (32)$$

We still have to select exactly one arc from $\delta^+(i)$ for all $i \in N^+$ and one arc from $\delta^-(i)$ for all $i \in N^-$ to extend $y$ to a feasible cycle cover. We can do this by solving a modified version of (31). Let $U_{N^+} \in \mathbb{R}^{[N^+] \times m}$ (resp. $V_{N^-} \in \mathbb{R}^{[N^-] \times m}$) denote the submatrix of $U$ (resp. $V$) induced by the rows corresponding to $N^+$ (resp. $N^-$). Let us define the following vector:

$$\bar{x}_e^{\text{out}} := \begin{cases} -\infty & \text{if } e^- \in N \setminus N^+ \text{ or } e^+ \in N \setminus N^+, \\ x_e^{\text{out}} & \text{otherwise}, \end{cases} \quad (33)$$

where some values are set to $-\infty$ in order to avoid in- or outflows larger than one. We now solve

$$z^* := \arg \max_{z \in \mathbb{R}^m} \{ z^T \bar{x}_e^{\text{out}} : U_{N^+} z = 1_{[N^+]}, V_{N^-} z = 1_{[N^-]}, 0_m \leq z \leq 1_m \} \quad (34)$$

A partial solution $y$ can be extended to a feasible cycle cover if and only if the optimal value to (34) is finite. Indeed, in that case we have $y + z^* \in P$, which yields the bound $UB_{US} = (y + z^*)^T Q (y + z^*)$.

We now repeat this procedure and store the smallest obtained bound.

As we select at most $n$ arcs at random and extend the solution to a full cycle cover, we call this method randomized undersampling. The steps of this method are summarized in Algorithm 3.

### Algorithm 3 Randomized Undersampling for the QCCP

**Input:** $G, Q, x^{\text{out}}$

1. Initialize $y_1 = 0_m$ and $y_2 = 0_m$.
2. for $i \in N$ do
3. Draw $f_1$ from $\delta^+(i)$ with respect to $\{x_e^{\text{out}}\}_{e \in \delta^+(i)}$ and $f_2$ from $\delta^-(i)$ with respect to $\{x_e^{\text{out}}\}_{e \in \delta^-(i)}$.
4. Set $y_1(f_1) = 1$ and $y_2(f_2) = 1$.
5. end for
6. $y := y_1 \circ y_2$.
7. Obtain the sets $N^+$ and $N^-$ and the vector $\bar{x}_e^{\text{out}} \in \mathbb{R}^m$ as in (32) and (33), respectively.
8. if problem (34) has a finite objective value then
9. $UB_{US} \leftarrow (y + z^*)^T Q (y + z^*)$ where $z^*$ is computed by (34).
10. else
11. Go back to STEP 2
12. end if

**Output:** $UB_{US}$

### 6.3 Randomized Oversampling

Instead of sampling a partial solution and deterministically extend it to a full cycle cover, we can also randomly add arcs to a subgraph $H$ of $G$ until it contains a cycle cover visiting all nodes. We call this method randomized oversampling.

We initialize $H = (N, \emptyset)$ and iteratively add pairs of successive arcs to $H$. This is done randomly using a probability distribution on the set $\delta^-(i) \times \delta^+(i)$ for all $i \in N$. We use a rank-one approximation of $Y^{\text{out}}$ for the sake of finite convergence, see Lemma [4] below.

The best rank-one approximation of $Y^{\text{out}}$ is given by $\lambda_{\max} w w^T$, where $\lambda_{\max}$ and $w \in \mathbb{R}^{m+1}$ are the corresponding Perron-Frobenius eigenvalue and eigenvector, respectively. Let $w_0$ denote the zeroth entry of $w$ and let $\bar{w} \in \mathbb{R}^m$ be the vector obtained by excluding $w_0$ from $w$. It follows from the Perron-Frobenius theorem that $w$ can be chosen such that it has nonnegative entries. Since the vectors $(-1, u_i^T)$ and $(1, v_i^T)$ are eigenvectors of $Y^{\text{out}}$ associated with the eigenvalue zero, see Lemma [2], it follows that

$$u_i^T \bar{w} = w_0 \land v_i^T \bar{w} = w_0 \quad \forall i \in N.$$
Suppose that \( w_0 = 0 \). Then \( u_i^T \bar{w} = u_i^T \bar{w} = 0 \) for all \( i \in N \), which implies that \( w \) only contains zeros. Since this contradicts with the fact that \( |w| > 0 \), we have \( w_0 > 0 \).

Now, let \( r \in \mathbb{R}^m \) be defined as \( r := \frac{1}{w_0} \bar{w} \). Since \( u_i^T r = u_i^T r = 1 \) for all \( i \in N \) and \( r \geq 0 \), we conclude that \( r \) is contained in the directed 2-factor polytope. Hence, we can view \( \{r_c \cdot r_f\}_{(c,f) \in \delta^-(i) \times \delta^+(i)} \) as a probability distribution on the pairs of successive arcs for all \( i \in N \).

The oversampling algorithm, see Algorithm 4, iteratively draws a pair of successive arcs \((e, f)\) around \( i \in N \) according to the distribution implied by \( r \) and adds this pair to \( H \). We repeat this until \( H \) contains a cycle cover. The best among possibly multiple cycle covers in \( H \) is obtained by solving problem (51) with respect to \( x^{out} \) restricted to the arcs in \( H \).

### Algorithm 4 Randomized Oversampling for the QCCP

**Input:** \( G, Q, Y^{out}, x^{out} \)

1. Obtain Perron-Frobenius eigenpair \((w, \lambda_{\max})\) of \( Y^{out} \) and let \( r = \frac{1}{w_0} \bar{w} \).
2. Let \( H = (N, \emptyset) \) be the empty subgraph of \( G \).
3. while \( H \) contains no directed 2-factor do
   4. for \( i \in N \) do
      5. Select a pair \((e, f)\) according to probability distribution \( \{r_c \cdot r_f\}_{(c,f) \in \delta^-(i) \times \delta^+(i)} \). Set \( H \leftarrow H \cup \{e, f\} \)
   6. end for
   7. end while
8. Solve (51) with respect to \( x^{out} \) restricted to \( H \), and compute the corresponding upper bound \( UB_{OS} \).

**Output:** \( UB_{OS} \)

We can prove the following result with respect to the termination of Algorithm 4.

**Lemma 4.** Algorithm 4 terminates in a finite number of steps with high probability.

**Proof.** See Appendix C

### 6.4 Sequential Q-learning

The final rounding approach we propose is based on a distributed reinforcement learning (RL) technique, namely Q-learning [4]. Q-learning is a branch of machine learning in which artificial agents learn how to take actions in order to maximize an expected total reward. Recently, RL techniques have shown successful in deriving good feasible solutions for combinatorial optimization problems, see e.g., [5]. We propose here an algorithm in which a set of agents learn how to find (near-)optimal cycles in \( G \) by exploiting our SDP relaxation. Our sequential Q-learning algorithm (SQ-algorithm) is inspired by the work of Gambardella and Dorigo [30] and exploits the solution of the CP-ALM within the learning process.

In the sequential Q-learning algorithm we introduce \( n \) agents each having the independent task to construct a set of node-disjoint cycles. This is done iteratively by adding nodes to the agent’s current path until the path contains a directed cycle or no more nodes can be added. For each agent \( k = 1, \ldots, n \), let \( P_k \) denote its current path and let \( c_k \) and \( p_k \) denote the current node and its predecessor on the agent’s search, respectively. Besides, let \( J_k \) be the set of nodes that is not placed on a cycle by agent \( k \). In each iteration, the successor \( s_k \) of \( c_k \) is selected among one of the nodes in \( N^+(c_k) \cap J_k \), where \( N^+(c_k) \) is the set of nodes reachable from \( c_k \) via a single arc, based on a matrix \( SQ \in \mathbb{R}^{m \times m} \). This matrix indicates on position \((c, f)\) how useful it is to traverse an arc \( f \) after an arc \( e \). We select the successor \( s_k \) that leads to a high \( SQ((p_k, c_k), (c_k, s_k))\)-value and add it to \( P_k \). If the addition of \( s_k \) to \( P_k \) does not result in a cycle, we set the current node \( c_k \) to be \( s_k \). If the addition of \( s_k \) does lead to a cycle \( C_k \subseteq P_k \), we memorize this cycle into the agent’s partial solution vector \( y_k \in \{0, 1\}^m \) and set \( c_k \) to one of the nodes not yet on a cycle. An agent’s search terminates whenever no new successor can be found, i.e., \( N^+(c_k) \cap J_k = \emptyset \), or when \( y_k \) is a full cycle cover. If one of these events occurs, we deactivitate the agent. We repeat the steps above for all active agents, until all agents have been deactivated. This results in \( n \) vectors \( y_k \) that represent sets of node-disjoint cycles, not necessarily full cycle covers. At the end of the cycle-building phase, the (partial) solution \( y_k \) that has relative minimum cost is used to update the \( SQ\)-matrix via delayed
reinforcement learning. Now all agents are again activated and a new cycle-building trial starts using the new SQ-matrix until certain stopping criteria are satisfied, e.g., after a fixed number of trials.

To decide which successor $s_k$ to select for a given $p_k, c_k$ and $J_k$, we define a fit function $f$ that depends on the SQ-values and the quadratic costs $Q = (q_{ef})$. The fit of visiting $u \in N^+(c_k) \cap J_k$ after $c_k$ is:

$$f(u | p_k, c_k, J_k) := \begin{cases} \sum_{e \in \delta^+(J_k, c_k)} \left( \sum_{g \in \delta^-(J_k, c_k)} SQ(e, (c_k, u)) \right)^\delta \cdot \left[ \sum_{g \in \delta^-(J_k, c_k)} \frac{1}{q_{(p_k, c_k)}, (u, g)} \right]^\beta & \text{if } p_k = \emptyset, \\ \left[ SQ((p_k, c_k), (c_k, u)) \right]^\delta \cdot \left[ \frac{1}{q_{(p_k, c_k)}, (u, g)} \right]^\beta & \text{otherwise,} \end{cases}$$

where $\delta, \beta > 0$ are parameters which represent the relative importance between the learned SQ-values and the quadratic costs and $\epsilon > 0$ is a small value to deal with quadratic costs that are zero. After computing the fit for all potential successors, we deterministically select the one with the highest fit value or select randomly proportional to their fit values. That is,

$$s_k = \begin{cases} \arg \max_{u \in N^+(c_k) \cap J_k} f(u | p_k, c_k, J_k) & \text{if } q \leq q_0 \\ S & \text{otherwise,} \end{cases}$$

(35)

where $S$ is a random variable over the set $N^+(c_k) \cap J_k$, where each node is chosen with probability proportional to its fit value. The parameter $q_0 \in [0, 1]$ from (35) is the probability of selecting the successor node deterministically.

