JOTA manuscript No.
(will be inserted by the editor)

A Tree Decomposition-based Branch-and-Bound Algorithm for the Max-Cut Problem with Chordal Sparsity Patterns

Cheng Lu · Zhibin Deng · Shu-Cherng Fang · Wenxun Xing

Received: date / Accepted: date

Abstract In this paper, we develop a semidefinite relaxation-based branch-and-bound algorithm that exploits the chordal sparsity patterns of the max-cut problem. We first study how the chordal sparsity pattern affects the hardness of a max-cut problem. To do this, we derive a polyhedral relaxation based on the clique decomposition of the chordal sparsity patterns, and prove some sufficient conditions for the tightness of this polyhedral relaxation. The theoretical results show that the max-cut problem is easy to be solved when the sparsity pattern embedded in the problem has a small treewidth and the number of vertices in the intersection of maximal cliques is small. Based on the theoretical results, we propose a new branching rule called hierarchy branching rule, which utilizes the tree decomposition of the sparsity patterns. We also analyze how the proposed branching rule affects the chordal sparsity patterns embedded in the problem, and explain why it can be effective. The numerical experiments show that the proposed algorithm outperforms those known algorithms using

Cheng Lu
School of Economics and Management, North China Electric Power University, Beijing 102206, China
lucheng1983@163.com

Zhibin Deng (corresponding author)
School of Economics and Management, University of Chinese Academy of Sciences; MOE Social Science Laboratory of Digital Economic Forecasts and Policy Simulation at UCAS, Beijing, 100190, China
zhibindeng@ucas.edu.cn

Shu-Cherng Fang
Department of Industrial and System Engineering, North Carolina State University, Raleigh 27695-7906, USA
fang@ncsu.edu

Wenxun Xing
Department of Mathematical Sciences, Tsinghua University, Beijing, 100084, China
wxing@tsinghua.edu.cn

classical branching rules and the state-of-the-art solver BiqCrunch on most instances with sparsity patterns arisen in practical applications.

Keywords Max-Cut · Branch-and-Bound · Sparsity Pattern

Mathematics Subject Classification (2000) 90C20 · 90C27 · 90C57

1 Introduction

In this paper, we consider the max-cut problem, which can be formulated as a binary quadratic programming problem as follows:

$$\begin{aligned} \max \quad & x^T Q x \\ \text{s.t.} \quad & x_i \in \{-1, +1\},\ i = 1, ..., n, \end{aligned} \tag{MC}$$

where $Q \in \mathbb{R}^{n \times n}$ is a given symmetric matrix.

The max-cut problem is a classical optimization problem with many practical applications that has received great attention in the past. Besides, it is known that the linearly constrained quadratic 0-1 programming problem can also be formulated as a max-cut problem [27]. Hence, (MC) is quite general in nature. It is known that (MC) is NP-hard in general [14].

The literature for the max-cut problem consists of a large number of solution methods in two strands. One is to design efficient heuristic algorithms for finding sub-optimal solutions. Goemans and Williamson [16] proposed a semidefinite relaxation based approximation method for the max-cut problem. Burer et al. [7] proposed an alternative rank two relaxation and developed a specialized version of rounding technique, which outperforms the classical rounding algorithm proposed in [16]. Besides of the rounding algorithms, some neighborhood search based heuristic method have also been proposed (for example, [11] and [29]). The readers may refer to [9] for a comprehensive review and systematic comparison of various types of approximation algorithms. However, it is well-known that approximation algorithms cannot guarantee to find global solutions of (MC) in general.

The other strand is to design branch-and-bound algorithms for finding exact solutions of (MC). Depending on the relaxation methods used, we may further classify these algorithms into four types: linear relaxation-based [4, 22], convex quadratic programming relaxation-based [6], second-order cone relaxation-based [23,31], and semidefinite relaxation-based [32,20] branch-and-bound algorithms. Rendl et al. [33] discussed the advantages and limitations of the above four types of algorithms. For the linear relaxation based branch-and-bound algorithms, some max-cut problem arising in statistical physics can be solved for instances with up to $12,100$ variables for Q having ± 1 entries only, and $22,500$ variables for Q having entries following a Gaussian distribution. However, Rendl et al. [33] also showed that the linear relaxation based algorithms may even fail to solve a general (MC) with only 100 variables when the density is over 20%. Since 2010, some semidefinite relaxation based algorithms became capable of solving certain types of sparse