The SQ-values measure the usefulness of traversing two successive arcs. Recall that $Y_{out}$ is the output of the CP-ALM. As $Y_{out}$ is likely to be larger when two arcs $e$ and $f$ are in an optimal solution, we initialize the SQ-matrix by setting $SQ(e, f) = Y_{out}$ for all pairs of successive arcs $(e, f)$. The SQ-update is based on a mixture between local memory and a reinforcement learning, similar to (30):

$$SQ(e, f) \leftarrow (1 - \alpha)SQ(e, f) + \alpha \left( \Delta SQ(e, f) + \gamma \max_{g \in \delta^+(f \cap J_k)} SQ(f, g) \right),$$

(36)

where $\alpha, \gamma \in (0, 1)$ represent the learning rate and discount factor, respectively. The learning update consists of a discounted reward of the next state and a reinforcement term $\Delta SQ(e, f)$. Similar to the algorithm of (30), we assume that this reinforcement term is zero throughout the cycle-building phase and update it only at the end of a trial. Hence, we only incur a delayed reinforcement term $\Delta SQ(e, f)$. The discounted reward, however, is incorporated during the cycle-building phase.

The delayed reinforcement of a pair of successive arcs $(e, f)$ can be seen as a reward for cost minimal cycles that is obtained at the end of each trial. After all agents are deactivated, each vector $y_k$ is the characteristic vector of a set of node-disjoint cycles. For each agent $k$ that constructed at least one cycle, we compute $L_k := (y_k^T Q y_k) / (1^T y_k)$, i.e., the relative cost per arc in $y_k$. Let $k_{best}$ denote the agent that constructed the solution with the smallest value $L_k$, and let $L_{best}$ denote its relative cost per arc. Then $\Delta SQ(e, f)$ is computed as:

$$\Delta SQ(e, f) = \begin{cases} \Omega \frac{\Omega_{min}}{\Omega_{best}} & \text{if } (e, f) \text{ is a pair of successive arcs in } y_{k_{best}}, \\ 0 & \text{otherwise}, \end{cases}$$

(37)

where $\Omega$ is a constant.

We let the SQ-algorithm run until some fixed number of trials has passed. All cycles that have been constructed throughout the entire algorithm are stored in memory. Let $\Gamma$ denote the number of distinct cycles that are constructed and define the matrix $B \in \mathbb{R}^{n \times \Gamma}$ as follows:

$$B_{i,k} = \begin{cases} 1 & \text{if node } i \text{ is on cycle } k, \\ 0 & \text{otherwise.} \end{cases}$$

23
Let $b \in \mathbb{R}^\Gamma$ be the vector containing the quadratic cost of each cycle. Then the best upper bound based on our SQ-algorithm is obtained by solving the following set partitioning problem (SPP):

$$\min \left\{ b^T x : \quad B x = 1_n, \ x \in \{0,1\}^\Gamma \right\}. \quad (38)$$

As the SPP is \( \mathcal{NP} \)-hard, computing an optimal solution to (38) might be too much to ask for. Instead, an approximate solution to (38) can be obtained efficiently, e.g., by using the Lagrangian heuristic of Atamtürk et al. [5] which is able to compute near-optimal or even optimal solutions to (38) most of the time. For moderate values of $\Gamma$ and $n$, however, current ILP solvers are able to solve (38) to optimality in a very short time.

A pseudocode of the SQ-algorithm is provided in Algorithm 5.

Algorithm 5 Sequential Q-learning for QCCP

Input: \( G, Q, Y_{\text{out}} \)

1: For all pairs of successive arcs \((e,f)\), initialize \( SQ(e,f) = Y_{\text{out}}^{ef} \).
2: For all agents \( k = 1, ..., n \), initialize the starting node \( c_k = k \in N \), starting edge \( e_k = \emptyset \) and \( J_k = N \). Set \( P_k = \emptyset \) and \( y_k = 0_m \) and activate all agents.
3: while there is at least one active agent do
   4: for all active agents \( k \) do
      5: Update the fit function \( f(u|p_k,c_k,J_k) \) for all \( u \in N^+(c_k) \cap J_k \) and compute \( s_k \) according to (35).
      6: Add \( s_k \) to \( P_k \).
   7: end for
   8: for all active agents \( k \) with \( p_k \neq \emptyset \) do
      9: \( SQ((p_k,c_k),(c_k,s_k)) \leftarrow (1 - \alpha)SQ((p_k,c_k),(c_k,s_k)) + \alpha \gamma \max_{e \in \delta^+(c_k,J_k)} SQ((c_k,s_k),e) \).
   10: end for
   11: for all active agents \( k \) do
      12: if \( P_k \) contains a cycle \( C_k \) then
         13: Set \( J_k \leftarrow J_k \setminus C_k \) and \( P_k \leftarrow \emptyset \).
         14: Set \( (y_k)_e = 1 \) for all arcs \( e \) in \( C_k \).
         15: if \( J_k = \emptyset \) then
            16: Deactivate agent \( k \).
         17: else
            18: Set \( p_k \leftarrow \emptyset \) and choose \( c_k \) uniformly at random out of \( J_k \).
         19: end if
      20: else
         21: Set \( p_k \leftarrow c_k \) and \( c_k \leftarrow s_k \).
         22: if \( N^+(c_k) \cap J_k = \emptyset \) then Deactivate agent \( k \) end if
      23: end if
   24: end for
   25: end while
26: for all pairs of successive arcs \((e,f)\) do
   27: Compute the delayed reinforcement \( \Delta SQ(e,f) \) according to (37).
   28: \( SQ(e,f) \leftarrow (1 - \alpha)SQ(e,f) + \alpha \Delta SQ(e,f) \).
   29: end for
30: If stopping criteria are met, obtain \( UB_{SQ} \) using (38). Otherwise, go to Step 2.

Output: \( UB_{SQ} \)

6.5 Hybrid upper bounding algorithm

The design of the SQ-algorithm discussed in the previous section gives rise to a straightforward hybrid implementation of all above-mentioned upper bounding approaches. Indeed, by adding all cycles that have been created by the best Euclidean approximation, the undersampling and the oversampling algorithm to the matrix \( B \) and solve or approximate the corresponding SPP, a hybrid upper bound \( UB_{HY} \) is obtained which provably outperforms any independent implementation of the mentioned upper bounds.
7 Computational Results

We now test the introduced SDP-based lower and upper bounds on several sets of instances and compare them to various bounds from the literature.

This section is organized as follows: we start by introducing the test sets and the parameter settings that we consider. After that, the performance of the lower and upper bounds are discussed in Section 7.2 and 7.3 respectively.

7.1 Design of numerical experiments

The SDP bounds that we take into account are \((SDP_{S2})\) and \((SDP_{S3})\), which we obtain via the PRSM and the CP-ALM, respectively. The CP-ALM is implemented as presented in Algorithm 2 i.e., using the PRSM and Dykstra’s semi-parallel projection algorithm in the subproblem.

We present results for different number of added cuts. To compare our SDP bounds, we use the first level RLT bound \((RLT_1)\), see Adams and Sherali [1, 2], the MILP-based bound \((MILP)\) and the linearization based bound \((LBB_1)\) from [43]. This latter bound is currently the best bound from the literature when taking both quality and efficiency into account. Since upper bounds for the QCCP are never considered before, we present and compare the upper bounds introduced in Section 6.

All lower and upper bounds are implemented in Matlab on a pc with an Intel(R) Core(TM) i7-8700 CPU, 3.20 GHz and 8 GB RAM. The linear programming problems appearing in our approaches and in the computation of MILP, LBB1 and RLT1 are solved using CPLEX 12.7.1.

We test our bounds on three sets of instances:

- **Reload instances**: The reload instances are the same as the ones used in Rostami et al. [50] for the QTSP and are based on a similar setting from Fischer et al. [24]. The underlying graph is the complete directed graph on \(n\) nodes. The quadratic costs are based on the reload model [57], where each arc is randomly assigned a color from a color set \(L\). The quadratic costs between two successive arcs with the same color is zero. If successive arcs \(e\) and \(f\) are assigned distinct colors \(s\) and \(t\), respectively, the costs equal \(r(s,t)\), where \(r : L \times L \rightarrow \{1, ..., D\}\) is a reload cost function. The function \(r\) is constructed uniformly at random. We consider 60 instances with \(n \in \{10, 15, 20\}\), \(D \in \{1, 10\}\) and \(|L| = 20\). Since preliminary experiments show that the addition of cuts do not significantly improve the bounds, we only compute \((SDP_{S2})\) for these instances.

- **Erdős-Rényi instances**: These instances are based on the \(G(n, p)\) model by Erdős and Rényi [21]. A graph is constructed by fixing \(n\) nodes and including each arc independently with probability \(p\). We present two types of cost structures on these instances:
  - **Uniform Erdős-Rényi instances**: the quadratic cost between any pair of successive arcs is chosen discrete uniformly at random from \(\{0, ..., 100\}\);
  - **Reload Erdős-Rényi instances**: the quadratic cost between any pair of successive arcs is based on a reload cost model using 20 colors and reload costs drawn uniformly from \(\{1, ..., 100\}\).

  We consider 15 instances of each type for \(n\) between 20 and 80 and \(p\) between 0.3 and 0.5.

- **Manhattan instances**: Comellas et al. [18] introduced multidimensional directed grid instances that resemble the street pattern of cities like New York and Barcelona. Given a set of positive integers \(\{n_1, ..., n_k\}\), the Manhattan instances are constructed as explained in [43]. The quadratic costs between any pair of successive arcs is chosen discrete uniformly at random out of \(\{0, ..., 10\}\). We consider a set of 32 Manhattan instances ranging from type \((5, 5)\) to type \((9, 10, 10)\).

The interested reader can download our Erdős-Rényi and Manhattan instances\(^2\)

\(^2\)Instances can be found at www.frankdemeijer.org
Numerical results show that Erdős-Rényi instances and reload instances up to approximately 400 arcs can be solved to optimality within one hour, respectively. The computation limit for Manhattan instances is around 2000 arcs, due to the small density of these graphs. As all costs are integer, we round up all bounds.

For the computation of the bounds we need to specify various parameters. The PRSM is implemented using \( \beta = \lceil m/n \rceil \), \( \gamma_1 = 0.9 \) and \( \gamma_2 = 1.09 \), as preliminary experiments show that this setting gives the most stable performance. The CP-ALM uses the same PRSM parameters in the subproblem, where \( K = 5 \) is used in the semi-parallel implementation of (CycDyk). The stopping criteria of the PRSM and the CP-ALM are as explained in Section 5.5, where we use \( \varepsilon_{stag} = 10^{-5} \) and \( \varepsilon_{proj} = 10^{-8} \). The parameter \( \varepsilon_{PRSM} \) is initially set to \( 10^{-6} \), but after the addition of cuts increased to \( 10^{-4} \), since solving the \( Y \)-subproblem using Dykstra’s algorithm is significantly slower than the initial \( Y \)-subproblem without cuts. Hence, we allow for a lower precision. For the same reason, the maximum number of iterations of the inner while-loop (i.e., \( maxIter \)) of the CP-ALM is initialized to some value and decreased when the first cuts are added, after which we do not change it anymore. For the Erdős-Rényi and Manhattan instances, we initialize \( maxIter \) to 1000 and 1500, respectively, and decrease it to 500 after the addition of cuts. The initial iteration limit for the Manhattan instances is larger, as the CP-ALM needs more iterations to converge for these type of instances.

It turns out that the number of cuts added per main loop, i.e., the value of \( numCuts \), is of major importance for the quality of the final bound. To demonstrate this behaviour, the true lower bounds against the iteration number for a moderate-size Erdős-Rényi instance (ER_4 with \( n = 35 \) and \( m = 361 \)) is plotted in Figure 3 for various values of \( numCuts \) using an iteration limit of 2500. The base line shows the behaviour of the PRSM, i.e., the CP-ALM without the addition of cuts. It is clear that the addition of cuts after 1000 iterations immediately starts improving the bounds. Moreover, as one might expect, the addition of more cuts leads to a higher lower bound, although the largest improvement is due to the addition of the first few cuts. As the addition of more cuts also leads to higher computation times, a trade-off between quality and time has to be made. Based on preliminary experiments, we report results for \( numCuts = 50, 150, 300 \) and 500 for the Erdős-Rényi instances. For the Manhattan instances, we only show results for \( numCuts = 300 \) and 500, as the addition of a small number of cuts does not significantly improve the bounds.

Finally, we need to specify the maximum number of iterations. For the reload instances we set this value to 2500 iterations, although the algorithm in most cases terminates earlier for these instances due to the other stopping criteria. The value of \( maxTotalIter \) for the other two instance types is based on preliminary tests. Similar to the PRSM and the ADMM, the CP-ALM can
suffer from tailing off. Since the addition of more cuts makes later iterations more expensive, one has to decide carefully when to stop. This threshold mainly depends on the value of \( m \). Figure 4 shows the behaviour of the lower bounds averaged over \( \text{numCuts} = 150, 300 \) and 500 on three instances: a small, a moderate-size and a large instance. We normalize the bounds in order to make them comparable, i.e., the plots show the fraction of the final lower bound that is obtained after each iteration. Although at first sight there seems not much difference, one can see from the zoomed image on the right-hand side that the CP-ALM converges relatively faster for smaller instances.

Figure 4: Normalized lower bounds (averaged over different number of cuts) for three instances with different numbers of arcs. Right figure shows zoomed plot including a threshold at 0.996.

Based on these preliminary results, the parameter \( \text{maxTotalIter} \) is set to 2500, 3000 or 3500 if \( m < 500, 500 \leq m < 1000 \) and \( m \geq 1000 \), respectively, for the Erdős-Rényi instances. For the Manhattan instances these values are 3000, 3500 and 4000, respectively, using the same distinction on \( m \).

For the computation of upper bounds, we compute the randomized undersampling and oversampling bounds 500 times for each instance and return the best value. For the SQ-algorithm, we use different parameter settings for each instance type based on preliminary tests. It turns out that the algorithm performs best if the value of \( \delta \) is significantly larger than the value of \( \beta \), i.e., we put more emphasis on the SDP-based SQ-values than on the original quadratic costs. This difference seems more beneficial for larger \( n \), since more agents provide more reliable information on useful cycles. Hence, we use \((\delta, \beta) = (20, 1)\) for the Erdős-Rényi and Manhattan instances, while we use \((\delta, \beta) = (5, 1)\) for the reload instances. Furthermore, we use \( q_0 = 0.4, \gamma = 0.6 \) and \( \Omega = 3(m/n) \) for all instance types. Finally, as it is not clear from our tests which value of \( \alpha \) provides the best results, we run the SQ-algorithm three times using \( \alpha = 0.3, 0.5 \) and 0.7 and solve the final SPP, see (38), using all generated cycles. The number of iterations of the SQ-algorithm is set to 500 for the Erdős-Rényi and the reload instances, while it is set to 100 for the Manhattan instances, due to the large number of nodes. As the final SPPs can be solved efficiently by CPLEX for all our instances, we report the optimal SPP bounds.

### 7.2 Results on Lower Bounds

We now discuss our findings with respect to the lower bounds on all test instances. For the reload instances we compare the performance of \((\text{SDP}_{S2})\) to the performance of \((\text{MILP}), (\text{LBB}1)\) and \((\text{RLT}1)\). We omit the brackets from now on to indicate the bound values. Table 3 shows for each of the 60 reload instances the bound value resulting from each of the approaches. Table 4 shows all computation times for the reload instances, including the number of iterations and the average of the primal and dual residual, see Section 5.5.1, for the PRSM. To visualize the quality of the bounds over the entire reload test set, Figure 5 shows a boxplot of the test data in Table 3. On the \( y \)-axis the deviation from the average bound is presented, i.e., for each instance we compute the ratio of each single bound over the average value of the four bounds and these ratios are visualized per bound type.
Figure 5: Boxplot showing the quality of lower bounds MILP, LBB1, RLT1 and SDP$S_2$ on reload instances.

| Instance | n | D | MILP | LBB1 | RLT1 | SDP$S_2$ |
|----------|---|---|------|------|------|---------|
| REL1     | 10 | 1 | 3    | 4    | 4    | 4       |
| REL2     | 10 | 3 | 9    | 9    | 9    | 9       |
| REL3     | 1   | 3 | 4    | 5    | 5    | 5       |
| REL4     | 10 | 3 | 8    | 9    | 12   |          |
| REL5     | 1   | 3 | 4    | 4    | 4    | 4       |
| REL6     | 10 | 5 | 12   | 13   | 14   |          |
| REL7     | 1   | 3 | 4    | 5    | 5    |          |
| REL8     | 10 | 4 | 9    | 11   | 11   |          |
| REL9     | 1   | 2 | 2    | 2    | 2    |          |
| REL10    | 10 | 4 | 9    | 11   | 12   |          |
| REL11    | 1   | 2 | 3    | 3    | 3    |          |
| REL12    | 10 | 5 | 9    | 9    | 9    | 9       |
| REL13    | 1   | 2 | 4    | 4    | 4    | 4       |
| REL14    | 10 | 3 | 9    | 11   | 11   |          |
| REL15    | 1   | 3 | 4    | 4    | 4    | 4       |
| REL16    | 10 | 3 | 8    | 9    | 11   |          |
| REL17    | 1   | 4 | 4    | 4    | 4    | 4       |
| REL18    | 10 | 3 | 8    | 9    | 10   |          |
| REL19    | 1   | 3 | 5    | 5    | 5    | 5       |
| REL20    | 10 | 3 | 10   | 11   | 11   |          |
| REL21    | 15 | 1 | 2    | 4    | 4    | 5       |
| REL22    | 10 | 2 | 9    | 9    | 12   |          |
| REL23    | 1   | 1 | 3    | 3    | 3    | 4       |
| REL24    | 10 | 1 | 7    | 8    | 11   |          |
| REL25    | 1   | 1 | 4    | 5    | 5    | 5       |
| REL26    | 10 | 1 | 6    | 6    | 9    |          |
| REL27    | 1   | 1 | 4    | 4    | 4    | 4       |
| REL28    | 10 | 1 | 7    | 7    | 9    |          |
| REL29    | 1   | 1 | 5    | 5    | 6    |          |
| REL30    | 10 | 0 | 6    | 7    | 10   |          |
| REL31    | 15 | 1 | 1    | 4    | 4    | 5       |
| REL32    | 10 | 1 | 7    | 8    | 11   |          |
| REL33    | 1   | 1 | 4    | 4    | 4    | 4       |
| REL34    | 10 | 1 | 5    | 5    | 8    |          |
| REL35    | 1   | 1 | 4    | 4    | 4    | 4       |
| REL36    | 10 | 1 | 4    | 5    | 8    |          |
| REL37    | 1   | 2 | 6    | 6    | 6    | 6       |
| REL38    | 10 | 1 | 9    | 9    | 11   |          |
| REL39    | 1   | 1 | 3    | 4    | 3    |          |
| REL40    | 10 | 1 | 6    | 7    | 7    |          |
| REL41    | 20 | 1 | 0    | 3    | 3    | 4       |
| REL42    | 10 | 1 | 4    | 4    | 7    |          |
| REL43    | 1   | 0 | 2    | 2    | 3    |          |
| REL44    | 10 | 0 | 5    | 5    | 7    |          |
| REL45    | 1   | 0 | 2    | 2    | 3    |          |
| REL46    | 10 | 0 | 5    | 5    | 6    |          |
| REL47    | 1   | 0 | 2    | 2    | 3    |          |
| REL48    | 10 | 0 | 3    | 3    | 5    |          |
| REL49    | 1   | 0 | 3    | 3    | 4    |          |
| REL50    | 10 | 0 | 5    | 6    | 8    |          |
| REL51    | 1   | 0 | 3    | 3    | 3    |          |
| REL52    | 10 | 0 | 3    | 4    | 6    |          |
| REL53    | 1   | 0 | 3    | 3    | 4    |          |
| REL54    | 10 | 0 | 6    | 6    | 9    |          |
| REL55    | 1   | 0 | 2    | 2    | 3    |          |
| REL56    | 10 | 1 | 6    | 6    | 8    |          |
| REL57    | 1   | 0 | 3    | 3    | 4    |          |
| REL58    | 10 | 0 | 3    | 4    | 7    |          |
| REL59    | 1   | 0 | 2    | 2    | 3    |          |
| REL60    | 10 | 0 | 5    | 5    | 8    |          |

Table 3: Comparison of different bounds for reload instances.

It follows from Table 3 and Figure 5 that SDP$S_2$ clearly provides the strongest bounds, followed by RLT1, LBB1 and finally by MILP, which behaves poorly for most of the instances. In fact, the hierarchy $LBB1 \leq RLT1 \leq SDP_S2$ can be proven easily and holds with strict inequality for the majority of the instances. It can be seen that SDP$S_2$ performs generally about 1.5 times better than the average of the four bounds. The bound SDP$S_2$ even turns out to be optimal for 88% of the instances. When considering Table 4, it follows that although the computation times are larger than those of MILP and LBB1, the SDP-bound can be computed efficiently for most of the instances. The computation times are always within 30 seconds and for 75% of the instances within 10 seconds, while the computation time of RLT1 is above 90 seconds for 67% of the instances.
Moreover, although the optimum can be computed for all tested reload instances, the computation time is in some cases as large as 2000 seconds. Hence, for the reload instances we conclude that SDP\textsubscript{S2} can be favoured above other bounds in both quality and time.