(MC) with 100-200 variables [33,25,26]. The Biq Mac algorithm [33] applies a bundle method to solve the semidefinite relaxations enhanced by triangle inequalities, and achieves a high performance on solving problems with hundreds of variables in different densities. Krislock et al. [25] proposed another semidefinite relaxation based algorithm by using a quasi-Newton algorithm to solve the relaxation problems. The numerical results in [25] showed that their algorithm outperforms Biq Mac on 75% instances for a test set of 328 instances. They further improved the bounding procedure in [26] and published a semidefinite-based solver "BiqCrunch" for binary quadratic problems. As far as the authors know, BiqCrunch has achieved the state-of-the-art efficiency, and the numerical results in [26] showed that BiqCrunch runs faster than Biq Mac on most test instances in a large benchmark test set.

Sparsity is often utilized in designing efficient algorithms to solve (MC) exactly. The existing algorithms generally exploit the edge sparsity of a graph to reduce the number of variables in a linear relaxations or to exploit the cycles in a graph to derive new valid inequalities [4,22]. However, there lacks research on exploiting the "chodal sparsity patterns" of (MC) to design efficient global algorithms. A sparsity pattern can be naturally interpreted as an undirected graph and the chordal sparsity pattern is then described as the clique decomposition of a sparsity pattern. Examples of matrices with chordal sparsity patterns include matrices with banded structure, overlapping block-diagonal structure and block-arrow structure. These matrices are often observed in real-life applications [1,17,35,37]. In recent years, many works started to explore the chordal sparsity pattern for solving semidefinite programming problems and sparse polynomial optimization problems [12,15,39,41,42,40,43,44]. The readers may refer to [38] for a review on the topic of chordal graph and its applications in optimization. However, as far as we know, in the area of global optimization, there is no work to study how to exploit the chordal sparsity pattern to design effective branching rules for a branch-and-bound algorithm.

This paper intends to design an efficient branch-and-bound algorithm for solving the max-cut problem with chordal sparsity patterns. In particular, we design a new branching rule based on the clique decomposition of the sparsity pattern of the max-cut problem to avoid exploring unpromising subproblems in order to achieve high efficiency. The contributions of this paper are two-fold:

1) In theory, in order to analyze how the chordal sparsity pattern affects the tightness of a convex relaxation of a max-cut problem, we derive a polyhedral relaxation and provide sufficient conditions for the tightness of the polyhedral relaxation. The sufficient conditions imply that the convex relaxation of a max-cut problem can be tight if the sparsity pattern of the problem has the following two properties: i) the treewidth is small, and ii) the number of vertices in the intersection of maximal cliques is small.

2) Based on the theoretical results, we derive a new branching rule called "hierarchy branching rule", which aims at reducing the treewidth of the sparsity pattern, and reducing the number of vertices in the intersection of maximal cliques. The intuitive idea behind the proposed method is that the new rule aims to manipulate the sparsity pattern of max-cut problem towards the one

of small treewidth. To the best of our knowledge, the proposed algorithm is the first one exploiting chordal sparsity pattern for designing a branching rule. We test the proposed algorithm on instances of the max-cut problem with five types of sparsity patterns. The numerical results clearly show the benefit of our new branching rule.

The rest of this paper is organized as follows. Section 2 reviews some useful definitions and theories in graph theory. Section 3 designs a polyhedral relaxation of the max-cut problem and studies the tightness of the proposed relaxation. Section 4 proposes a new branching rule and a cutting-plane selection scheme based on the clique decomposition of sparsity pattern embedded in the max-cut problem. Section 5 designs a branch-and-bound algorithm that adopts the new branching rule and cutting-plane selection scheme. Numerical results are presented in Section 6, while Section 7 concludes the paper.