Next, we consider the Erdős-Rényi instances. Table 5 shows the bound values for the Erdős-Rényi instances. For most instances, this improvement of SDP\textsubscript{S1} optimal, we see that CP-ALM bounds by adding cuts using the new LBB otherwise. The computation times are reported in Table 6, alongside the average of primal and dual residual and the number of final cuts in the CP-ALM.

For the Erdős-Rényi instances we also see that SDP\textsubscript{S2} significantly outperforms MILP and LBB1 in terms of quality of the bound. Moreover, it is clear that we can successfully improve the bounds by adding cuts using the new CP-ALM. Except for the instances where SDP\textsubscript{S2} is already optimal, we see that SDP\textsubscript{S3} provides a strictly higher bound already after adding 50 cuts at a time. For most instances, this improvement of SDP\textsubscript{S3} compared to SDP\textsubscript{S2} is about 3%-6%. Interestingly, this improvement seems to be independent of the problem size. As we already observed in Figure 3, we see that a higher value of numCuts leads to a higher lower bound. This higher value comes, however, at the cost of computation time as can be seen from Table 6. When taking both quality and efficiency into account, it seems beneficial to add only a small number of cuts, as this often leads to a significant increase of the bound at a relatively low computational cost. For instances up to 1000 arcs the CP-ALM terminates often within 30 minutes, while SDP bounds for instances up to 1850 arcs (!) can be computed within 2 hours. Hence, the CP-ALM is able to provide strong lower bounds for very large-scale SDPs in a reasonable time span, whereas the interior point method of Mosek [44] can solve (SDP\textsubscript{S2}) for instances up to only 300 arcs without running out of memory.

| Instance | MILP time | LBB1 time | RLT1 time | SDP\textsubscript{S2} time | iter | res |
|----------|-----------|-----------|-----------|----------------|------|-----|
| REL1     | 0.112     | 0.007     | 1.004     | 0.278          | 437  | 0.003|
| REL2     | 0.133     | 0.007     | 0.398     | 0.215          | 300  | 0.032|
| REL3     | 0.111     | 0.006     | 0.350     | 0.751          | 1192 | 0.005|
| REL4     | 0.106     | 0.006     | 0.380     | 0.624          | 978  | <0.001|
| REL5     | 0.107     | 0.006     | 0.321     | 0.332          | 529  | <0.001|
| REL6     | 0.106     | 0.006     | 0.358     | 0.230          | 377  | 0.045|
| REL7     | 0.108     | 0.006     | 0.364     | 0.665          | 1044 | 0.005|
| REL8     | 0.107     | 0.006     | 0.366     | 0.182          | 284  | 0.086|
| REL9     | 0.107     | 0.006     | 0.217     | 0.370          | 594  | 0.042|
| REL10    | 0.106     | 0.006     | 0.377     | 0.430          | 681  | 0.053|
| REL11    | 0.108     | 0.006     | 0.320     | 0.358          | 586  | 0.027|
| REL12    | 0.108     | 0.006     | 0.334     | 0.147          | 232  | 0.078|
| REL13    | 0.108     | 0.005     | 0.384     | 0.273          | 441  | 0.004|
| REL14    | 0.106     | 0.006     | 0.380     | 0.287          | 327  | 0.038|
| REL15    | 0.108     | 0.005     | 0.284     | 0.396          | 589  | 0.039|
| REL16    | 0.107     | 0.006     | 0.374     | 0.494          | 790  | 0.048|
| REL17    | 0.109     | 0.005     | 0.218     | 0.425          | 695  | 0.040|
| REL18    | 0.107     | 0.006     | 0.393     | 0.442          | 695  | 0.041|
| REL19    | 0.108     | 0.005     | 0.425     | 0.659          | 1034 | 0.005|
| REL20    | 0.106     | 0.006     | 0.408     | 0.170          | 278  | 0.170|
| REL21    | 0.421     | 0.050     | 9.542     | 3.146          | 1371 | 0.007|
| REL22    | 0.422     | 0.048     | 8.475     | 3.950          | 1684 | <0.001|
| REL23    | 0.415     | 0.044     | 8.617     | 3.066          | 1277 | 0.006|
| REL24    | 0.421     | 0.048     | 8.693     | 2.695          | 1124 | 0.001|
| REL25    | 0.415     | 0.044     | 8.565     | 5.069          | 2149 | 0.006|
| REL26    | 0.413     | 0.046     | 8.697     | 2.701          | 1117 | <0.001|
| REL27    | 0.414     | 0.044     | 9.208     | 6.097          | 2500 | 0.012|
| REL28    | 0.411     | 0.051     | 8.229     | 1.667          | 689  | 0.017|
| REL29    | 0.414     | 0.045     | 8.249     | 1.972          | 818  | 0.006|
| REL30    | 0.421     | 0.050     | 8.676     | 3.979          | 1626 | <0.001|

Table 4: Computation times in seconds, average residuals and number of iterations for reload instances.
Finally, we consider the performance of the lower bounds on the Manhattan instances, which can be

| Instance | $p$ | $n$ | $m$ | $OPT$ | MILP | LBB1 | SDP$_{52}$ |
|----------|-----|-----|-----|-------|------|------|-----------|
| ER1      | 0.3 | 20  | 119 | 319   | 165  | 165  | 159       |
| ER2      | 92  | 194 | 128 | 235   | 235  | 235  | 235       |
| ER3      | 24  | 50  | 120 | 240   | 240  | 240  | 240       |
| ER4      | 24  | 50  | 120 | 240   | 240  | 240  | 240       |
| ER5      | 24  | 50  | 120 | 240   | 240  | 240  | 240       |
| ER6      | 24  | 50  | 120 | 240   | 240  | 240  | 240       |
| ER7      | 24  | 50  | 120 | 240   | 240  | 240  | 240       |
| ER8      | 24  | 50  | 120 | 240   | 240  | 240  | 240       |
| ER9      | 24  | 50  | 120 | 240   | 240  | 240  | 240       |
| ER10     | 24  | 50  | 120 | 240   | 240  | 240  | 240       |

Table 5: Comparison of different bounds for Erd˝os-R´enyi instances.

| Instance | time | time | cuts | numCuts = 50 | time | time | cuts | numCuts = 150 |
|----------|------|------|------|---------------|------|------|------|---------------|
| ER1      | 0.204| 0.019| 0.390| <0.001       | 0.333| 0.003| <0.001| 0.395| <0.001       |
| ER2      | 0.319| 0.016| 1.203| <0.001       | 1.790| 0.002| <0.001| 1.710| <0.001       |
| ER3      | 0.327| 0.016| 1.359| <0.001       | 1.967| 0.003| <0.001| 1.970| <0.001       |
| ER4      | 0.673| 0.042| 2.182| <0.001       | 2.955| 0.003| <0.001| 2.674| <0.001       |
| ER5      | 1.151| 0.100| 2.448| <0.001       | 3.973| 0.005| <0.001| 3.503| <0.001       |
| ER6      | 1.166| 0.107| 2.600| <0.001       | 3.997| 0.005| <0.001| 3.470| <0.001       |
| ER7      | 1.594| 0.139| 2.962| <0.001       | 4.801| 0.007| <0.001| 4.053| <0.001       |
| ER8      | 2.088| 0.153| 3.317| <0.001       | 5.153| 0.008| <0.001| 4.244| <0.001       |
| ER9      | 2.856| 0.201| 4.831| <0.001       | 6.034| 0.011| <0.001| 5.194| <0.001       |
| ER10     | 3.048| 0.220| 5.343| <0.001       | 6.727| 0.012| <0.001| 5.725| <0.001       |
| ER11     | 4.489| 0.327| 6.838| <0.001       | 9.078| 0.016| <0.001| 7.700| <0.001       |
| ER12     | 4.207| 0.297| 6.088| <0.001       | 9.652| 0.016| <0.001| 8.578| <0.001       |
| ER13     | 8.823| 0.625| 14.500| <0.001      | 21.000| 0.025| <0.001| 19.200| <0.001      |
| ER14     | 7.969| 0.529| 13.405| <0.001      | 21.600| 0.026| <0.001| 19.900| <0.001      |
| ER15     | 24.18| 1.346| 31.005| <0.001     | 42.250| 0.034| <0.001| 38.650| <0.001      |
| ER16     | 25.23| 1.420| 38.100| <0.001     | 46.250| 0.039| <0.001| 42.500| <0.001      |
| ER17     | 42.03| 2.273| 64.430| <0.001     | 85.830| 0.055| <0.001| 77.200| <0.001      |
| ER18     | 38.46| 2.205| 62.000| <0.001     | 90.650| 0.061| <0.001| 82.200| <0.001      |
| ER19     | 32.79| 1.985| 51.470| <0.001     | 64.850| 0.057| <0.001| 58.000| <0.001      |
| ER20     | 19.74| 2.305| 28.860| <0.001     | 37.500| 0.046| <0.001| 32.200| <0.001      |

Table 6: Computation times (in seconds), average residuals and cuts for Erd˝os-R´enyi instances.

Finally, we consider the performance of the lower bounds on the Manhattan instances, which can be...
found in Table 7 and 8. With respect to the quality of the bounds we can draw the same conclusions as before. Namely, the SDP bound $SDP_{S3}$ performs best on all instances, followed by $SDP_{S2}$. Since the optimal values for many of these instances can be computed, we moreover see that our SDP bounds are very close to optimal. Although we again see that the cuts can successfully improve the lower bounds, the relative improvement is smaller than for the Erdős-Rényi test set. An explanation can be found by looking at the residuals in Table 8, which are significantly larger than the residuals for the first two types of instances. Apparently, the Manhattan instances need more iterations to converge, probably due to the inner structure of these instances. Stopping the CP-ALM when it has only partly converged, leads to weaker and less stable lower bounds. Namely, the true lower bound is obtained by a projection of the current dual matrix, and further experiments show that in particular the dual residual converges slowly. The residuals increase with the size of the instance. Hence, we expect that even better bounds for the Manhattan instances can be obtained by letting the CP-ALM run for more iterations. However, we conclude from the current tables that the SDP bounds for the Manhattan instances significantly outperform the bounds from the literature in a reasonable time span.

| Instance | Type | n | m | OPT | MILP | LBB | $SDP_{S3}$ | numCuts = 300 | numCuts = 500 |
|----------|------|---|---|-----|------|-----|---------|--------------|--------------|
| MH1      | (5, 5) | 25 | 50 | 103 | 102  | 103 | 103     | 103          | 103          |
| MH2      | (10, 10) | 100 | 200 | 418 | 394  | 418 | 418     | 418          | 418          |
| MH3      | (15, 15) | 225 | 350 | 692 | 647  | 692 | 692     | 692          | 692          |
| MH4      | (16, 16) | 256 | 512 | 1030| 985  | 1030| 1030    | 1030         | 1030         |
| MH5      | (17, 17) | 289 | 578 | 1226| 1162 | 1214| 1226    | 1226         | 1226         |
| MH6      | (18, 18) | 324 | 648 | 1283| 1230 | 1282| 1282    | 1282         | 1282         |
| MH7      | (19, 19) | 361 | 722 | 1448| 1378 | 1446| 1446    | 1446         | 1446         |
| MH8      | (20, 20) | 400 | 800 | 1539| 1472 | 1537| 1537    | 1537         | 1537         |
| MH9      | (25, 25) | 625 | 1250| 2572| 2439 | 2569| 2568    | 2568         | 2568         |
| MH10     | (4, 4) | 64 | 192 | 199 | 156  | 199 | 199     | 199          | 199          |
| MH11     | (4, 5) | 80 | 240 | 258 | 203  | 258 | 258     | 258          | 258          |
| MH12     | (4, 5) | 100 | 300 | 343 | 260  | 342 | 342     | 342          | 342          |
| MH13     | (4, 6) | 120 | 360 | 400 | 312  | 398 | 400     | 400          | 400          |
| MH14     | (5, 5) | 125 | 375 | 391 | 304  | 376 | 391     | 391          | 391          |
| MH15     | (5, 6) | 150 | 450 | 528 | 422  | 513 | 528     | 528          | 528          |
| MH16     | (5, 6) | 180 | 540 | 607 | 479  | 586 | 607     | 607          | 607          |
| MH17     | (5, 6) | 210 | 630 | 698 | 539  | 668 | 696     | 697          | 697          |
| MH18     | (6, 6) | 216 | 648 | 700 | 561  | 683 | 697     | 698          | 698          |
| MH19     | (6, 7) | 252 | 756 | 834 | 663  | 808 | 830     | 832          | 832          |
| MH20     | (6, 7) | 294 | 882 | 994 | 779  | 958 | 990     | 992          | 992          |
| MH21     | (6, 8) | 336 | 1008| 1087| 847  | 1047| 1079    | 1083         | 1083         |
| MH22     | (7, 7) | 343 | 1029| 1162| 907  | 1107| 1155    | 1158         | 1158         |
| MH23     | (7, 8) | 392 | 1176| 1246| 975  | 1201| 1238    | 1241         | 1242         |
| MH24     | (7, 8) | 448 | 1344| 1449| 1135 | 1393| 1439    | 1442         | 1442         |
| MH25     | (7, 9) | 504 | 1512| 1645| 1281 | 1576| 1626    | 1631         | 1631         |
| MH26     | (8, 8) | 512 | 1536| 1566| 1247 | 1530| 1555    | 1557         | 1557         |
| MH27     | (8, 9) | 576 | 1728| 1883| 1485 | 1817| 1861    | 1866         | 1867         |
| MH28     | (8, 9) | 648 | 1944| 2075| 1643 | 2003| 2057    | 2060         | 2060         |
| MH29     | (8, 10) | 720 | 2160| 2339| 1850 | 2259| 2309    | 2313         | 2314         |
| MH30     | (9, 9) | 729 | 2187| -   | 1894 | 2329| 2416    | 2421         | 2422         |
| MH31     | (9, 10) | 810 | 2430| -   | 2081 | 2535| 2603    | 2608         | 2608         |
| MH32     | (9, 10) | 900 | 2700| -   | 2304 | 2817| 2886    | 2888         | 2889         |

Table 7: Comparison of different bounds for Manhattan instances.

7.3 Upper bounds

We discuss here the results on the upper bounds. Table 9 shows several statistics related to the performance of the hybrid and non-hybrid upper bounds on the full test set. Table 10 provides an overview of the best lower bound, best upper bound and their relative gap (w.r.t. best lower bound) for the full set of instances. For each instance and upper bound type, we compute the upper bound based on the SDP solution resulting from the CP-ALM, and select the best among all to report in Table 10. Since, by construction, the hybrid algorithm always provides the best among all upper bounds, the last column of Table 10 indicates which of the non-hybrid heuristics performs best when applied independently. Since all upper bounds can be computed relatively fast, we omit computation times here.
It follows from the tables that the bounds perform very well on the Manhattan and the reload instances, as the average gap between the best lower and best upper bound using the hybrid heuristic is 1.25% and 3.90%, respectively. For the Erdős-Rényi instances, this gap is much larger. Namely, it is known that the quality of a lower bound, and thus also of a related upper bound, deteriorate when the size of the problem increases. Moreover, the results in Table 5 and 6 show that it is more difficult to solve the Erdős-Rényi instances than the other two types of instances. Nevertheless, the average gap on the Erdős-Rényi instances with up to 1000 arcs is only 10%.

When comparing the different upper bounds, we conclude that the SQ-algorithm overall outperforms the other methods, followed by oversampling and undersampling rounding. We however observe a clear relationship with the instance type. For the Erdős-Rényi instances the SQ-algorithm is convincingly the best heuristic, while for the reload instances the other methods perform reliable as well, probably due to the smaller instance size. For the Manhattan instances, however, the sequential Q-learning heuristic performs well on the smaller instances, but is outperformed by oversampling rounding for larger $m$. This can be explained by the smaller number of iterations of the SQ-algorithm for these type of instances. Since the number of agents in the SQ-algorithm for the Manhattan instances is significantly larger than for the other instance types, we needed to decrease the number of iterations in order to be able to solve the resulting SPP efficiently. Hence,

Table 8: Computation times in seconds, average residuals and cuts for Manhattan instances.

| Instance | MILP time (s) | LBB1 time (s) | $SDP_{S1}$ time (s) | numCuts = 300 time (s) | numCuts = 500 time (s) |
|----------|--------------|--------------|------------------|------------------|------------------|
| MH1 | 0.103 | 0.016 | 0.047 | 0.030 | 0.002 |
| MH2 | 0.309 | 0.011 | 0.916 | 57.07 | 0.001 |
| MH3 | 3.805 | 0.044 | 0.013 | 0.013 | 0.013 |
| MH4 | 5.406 | 0.057 | 0.017 | 380.02 | 0.011 |
| MH5 | 7.684 | 0.063 | 0.011 | 558.2 | 0.011 |
| MH6 | 10.29 | 0.079 | 0.024 | 336.9 | 0.024 |
| MH7 | 13.44 | 0.118 | 0.024 | 489.5 | 0.023 |
| MH8 | 19.72 | 0.233 | 0.029 | 2226 | 0.035 |
| MH9 | 3.805 | 0.044 | 0.013 | 0.013 | 0.013 |
| MH10 | 5.406 | 0.057 | 0.017 | 380.2 | 0.011 |
| MH11 | 7.684 | 0.063 | 0.011 | 558.2 | 0.011 |
| MH12 | 10.29 | 0.079 | 0.024 | 336.9 | 0.024 |
| MH13 | 13.44 | 0.118 | 0.024 | 489.5 | 0.023 |
| MH14 | 19.72 | 0.233 | 0.029 | 2226 | 0.035 |
| MH15 | 3.805 | 0.044 | 0.013 | 0.013 | 0.013 |
| MH16 | 5.406 | 0.057 | 0.017 | 380.2 | 0.011 |
| MH17 | 7.684 | 0.063 | 0.011 | 558.2 | 0.011 |
| MH18 | 10.29 | 0.079 | 0.024 | 336.9 | 0.024 |
| MH19 | 13.44 | 0.118 | 0.024 | 489.5 | 0.023 |
| MH20 | 19.72 | 0.233 | 0.029 | 2226 | 0.035 |
| MH21 | 3.805 | 0.044 | 0.013 | 0.013 | 0.013 |
| MH22 | 5.406 | 0.057 | 0.017 | 380.2 | 0.011 |
| MH23 | 7.684 | 0.063 | 0.011 | 558.2 | 0.011 |
| MH24 | 10.29 | 0.079 | 0.024 | 336.9 | 0.024 |
| MH25 | 13.44 | 0.118 | 0.024 | 489.5 | 0.023 |
| MH26 | 19.72 | 0.233 | 0.029 | 2226 | 0.035 |
| MH27 | 3.805 | 0.044 | 0.013 | 0.013 | 0.013 |
| MH28 | 5.406 | 0.057 | 0.017 | 380.2 | 0.011 |
| MH29 | 7.684 | 0.063 | 0.011 | 558.2 | 0.011 |
| MH30 | 10.29 | 0.079 | 0.024 | 336.9 | 0.024 |
| MH31 | 13.44 | 0.118 | 0.024 | 489.5 | 0.023 |
| MH32 | 19.72 | 0.233 | 0.029 | 2226 | 0.035 |

Table 9: Statistics on performance of upper bounds on total test set.

| Statistic | Value |
|-----------|-------|
| Average gap on all instances | 20.02% |
| Average gap on Erdős-Rényi instances | 72.30% |
| Average gap on Manhattan instances | 1.25% |
| Average gap on Reload instances | 3.90% |
| Average gap on instances with $m \leq 1000$ | 10.58% |
| Percentage of instances $UB_{LB}$ performs best | 36.89% |
| Percentage of instances $UB_{SQ}$ performs best | 77.87% |
| Percentage of instances $UB_{HY}$ strictly lower than others | 25.41% |
the learning effect of the SQ-algorithm is decreased, while it is in particular that part that makes the algorithm powerful. Nevertheless, we observe for almost all Manhattan instances that the hybrid algorithm obtains a strictly lower upper bound than $UB_{EB}, UB_{US}$ or $UB_{OS}$. This means that the SQ-algorithm, although not always the favoured heuristic when implemented independently, creates cycles that can lead to an improvement of the best upper bound.