The following notations are adopted in this paper. $\mathbb{S}^n$ is the set of real symmetric matrices of size $n \times n$ and $\mathbb{S}^n_+$ is the set of real positive semidefinite matrices of size $n \times n$. For a matrix $X \in \mathbb{S}^n$ and an index set $C \subseteq \{1, 2, ..., n\}$, $X[C]$ represents the square submatrix of X with the rows and columns indexed by the set C. $|C|$ is the cardinality of set C. $X_{i:}$ denotes the i-th row of X, and $X_{:j}$ denotes the j-th column. $X \succeq 0$ means that the matrix X is positive semidefinite. $Z = \mathcal{X}(P, C)$ represents the mapping that lifts a matrix $P \in \mathbb{S}^{|C|}$ to the matrix $Z \in \mathbb{S}^n$ where $Z[C] = P$ and $Z_{ij} = 0$ if either $i \notin C$ or $j \notin C$.

2 Preliminary results in graph theory

Since the sparsity pattern of a matrix $Q \in \mathbb{S}^n$ can be represented by a graph, we review some related results in graph theory in this section.

An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined by its vertex set $\mathcal{V}$ and edge set $\mathcal{E} = \{[i, j] \,|\, i, j \in \mathcal{V}, i \neq j\}$. For an undirected graph, both the two notations $[i, j]$ and $[j, i]$ represent the same edge. A cycle of length r in a graph is defined by a sequence of vertices $i_1, \ldots, i_r, i_{r+1}$ in $\mathcal{V}$ such that $[i_t, i_{t+1}] \in \mathcal{E}$ for all $t = 1, \ldots, r$, $i_{r+1} = i_1$ and $i_1, \ldots, i_r$ are different vertices in $\mathcal{V}$. A chord in a cycle $i_1, \ldots, i_r, i_{r+1}$ of length $r \geq 4$ is an edge $[i_s, i_t] \in \mathcal{E}$ with the pair of vertices i_s, i_t being two nonconsecutive vertices in the cycle.

To represent the sparsity pattern of a matrix $Q \in \mathbb{S}^n$, we define the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ corresponding to Q, where $\mathcal{V} = \{1, 2, \ldots, n\}$ and $\mathcal{E} = \{[i, j] \,|\, Q_{ij} \neq 0,\ i, j \in \mathcal{V},\ i \neq j\}$. The extended set of edges is then defined as

$$\bar{\mathcal{E}} := \mathcal{E} \bigcup \{[1, 1], [2, 2], \ldots, [n, n]\}.$$

The set of real symmetric matrices having a given sparsity pattern $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is then denoted as

$$\mathbb{S}^n(\mathcal{G}) = \{X \in \mathbb{S}^n \,|\, X_{ij} = X_{ji} = 0 \text{ if } [i, j] \notin \bar{\mathcal{E}}\}.$$

Similarly, the set of real positive semidefinite matrices with a given sparity pattern $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is denoted as

$$\mathbb{S}^n_+(\mathcal{G}) = \{X \in \mathbb{S}^n_+ \,|\, X_{ij} = X_{ji} = 0 \text{ if } [i, j] \notin \bar{\mathcal{E}}\}.$$

Some definitions in graph theory, including clique, tree decomposition, chordal graph and graph contraction, are important for deriving the branch-and-bound algorithm that will be proposed in Section 5. We introduce them in sequence.

Definition 2.1 (Clique, maximal clique and maximum clique) Given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a clique of $\mathcal{G}$ is a subset of vertices $C \subseteq \mathcal{V}$, such that for any two different vertices $i, j \in C$, we have $[i, j] \in \mathcal{E}$. A clique C is called maximal clique if $C \bigcup \{i\}$ is not a clique of $\mathcal{G}$ for any vertex $i \notin C$. A clique C is called maximum clique if $|C|$ achieves the largest value among all different cliques.

Definition 2.2 (Tree decomposition ([8], Section 12.3)) For an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a tree decomposition of $\mathcal{G}$ is defined by a graph $\mathcal{T} = (\mathcal{U}, \mathcal{H})$, where vertex set $\mathcal{U} = \{C_1, \ldots, C_k\}$ is a family of nonempty subsets (sometimes called bags) of $\mathcal{V}$, where $k \leq n$, and edge set $\mathcal{H}$ is a tree on the k vertices, satisfying the Properties (P1), (P2) and (P3), or Properties (P1), (P2) and (P3')[1]:

- (P1) For each $i \in \mathcal{V}$, there exists a $C_r \in \mathcal{U}$, such that $i \in C_r$.
- (P2) For each $[i, j] \in \mathcal{E}$, there exists a $C_r \in \mathcal{U}$, such that $i, j \in C_r$.
- (P3) For two different vertices $C_s, C_t \in \mathcal{U}$, each vertex C_r in the path that connects C_s and C_t in $\mathcal{T}$ satisfies $C_s \bigcap C_t \subseteq C_r$.
- (P3') For any $i \in \mathcal{V}$, the set of vertices in $\mathcal{T}$ that contain i, defined by $\{C_r \,|\, i \in C_r, C_r \in \mathcal{U}\}$, forms a connected sub-tree of $\mathcal{T}$.

The width of $\mathcal{T}$ is defined by $\max\{|C_1|, \ldots, |C_k|\} - 1$. The treewidth of $\mathcal{G}$, denoted by $\text{tw}(\mathcal{G})$, is the minimum width among all tree decompositions of $\mathcal{G}$. Treewidth is an important parameter to describe how a graph looks like a tree. It can be shown that $\mathcal{G}$ is a tree if and only if $\text{tw}(\mathcal{G}) = 1$. Apparently, the tree decomposition of a graph is far from unique. For example, a trivial tree decomposition contains all vertices of the graph in its single root node, and has width $n - 1$. Finding a tree decomposition of a graph with minimum width is not easy. It is shown that for an integer r, to check whether $\mathcal{G}$ has a tree decomposition with width at most r is NP-complete, if r is part of the input [2].

Definition 2.3 (Chordal graph) A graph $\mathcal{G}$ is chordal if every cycle of $\mathcal{G}$ with length no less than four has at least one chord.

If a graph $\mathcal{G}$ is chordal, then there exists a tree decomposition $\mathcal{T} = (\mathcal{U}, \mathcal{H})$ such that $\mathcal{U}$ is exactly the set of all maximal cliques of $\mathcal{G}$. In this case, we call this tree decomposition $\mathcal{T}$ as the *clique decomposition* of $\mathcal{G}$ with $\text{tw}(\mathcal{G}) = |C^*| - 1$ exactly, where C^* is one maximum clique of $\mathcal{G}$. It has been shown that there exist linear-time algorithms (in terms of the number of vertices and edges) to test chordality of a graph and identify the maximal cliques of

[1] It is easy to check that Properties (P3) and (P3') are equivalent.

a chordal graph efficiently [36]. When $\mathcal{G}$ is not chordal, there are heuristic algorithms to expand $\mathcal{G}$ to a chordal graph by adding edges into $\mathcal{G}$ [19]. Figure 1 illustrates an example of chordal extension, tree decomposition and clique decomposition of a graph.

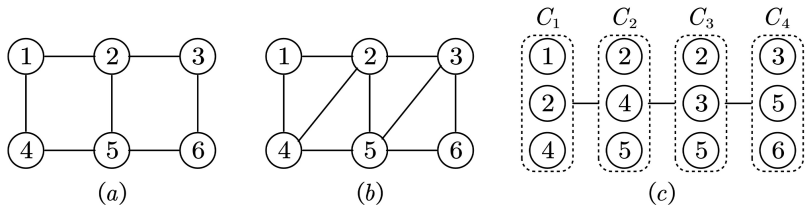

Fig. 1 Illustrations of the definitions in graph theory: (a) a ladder graph cited from Figure 4 in [28]; (b) the chordal-extension of the ladder graph; (c) the tree decomposition of the ladder graph, which is also the clique decomposition of the chordal graph in (b).

Now we give the definition of vertex contraction, which will be an important graph operation in the branching rule to be proposed in Subsection 4.2.

Definition 2.4 (Vertex contraction) Let $i, j \in \mathcal{V}$ be two different vertices in $\mathcal{G}$. By $\mathcal{G}/[i,j]$, we denote the graph obtained from $\mathcal{G}$ by contracting the two vertices i, j into a new vertex e, which is adjacent to all the neighbors of i and of j.

In the next, we define a similar vertex contraction operation in a tree decomposition.