| Instance | Best lower bound | Hybrid upper bound | Gap (%) | Best non-hybrid heuristic |
|----------|------------------|--------------------|---------|----------------------------|
| REL1     | 4                | 4                  | 0       | EB, US, OS, SQ             |
| REL2     | 9                | 9                  | 0       | EB, US, OS, SQ             |
| REL3     | 5                | 5                  | 0       | EB, US, OS, SQ             |
| REL4     | 12               | 12                 | 0       | US                         |
| REL5     | 4                | 4                  | 0       | EB, US, OS, SQ             |
| REL6     | 14               | 14                 | 0       | US                         |
| REL7     | 5                | 5                  | 0       | US, OS, SQ                 |
| REL8     | 11               | 11                 | 0       | US, OS, SQ                 |
| REL10    | 2                | 2                  | 0       | US, OS, SQ                 |
| REL11    | 3                | 3                  | 0       | US, OS, SQ                 |
| REL12    | 9                | 9                  | 0       | US, OS, SQ                 |
| REL13    | 4                | 4                  | 0       | US, OS, SQ                 |
| REL14    | 11               | 11                 | 0       | US, OS, SQ                 |
| REL15    | 4                | 4                  | 0       | US, OS, SQ                 |
| REL16    | 11               | 11                 | 0       | US, OS, SQ                 |
| REL17    | 4                | 4                  | 0       | US, OS, SQ                 |
| REL18    | 10               | 10                 | 0       | US, OS, SQ                 |
| REL19    | 5                | 5                  | 0       | EB, US, OS, SQ             |
| REL20    | 11               | 11                 | 0       | US, OS, SQ                 |
| REL21    | 5                | 5                  | 0       | US, OS, SQ                 |
| REL22    | 12               | 12                 | 0       | SQ                         |
| REL23    | 9                | 9                  | 0       | US, OS, SQ                 |
| REL24    | 14               | 14                 | 0       | US, OS, SQ                 |
| REL25    | 9                | 9                  | 0       | US, OS, SQ                 |
| REL26    | 12               | 12                 | 0       | US, OS, SQ                 |
| REL27    | 4                | 4                  | 0       | US, OS, SQ                 |
| REL28    | 9                | 9                  | 0       | US, OS, SQ                 |
| REL29    | 6                | 6                  | 0       | US, OS, SQ                 |
| REL30    | 10               | 10                 | 0       | US, OS, SQ                 |
| REL31    | 5                | 5                  | 0       | US, OS, SQ                 |
| REL32    | 11               | 11                 | 0       | US, OS, SQ                 |
| REL33    | 4                | 4                  | 0       | US, OS, SQ                 |
| REL34    | 8                | 8                  | 0       | US, OS, SQ                 |
| REL35    | 4                | 4                  | 0       | EB, US, OS, SQ             |
| REL36    | 8                | 8                  | 0       | EB, US, OS, SQ             |
| REL37    | 6                | 6                  | 0       | US, OS, SQ                 |
| REL38    | 11               | 11                 | 0       | US, OS, SQ                 |
| REL39    | 3                | 3                  | 0       | EB, US, OS, SQ             |
| REL40    | 7                | 7                  | 0       | EB, US, OS, SQ             |
| REL41    | 4                | 4                  | 0       | EB, US, OS, SQ             |
| REL42    | 7                | 11                 | 57      | SQ                         |
| REL43    | 3                | 3                  | 0       | EB, US, OS, SQ             |
| REL44    | 7                | 10                 | 43      | SQ                         |
| REL45    | 3                | 3                  | 0       | SQ                         |
| REL46    | 6                | 6                  | 0       | EB, US, OS, SQ             |
| REL47    | 3                | 3                  | 0       | SQ                         |
| REL48    | 5                | 5                  | 0       | OS, SQ, SQ                 |
| REL49    | 4                | 4                  | 0       | US, OS, SQ                 |
| REL50    | 8                | 8                  | 0       | US                         |
| REL51    | 3                | 3                  | 0       | SQ                         |
| REL52    | 6                | 9                  | 50      | SQ                         |
| REL53    | 4                | 4                  | 0       | EB, US, OS, SQ             |
| REL54    | 9                | 11                 | 22      | SQ                         |
| REL55    | 3                | 3                  | 0       | EB, US, OS, SQ             |
| REL56    | 8                | 11                 | 38      | SQ                         |
| REL57    | 4                | 4                  | 0       | US, OS, SQ                 |
| REL58    | 7                | 7                  | 0       | US, OS, SQ                 |
| REL59    | 3                | 3                  | 0       | US                         |
| REL60    | 8                | 9                  | 13      | OS, SQ, SQ                 |

Table 10: Overview of best hybrid and non-hybrid upper bounds on all instances.
8 Conclusions

In this paper we first introduce several SDP relaxations with increasing complexity. Our strongest SDP relaxation, \((SDP_{S3})\), contains a large number of constraints which makes it a strong but very difficult to solve relaxation. Therefore we derive a cutting plane augmented Lagrangian method. Our algorithm starts from the Peaceman–Rachford splitting method where the involved polyhedral set is strengthened throughout the algorithm by adding valid cuts. To project onto the polyhedral set, we implement a semi-parallelized version of Dykstra’s cyclic projection algorithm. Parallelization here refers to clustering the set of BQP inequalities into subsets of nonoverlapping cuts. Besides the parallelization step we implement several other efficiency improving steps that contribute to the effectiveness of the CP-ALM. Our algorithm also benefits from warm starts when adding new cuts. The CP-ALM is able to compute lower bounds for large instances up to 2700 arcs, thus having a semidefinite constraint of order 2700, including 7,290,000 nonnegative constrained variables, and up to 2500 BQP cuts within two hours.

We also introduce several upper bounding approaches based on solutions of our SDP relaxations. Among them, we propose an SDP-based distributed reinforcement learning algorithm, which we call sequential Q-learning. Starting from the SDP solution matrix, we let artificial agents learn how to find near-optimal cycles in the graph. We are not aware of other approaches in the literature that combine SDP and reinforcement learning.

We perform extensive numerical experiments. Our numerical results show that both \(SDP_{S2}\) and \(SDP_{S3}\) outperform the current strongest QCCP bounds. The results show that \(SDP_{S3}\) bounds are significantly better than \(SDP_{S2}\) bounds, provided that there exist violated triangle inequalities. Among the upper bounding approaches, our sequential Q-learning algorithm is the winner. The average gap between the best lower and upper bounds on test instances with up to 1000 arcs is about 10%, while for certain instances this average gap can be as low as 1.25%.

Several of the newly introduced approaches can be extended to other problems. The various components of the CP-ALM are rather general, which make it possible to adopt it for solving other SDP models that involve large number of cutting planes, such as for the quadratic traveling salesman problem. Our SDP-based reinforcement learning approach can also be extended for finding feasible solutions of other optimization problems. We expect that the sequential Q-learning approach should perform well for problems on complete graphs. Finally, an interested reader can download our code for computing a basis for the flow space of the bipartite representation of a directed graph.

Acknowledgements. We would like to thank Christoph Helmberg for an insightful discussion about the graph theoretical interpretation of the facial reduction. We would also like to thank Dion Gijswijt for carefully reading the manuscript and his valuable feedback. Moreover, we thank Borzou Rostami for sharing the reload instances with us.

References

[1] W.P. Adams, H.D. Sherali. A tight linearization and an algorithm for zero-one quadratic programming problems. *Management Science*, 32(10):1274–1290, 1986.

[2] W.P. Adams, H.D. Sherali. Linearization strategies for a class of zero-one mixed integer programming problems. *Operations Research*, 38(2):217–226, 1990.

[3] A. Aggarwal, D. Coppersmith, S. Khanna, R. Motwani and B. Schieber. The angular-metric traveling salesman problem. *SIAM Journal on Computing*, 29:697–711, 1999.

[4] E.M. Arkin, M.A. Bender, E.D. Demaine, S.P. Fekete, J.S.B. Mitchell, S. Sethia. Optimal covering tours with turn costs. *SIAM Journal on Computing*, 35(3):531–566, 2005.

[5] A. Atamtürk, G.L. Nemhauser, M.W.P. Savelsbergh. A combined Lagrangian, linear programming and implication heuristic for large-scale set partitioning problems. *Journal of Heuristics*, 1:247–259, 1995.
[6] J. Bang-Jensen, S. Bessy, B. Jackson, M. Kriesell. Antistrong digraphs. *Journal of Combinatorial Theory, Series B*, 122, 2016.

[7] J. Bang-Jensen, G. Gutin. Digraphs: Theory, Algorithms and Applications. *Springer Verlag*, 2009.

[8] T.D. Barrett, W.R. Clements, J.N. Foerster, A.I. Lvovsky. Exploratory combinatorial optimization with reinforcement learning. *arXiv: 1909.04063v2*, 2020.

[9] H.H. Bauschke, J.M. Borwein. Dykstras alternating projection algorithm for two sets. *Journal of Approximation Theory*, 79:418-443, 1994.

[10] H.H. Bauschke, V.R. Koch. Projection methods: Swiss army knives for solving feasibility and best approximation problems with halfspaces. *Contemporary Mathematics*, 636, 1–40, 2015.

[11] J.M. Borwein, H. Wolkowicz. Facial reduction for a cone-convex programming problem. *Journal of Australian Mathematical Society*, 30(3):369-380, 1980.

[12] J.P. Boyle, R.L. Dykstra. A method for finding projections onto the intersection of convex sets in Hilbert spaces. *Advances in Order Restricted Statistical Inference, Lecture Notes in Statistics*, 37, Springer, New York, 1985.

[13] S. Burer, D. Vandenbussche. Solving lift-and-project relaxations of binary integer programs. *SIAM Journal of Optimization*, 16(3):726–750, 2006.

[14] R. Burkard, M. Dell’Amico, S. Martello. Assignment Problems. *Society for Industrial and Applied Mathematics*, Philadelphia, PA, USA, 2009.

[15] Y. Büyükkolak, D. Gözüpek, S. Özkan. On minimum reload cost paths, tours and flows, *Networks*, 74(31): 274–286, 2019.

[16] A. Cegielski. Iterative Methods for Fixed Point Problems in Hilbert Spaces. *Springer*, 2012.

[17] S. Chiba, T. Yamashita. On directed 2-factors in digraphs and 2-factors containing perfect matchings in bipartite graphs. *SIAM Journal of Discrete Mathematics*, 32(1):394-409, 2018.

[18] F. Comellas, C. Dalfó, M.A. Fiol. Multidimensional Manhattan street networks. *SIAM Journal on Discrete Mathematics*, 22(4):1428–1447, 2008.

[19] R.L. Dykstra. An algorithm for restricted least squares regression. *Journal of the American Statistical Association*, 78(384): 837–842, 1983.

[20] D. Drusvyatskiy, H. Wolkowicz. The many faces of degeneracy in conic optimization. *Foundations and Trends in Optimization*, 3(2):77–170, 2017.

[21] P. Erdős, A. Rényi. On random graphs. *Publicationes Mathematicae*, 6(2):290–297, 1959.

[22] S.P. Fekete, D. Krupke. Covering tours and cycle covers with turn costs: hardness and approximation. *arXiv:1808.04417v2*, 2019.

[23] A. Fischer. A polyhedral study of quadratic traveling salesman problems. *Dissertation*, Chemnitz University of Technology, 2013.

[24] A. Fischer, F. Fischer, G. Jäger, J. Keilwagen, P. Molitor, I. Grosse. Exact algorithms and heuristics for the quadratic traveling salesman problem with an application in bioinformatics. *Discrete Applied Mathematics*, 166:87–114, 2014.

[25] F. Fischer, G. Jäger, A. Lau, P. Molitor. Complexity and algorithms for the traveling salesman problem and the assignment problem of second order. *Preprint 2009-16*, Technische Universität Chemnitz, 2009.
[26] N. Gaffke, R. Mathar. A cyclic projection algorithm via duality. *Metrika*, 36:29–54, 1989.
[27] G. Galbiati, S. Gualandi, F. Maffioli. On minimum reload cost cycle cover. *Discrete Applied Mathematics*, 164:112–120, 2014.
[28] P. Galinier, J.K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3:379–397, 1999.
[29] P. Galinier, A. Hertz. A survey of local search methods for graph coloring. *Computers & Operations Research*, 33:2547–2562, 2006.
[30] L.M. Gambardella and M. Dorigo. Ant-Q: a reinforcement learning approach to the traveling salesman problem. *Proceedings of ML-95, 12th international conference on machine learning*, 252–260, 1995.
[31] C. Godsil, G. Royle. Algebraic graph theory. *Springer-Verlag*, 2001.
[32] M.X. Goemans, D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the Association for Computing Machinery*, 42(6):1115–1145, 1995.
[33] S.P. Han. A successive projection method. *Mathematical Programming*, 40:1-14, 1988.
[34] B. He, H. Liu, Z. Wang, X. Yuan. A strictly contractive Peaceman–Rachford splitting method for convex programming. *SIAM Journal on Optimization*, 24(3):1011–1040, 2014.
[35] B. He, F. Ma, X. Yuan. Convergence study of the symmetric version of ADMM with larger step sizes. *SIAM Journal on Imaging Sciences*, 9(3):1467–1501, 2016.
[36] M. Held, P. Wolfe, H.P. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6(1):62–88, 1974.
[37] A. Hertz, D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39:345–351, 1987.
[38] N.J. Higham. Computing a nearest symmetric positive semidefinite matrix. *Linear Algebra and its Applications*, 103:103–118, 1988.
[39] H. Hu, R. Sotirov. On solving the quadratic shortest path problem. *INFORMS Journal on Computing*, 32(2), 219–233, 2020.
[40] H. Hu, R. Sotirov, H. Wolkowicz. Facial reduction for symmetry reduced semidefinite programs. *arXiv:1912.10245v1*, 2019.
[41] A.N. Iusem, A.R. De Pierro. On the convergence of Han’s method for convex programming with quadratic objective. *Mathematical Programming*, 52:265–284, 1991.
[42] G. Jäger and P. Molitor. Algorithms and experimental study for the traveling salesman problem of second order. *Lecture Notes in Computer Science*, 5165:211-224, 2008.
[43] F. de Meijer, R. Sotirov. The quadratic cycle cover problem: special cases and efficient bounds. *Journal of Combinatorial Optimization*, 39:1096-1128, 2020.
[44] MOSEK, Aps. The MOSEK optimization toolbox for MATLAB manual. online at [http://www.mosek.com](http://www.mosek.com), Version 8.0, 2018.
[45] D.E. Oliveira, H. Wolkowicz, Y. Xu. ADMM for the SDP relaxation of the QAP. *Mathematical Programming Computation*, 10:631–658, 2018.
[46] M. Padberg. The boolean quadric polytope: some characteristics, facets and relatives. *Mathematical Programming*, 45:139-172, 1989.
[47] G. Pierra. Decomposition through formalization in a product space. *Mathematical Programming*, 28:96–115, 1984.

[48] J. Povh, F. Rendl, A. Wiegele. A boundary point method to solve semidefinite programs. *Computing*, 78(3):277–286, 2006.

[49] P. Raghavan, C.D. Tompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

[50] B. Rostami, F. Malucelli, P. Belotti, S. Gualandi. Lower bounding procedure for the asymmetric quadratic traveling salesman problem. *European Journal of Operational Research*, 253(3):584–592, 2016.

[51] D. Sun, K.C. Toh, Y. Yuan, and X.Y. Zhao. SDPNAL +: A Matlab software for semidefinite programming with bound constraints (version 1.0). *Optimization Methods and Software*, 35(1):87–115, 2020.

[52] R. Stanček, P. Greistorfer, K. Ladner and U. Pferschy. Geometric and LP-based heuristics for angular travelling salesman problems in the plane. *Computers and Operations Research*, 108:97–111, 2019.

[53] R.J. Tibshirani. Dykstra’s algorithm, ADMM, and coordinate descent: connections, insights, and extensions. *31st Conference on Neural Information Processing Systems*, Long Beach, CA, USA, 2017.

[54] L. Tunçel. On the Slater condition for the SDP relaxations of nonconvex sets. *Operations Research Letters*, 29:181-186, 2001.

[55] C.J.C.H. Watkins. Learning with delayed rewards. *Dissertation*, University of Cambridge, 1989.

[56] Z. Wen, D. Goldfarb, W. Yin. Alternating direction augmented Lagrangian methods for semidefinite programming. *Mathematical Programming Computation*, 2:203–230, 2010.

[57] H. Wirth and J. Steffan. Reload cost problems: minimum diameter spanning tree. *Discrete Applied Mathematics*, 113:73–85, 2001.

[58] X. Zhao, D. Sun, K. Toh. A Newton-CG augmented Lagrangian method for semidefinite programming. *SIAM Journal on Optimization*, 20(4): 1737–1765, 2010.
Appendices

A Proof of Lemma

$\mathcal{P}_{\mathcal{H}_{ef}}(M)$ equals the solution of the following convex optimization problem:

$$\min_{M \in S_{m+1}} \left\{ \|\hat{M} - M\|_F^2 : \hat{M} \in \mathcal{H}_{efg} \right\}.$$ 

Since $\hat{M}_{st} = M_{st}$ for all entries $(s,t)$ that are not involved in the constraints, this optimization problem can be rewritten as:

$$\min_{\delta, \theta, \mu, \pi} 2(\delta - M_{ef})^2 + 2(\theta - M_{eg})^2 + 2(\mu - M_{fg})^2 + (\pi - M_{ee})^2 + 2(\pi - M_{0e})^2$$

s.t.  $\delta + \theta \leq \pi + \mu.$

The explicit expression of $\hat{M}$ follows from the KKT-conditions of the problem above. Let $\lambda \geq 0$ be the Lagrange multiplier of the inequality $\delta + \theta \leq \pi + \mu$. Then, the KKT conditions lead to the following system:

$$\begin{cases}
4(\delta - M_{ef}) + \lambda = 0 \\
4(\theta - M_{eg}) + \lambda = 0 \\
4(\mu - M_{fg}) - \lambda = 0 \\
2(\pi - M_{ee}) + 4(\pi - M_{0e}) - \lambda = 0 \\
\lambda \geq 0 \\
\lambda(\delta + \theta - \mu - \pi) = 0 \\
\delta + \theta \leq \pi + \mu.
\end{cases}$$

Complementarity implies that either $\mu = \delta + \theta - \pi$ or $\lambda = 0$. The latter case leads to the KKT-point $(\delta, \theta, \mu, \pi) = (M_{ef}, M_{eg}, M_{fg}, \frac{M_{ee} + 2M_{0e}}{3})$, which is optimal if and only if $M_{ef} + M_{eg} \leq \frac{M_{ee} + 2M_{0e}}{3} + M_{fg}$. If this inequality does not hold, the substitution $\mu = \delta + \theta - \lambda$ leads to the system

$$\begin{cases}
4(\delta - M_{ef}) + \lambda = 0 \\
4(\theta - M_{eg}) + \lambda = 0 \\
4(\delta + \theta - \pi - M_{fg}) - \lambda = 0 \\
6\pi - 2M_{ee} - 4M_{0e} - \lambda = 0
\end{cases} \quad \Leftrightarrow \quad \begin{cases}
\delta = -\frac{1}{4} \lambda + M_{ef} \\
\theta = -\frac{1}{4} \lambda + M_{eg} \\
\lambda = 6\pi - 2M_{ee} - 4M_{0e}
\end{cases}$$

Substitution into the third equation yields

$$4\pi = 4 \left( -\frac{1}{4} \lambda + M_{ef} - \frac{1}{4} \lambda + M_{eg} - M_{fg} \right) - \lambda \quad \Leftrightarrow \quad 4\pi = -3\lambda + 4M_{ef} + 4M_{eg} - 4M_{fg} \quad \Leftrightarrow \quad 4\pi = -3(6\pi - 2M_{ee} - 4M_{0e}) + 4M_{ef} + 4M_{eg} - 4M_{fg} \quad \Leftrightarrow \quad \pi = \frac{3}{11} M_{ee} + \frac{6}{11} M_{0e} - \frac{2}{11} M_{fg} + \frac{2}{11} M_{ef} + \frac{2}{11} M_{eg}.$$ 

By substitution of this expression into the remaining three equations, we obtain:

$$\lambda = -\frac{4}{11} M_{ee} - \frac{8}{11} M_{0e} - \frac{12}{11} M_{fg} + \frac{12}{11} M_{ef} + \frac{12}{11} M_{eg},$$

$$\delta = \frac{1}{11} M_{ee} + \frac{2}{11} M_{0e} + \frac{3}{11} M_{fg} + \frac{8}{11} M_{ef} - \frac{3}{11} M_{eg},$$

$$\theta = \frac{1}{11} M_{ee} + \frac{2}{11} M_{0e} + \frac{3}{11} M_{fg} - \frac{3}{11} M_{ef} + \frac{8}{11} M_{eg},$$

38
\[ \mu = -\frac{1}{\Pi} M_{ee} - 2 \frac{2}{\Pi} M_{0e} + \frac{8}{\Pi} M_{fg} + \frac{3}{\Pi} M_{ef} + \frac{3}{\Pi} M_{eg}. \]

By setting \( \tilde{M}_{st} = \delta \) for \((s,t) \in \{(e,f),(f,e)\}\), \( \tilde{M}_{st} = \theta \) for \((s,t) \in \{(e,g),(g,e)\}\), \( \tilde{M}_{st} = \mu \) for \((s,t) \in \{(f,g),(g,f)\}\) and \( \tilde{M}_{st} = \pi \) for \((s,t) \in \{(0,e),(e,0),(e,e)\}\), the claim follows.

**B Dykstra’s parallel projection algorithm**

In Section 5.4 Dykstra’s cyclic algorithm is presented to iteratively project onto the polyhedra induced by the BQP cuts. Instead of projecting on each polyhedron one after another, it is also possible to project on all polyhedra simultaneously. This method is referred to as parallel Dykstra. Gaffke and Mathar [20] were the first who proposed this fully simultaneous method. The convergence of this algorithm in Euclidean spaces was shown by Iusem and De Pierro [41] using a construction by Pierra [47]. The approach was later generalized to Hilbert spaces, see e.g., [9].