Definition 2.5 (Vertex contraction in a tree decomposition) Let $\mathcal{T} = (\mathcal{U}, \mathcal{H})$ be a tree decomposition of $\mathcal{G}$, $C_s, C_t \in \mathcal{U}$ be two different vertices in $\mathcal{T}$, and $[C_s, C_t] \in \mathcal{H}$. By $\mathcal{T}/[C_s, C_t]$, we denote the graph obtained by contracting the two vertices C_s, C_t into a new vertex $C_e = C_s \bigcup C_t$, which is adjacent to all the neighbors of C_s and of C_t.

Notice that, in Definition 2.4, the contracted vertices i and j are not required to be adjacent in $\mathcal{G}$. They can be selected arbitrarily. However, in Definition 2.5, the pair of vertices C_s and C_t is required to be an edge in $\mathcal{T}$. This requirement is necessary to guarantee that $\mathcal{T}/[C_s, C_t]$ is still a tree. Following from Definition 2.5, we have the following result.

Lemma 2.1 *Let $\mathcal{T} = (\mathcal{U}, \mathcal{H})$ be a tree decomposition of $\mathcal{G}$ that has k vertices with $k \geq 2$. Then, for any $[C_s, C_t] \in \mathcal{H}$, $\mathcal{T}' := \mathcal{T}/[C_s, C_t]$ is also a tree decomposition of $\mathcal{G}$ with $k-1$ vertices.*

Proof It is easy to check that $\mathcal{T}'$ is a tree with $k-1$ vertices that satisfies properties (P1) and (P2) in Definition 2.2. We show that $\mathcal{T}'$ also satisfies

property (P3). Denote C_e as the vertex contracted from C_s and C_t. Let C_u, C_v be two different vertices in $\mathcal{T}'$, and $C_{i_1}, \ldots, C_{i_r}$ be the unique path in $\mathcal{T}'$ that connects C_u and C_v, where $C_{i_1} = C_u$, $C_{i_r} = C_v$, and r is the number of vertices in the path. Now consider the following three cases:

Case 1) If C_e is not in the path of $C_{i_1}, \ldots, C_{i_r}$, then the path is also a path in $\mathcal{T}$ and thus $C_u \bigcap C_v \subseteq C_p$ for all $C_p \in \{C_{i_1}, C_{i_2}, \ldots, C_{i_r}\}$.

Case 2) If C_e is in the interior of the path, i.e., there exists a vertex C_{i_w}, $1 < w < r$, such that $C_e = C_{i_w}$, then one of the following four paths is the unique path in $\mathcal{T}$ that connects C_u and C_v:

$$\begin{aligned}
&C_{i_1}, \ldots, C_{i_{w-1}}, C_s, C_t, C_{i_{w+1}}, \ldots, C_{i_r},\\
&C_{i_1}, \ldots, C_{i_{w-1}}, C_t, C_s, C_{i_{w+1}}, \ldots, C_{i_r},\\
&C_{i_1}, \ldots, C_{i_{w-1}}, C_s, C_{i_{w+1}}, \ldots, C_{i_r},\\
&C_{i_1}, \ldots, C_{i_{w-1}}, C_t, C_{i_{w+1}}, \ldots, C_{i_r}.
\end{aligned}$$

Hence, $C_u \bigcap C_v = C_{i_1} \bigcap C_{i_r} \subseteq C_p$ holds for each

$$C_p \in \{C_{i_1}, \ldots, C_{i_{w-1}}, C_{i_{w+1}}, \ldots, C_{i_r}\},$$

and either $C_u \bigcap C_v \subseteq C_s$ or $C_u \bigcap C_v \subseteq C_t$ holds (or both). Then, for a vertex C'_p in the path of $C_{i_1}, \ldots, C_{i_r}$ in $\mathcal{T}'$, either $C'_p \in \{C_{i_1}, \ldots, C_{i_{w-1}}, C_{i_{w+1}}, \ldots, C_{i_r}\}$ or $C'_p = C_s \bigcup C_t$ holds. In both situations, we have $C_u \bigcap C_v \subseteq C'_p$.