The idea of the parallel Dykstra algorithm is to project onto each set simultaneously and monitor the sequence of weighted averages of these projections. We present here a tailor-made version of this algorithm in Euclidean spaces was shown by Iusem and De Pierro [41] using a construction by Pierra [47]. The convergence of (ParDyk) in general takes more iterations than the convergence of (CycDyk), where we use the semi-parallel implementation of the latter. This is what one might expect, since in the cyclic version each iterate directly builds on the output of the previous iterates. However, since the projections can be performed simultaneously, the total computation time can still be smaller when implemented on parallel machines. Table 11 shows a comparison of both methods within the CP-ALM on a test set of Erdős-Rényi instances implemented:

\[ \text{Table 11: Comparison of methods in CP-ALM on Erdős-Rényi instances.} \]

Note that the projections in \((\text{ParDyk})\) can be performed simultaneously, as each projection solely uses information resulting from the previous iterate. Under some regularity conditions, the sequence \(\{\tilde{X}^k\}_{k \geq 1}\) in \((\text{ParDyk})\) converges strongly to the solution of the best approximation problem, see [11, 9]. One of the sufficient conditions for convergence is that \(\mathcal{Y}_{\mathcal{T}} \neq \emptyset\), which always holds in our setting.

Based on a construction by Pierra [47], it follows that the algorithm \((\text{ParDyk})\) is equivalent to the cyclic Dykstra algorithm performed to the following two convex sets in the higher dimensional space \((S^{m+1})^{|\mathcal{T}|+1} : S^{m+1} \times \ldots \times S^{m+1} \):

\[ \mathbf{S}_1 := \mathcal{Y} \times \prod_{(e,f,g) \in \mathcal{T}} \mathcal{H}_{e,f,g} \quad \text{and} \quad \mathbf{S}_2 := \left\{ (X,X,\ldots,X) \in (S^{m+1})^{|\mathcal{T}|+1} : X \in S^{m+1} \right\}, \]

using the inner product \(\langle \cdot, \cdot \rangle_\theta\) defined as

\[ \langle (X_0,X_1,\ldots,X_\mathcal{T}), (Y_0,Y_1,\ldots,Y_\mathcal{T}) \rangle_\theta := \theta \langle X_0,Y_0 \rangle + (1-\theta) \frac{1}{|\mathcal{T}|} \sum_{i=1}^{|\mathcal{T}|} \langle X_i,Y_i \rangle. \]
on non-parallel machines. Results are presented for different values of $\theta$. In all cases the true lower bound obtained by the CP-ALM using \(\text{ParDyk}\) in the subproblem at the moment the iteration limit is reached is weaker than the lower bound obtained from using \(\text{CycDyk}\) in the subproblem. Moreover, since the parallel version takes more iterations to converge, the computation times are significantly larger. We conclude that the use of \(\text{CycDyk}\) is favoured above the use of \(\text{ParDyk}\) within the CP-ALM in both quality and computation time. For that reason, we only use \(\text{CycDyk}\) in the numerical experiments of Section 7.

| $p$ | $n$ | $m$ | \(\text{PRSM}\) value | \(\text{PRSM}\) times | \(\text{CP-ALM using cyclic Dykstra}\) value | \(\text{CP-ALM using cyclic Dykstra}\) times | \(\text{CP-ALM using parallel Dykstra}\) value | \(\text{CP-ALM using parallel Dykstra}\) times |
|-----|-----|-----|----------------|----------------|----------------|----------------|----------------|----------------|
| 0.3 | 20  | 119 | 319 0.331     | 319 0.384      | 319 0.415      | 319 0.378      | 319 0.389      |
| 25  | 177 | 386 | 1,822 5.437   | 386 24.35      | 386 24.01      |                |                |
| 30  | 280 | 333 | 20.75 96.27   | 335 70.36      | 333 143.37     |                |                |
| 0.5 | 20  | 195 | 227 10.15     | 234 92.89      | 229 4203       | 227 2923       | 227 733.7      |
| 25  | 327 | 169 | 35.68 173     | 170 6640       | 169 5623       | 169 1852       |
| 30  | 442 | 198 | 91.71 202     | 202 130.8      | 199 14137      | 198 10815      | 198 3677       |

Table 11: Performance of CP-ALM using cyclic and parallel Dykstra on a test set of 6 Erdős-Rényi instances with \(\text{maxIter} = 1000\), \(\text{maxTotalIter} = 5000\), \(\text{numCuts} = 150\) and all other parameters the same as given in Section 7.

In order to reduce the number of iterations to converge, we can perform a preprocessing step before the $Y$-subproblem is solved using \(\text{ParDyk}\). Suppose this subproblem involves the projection of a matrix $M$ onto $\mathcal{Y}_T$. Since this projection is done iteratively, the length of the sequence before convergence depends on the initial distance between $M$ and $\mathcal{Y}_T$. This distance can be shortened using a simple preprocessing step. This step involves the projection onto all affine constraints of $M$. We define:

$$\mathcal{Y}_T := \{Y \in S^{m+1} : Y_{00} = 1, \text{diag}(Y) = Ye_0, \text{tr}(Y) = n + 1, Y_{ef} = 0 \ (\forall (e, f) \in Z)\}.$$  

Since $\mathcal{Y}_T$ is an affine subspace, the projection $P_{\mathcal{Y}_T}(\cdot)$ onto $\mathcal{Y}_T$ can be found explicitly. Now, instead of projecting $M$ onto $\mathcal{Y}_T$, we can equivalently project the ‘closer’ matrix $P_{\mathcal{Y}_T}(M)$ onto $\mathcal{Y}_T$, as shown by the following lemma.

**Lemma 5.** $P_{\mathcal{Y}_T}(M) = P_{\mathcal{Y}_T}(P_{\mathcal{Y}_T}(M))$.

**Proof.** Let $\tilde{M} := P_{\mathcal{Y}_T}(M)$ and $\hat{M} := P_{\mathcal{Y}_T}(M)$. We have to show that $P_{\mathcal{Y}_T}(\tilde{M}) = \hat{M}$. Using the Kolmogorov conditions, the projection of $\tilde{M}$ onto $\mathcal{Y}_T$ is the unique solution s.t.:

$$P_{\mathcal{Y}_T}(\tilde{M}) \in \mathcal{Y}_T \quad \text{and} \quad \langle Y - P_{\mathcal{Y}_T}(\tilde{M}), \hat{M} - P_{\mathcal{Y}_T}(\tilde{M}) \rangle \leq 0 \quad \text{for all } Y \in \mathcal{Y}_T.$$

Clearly, $\hat{M}$ satisfies the first condition. Moreover,

$$\langle Y - \tilde{M}, \hat{M} - \tilde{M} \rangle = \langle Y - \tilde{M}, \hat{M} - \tilde{M} \rangle \leq 0,$$

for all $Y \in \mathcal{Y}_T$. Here $\langle Y - \tilde{M}, \hat{M} - \tilde{M} \rangle \leq 0$ follows from the Kolmogorov conditions for the projection of $M$ onto $\mathcal{Y}_T$ and the equality $\langle Y - \tilde{M}, \hat{M} - \tilde{M} \rangle = 0$ follows from the fact that $Y, \tilde{M} \in \mathcal{Y}_T$ and $\tilde{M}$ is orthogonal to the affine space $\mathcal{Y}_T$. We conclude that $\hat{M} = P_{\mathcal{Y}_T}(\tilde{M})$.  

Observe that the projection onto the unconstrained simplex $\Delta(a) := \{x \in \mathbb{R}^m : 1^\top x = a\}$ is given by $P_{\Delta(a)}(x) = x - \frac{1}{1 + a} 1$. Thus the projection of $M$ onto $\mathcal{Y}_T$ is explicitly given by:

$$P_{\mathcal{Y}_T}(M) = E_{00} + T_{\text{inner}}(M) + T_{\text{arrow}} \left(3 \cdot P_{\Delta(a)}(T_{\text{arrow}}(M))\right).$$

Solving the $Y$-subproblem is now equivalent to perform the projection onto $\mathcal{Y}_T$ once and apply \(\text{ParDyk}\) to project $P_{\mathcal{Y}_T}(M)$ onto $\mathcal{Y}_T$. Further experiments show that this step indeed reduces the number of iterations, but this reduction is not enough to exceed the performance of \(\text{CycDyk}\).
C Proof of Lemma \[4\]

Observe that a pair of successive arcs \((e, f) \in \delta^-(i) \times \delta^+(i)\) can be added to \(H\) either by sampling \(e\) and \(f\) simultaneously in step 5 of Algorithm \[4\] or since both arcs are added to \(H\) in combination with some other arc. In the former case, we say that the pair \((e, f)\) is drawn around \(i\). For all \(i \in N\), \((e, f) \in \delta^-(i) \times \delta^+(i)\) and \(k \geq 0\), let \(Y_{i,(e,f)}^k\) denote the following random variable:

\[
Y_{i,(e,f)}^k := \begin{cases} 
0 & \text{if } (e, f) \text{ is not drawn around } i \text{ during the first } k \text{ iterations}, \\
1 & \text{otherwise}.
\end{cases}
\]

Observe that \(Y_{i,(e,f)}^k\) is independent over \(i\), as step 5 is performed independently over \(N\). Since the probability that a pair \((e, f)\) is added to \(H\) in a single iteration equals \(r_e \cdot r_f\), we have

\[
\Pr (Y_{i,(e,f)}^k = 1) = 1 - \Pr (Y_{i,(e,f)}^k = 0) = 1 - (1 - r_e \cdot r_f)^k.
\]

Since \(r \in \text{Conv}(P)\), there must be at least one cycle cover, say \(\bar{x} \in P\), that has full support in \(r\). We define the functions \(p : N \to A\) and \(s : N \to A\) that map each node \(i\) to its predecessor and successor in \(\bar{x}\), respectively. We show that the probability that the support of \(\bar{x}\) is in \(H\) converges to 1 if \(k\) increases. Since the probability that a pair of successive arcs \((e, f) \in \delta^-(i) \times \delta^+(i)\) is present in \(H\) after \(k\) iterations is at least \(\Pr (Y_{i,(e,f)}^k = 1)\), we have:

\[
\Pr (\text{supp}(\bar{x}) \subseteq H \text{ after } k \text{ iterations}) \geq \Pr \left( \prod_{i \in N} Y_{i,(p(i),s(i))}^k = 1 \right) = \prod_{i \in N} \left( 1 - (1 - r_{p(i)} \cdot r_{s(i)})^k \right).
\]

Since there exists an \(\xi > 0\) such that \(r_{p(i)} \cdot r_{s(i)} > \xi\) for all \(i \in N\), we have

\[
\Pr (\text{supp}(\bar{x}) \subseteq H \text{ after } k \text{ iterations}) \geq (1 - (1 - \xi)^k)^n.
\]

Now for any \(q < 1\), take \(k^* = \left\lceil \frac{\log(1 - q/\xi)}{\log(1 - \xi)} \right\rceil\). Then \(\Pr (\text{supp}(\bar{x}) \subseteq H \text{ after } k^* \text{ iterations}) \geq q\). Thus, the cycle cover \(\bar{x}\) is included in \(H\) after a finite number of iterations with arbitrary high probability. \(\Box\)

41