Case 3) If C_e is an end point of the path, without loss of generality, we may assume that $C_e = C_u$. Then either $C_s, C_t, C_{i_2}, \ldots, C_{i_r}$ or $C_t, C_s, C_{i_2}, \ldots, C_{i_r}$ is a path in $\mathcal{T}$. In both situations, we have $C_s \bigcap C_{i_r} \subseteq C_p$ and $C_t \bigcap C_{i_r} \subseteq C_p$ for all $C_p \in \{C_{i_2}, \ldots, C_{i_{r-1}}\}$. Then $C_u \bigcap C_v = (C_s \bigcup C_t) \bigcap C_{i_r} \subseteq C_p$ for all $C_p \in \{C_{i_1}, C_{i_2}, \ldots, C_{i_r}\}$.

Based on the discussions on the above three cases, we have shown that $\mathcal{T}'$ also satisfies property (P3) to be a tree decomposition of $\mathcal{G}$ that has $k-1$ vertices. □

3 A sparse polyhedral relaxation of the max-cut problem

In this section, we analyze how the chordal sparsity pattern affects the hardness of a max-cut problem. To do this, we design a sparse polyhedral relaxation of the max-cut problem. Although this polyhedral relaxation is not computational trackable, it helps us to understand when a max-cut problem can be relaxed to a convex problem that has a small relaxation gap. Such a convex relaxation provides a new approach to designing an effective branching rule for a branch-and-bound algorithm.

By introducing $X = xx^T$, problem (MC) can be reformulated as follows:

$$\begin{aligned}
\max \quad & Q \cdot X \\
\text{s.t.} \quad & X \in \mathbb{B}^n,
\end{aligned} \tag{RF1}$$

where $\mathbb{B}^n = \{X = xx^T \in \mathbb{S}^n \,|\, x_i \in \{-1, +1\}, i = 1, \ldots, n\}$. Note that $xx^T = (-x)(-x)^T$, hence there are 2^{n-1} distinct points in $\mathbb{B}^n$. Note that (RF1) has a linear objective function, the problem can be reformulated as

$$\begin{aligned} \max \quad & Q \cdot X \\ \text{s.t.} \quad & X \in \mathbb{P}^n, \end{aligned} \tag{RF2}$$

where $\mathbb{P}^n$, called the max-cut polytope of order n, represents the convex hull of $\mathbb{B}^n$. Because the number of extreme-points in $\mathbb{P}^n$ is 2^{n-1}, it is impractical to solve (RF2) directly unless n is small. However, when n is large and the problem has certain type of sparsity patterns, we may exploit the sparse patterns to design a tight polyhedral relaxation for (MC) from (RF2).

Assume that $Q \in \mathbb{S}^n(\mathcal{G})$, where $\mathcal{G}$ is a connected graph with vertices $\{1, \ldots, n\}$. Without loss of generality, we may also assume that $\mathcal{G}$ is a chordal graph, otherwise we can expand $\mathcal{G}$ to a chordal one by applying the heuristic algorithms in [19]. Let $\mathcal{T} = (\mathcal{U}, \mathcal{H})$ be a clique decomposition of $\mathcal{G}$ with $\mathcal{U} = \{C_1, \ldots, C_k\}$ being the set of all maximal cliques of $\mathcal{G}$. If $X \in \mathbb{P}^n$, then $X[C_r] \in \mathbb{P}^{n_r}$ holds for each $r = 1, \ldots, k$. Hence, we define the following polyhedral relaxation of (RF2):

$$\begin{aligned} \max \quad & Q \cdot X \\ \text{s.t.} \quad & X[C_r] \in \mathbb{P}^{n_r},\ r = 1, \ldots, k, \end{aligned} \tag{PR}$$

where $n_r = |C_r|$ and $\mathbb{P}^{n_r}$ is the max-cut polytope of order n_r for $r = 1, ..., k$. The constraint $X[C_r] \in \mathbb{P}^{n_r}$ can be represented by using the exact subgraph constraint formulation [13] or approximated by the facet-defining inequalities [3,4]. Here, we use the exact subgraph constraints to describe the constraint $X[C_r] \in \mathbb{P}^{n_r}$. Let $\mathbb{B}^{n_r}$ be the set of extreme points in $\mathbb{P}^{n_r}$, i.e., $\mathbb{B}^{n_r} = \{P_j^{n_r} \,|\, j = 1, \ldots, N_r\}$ with $N_r = 2^{n_r - 1}$ and $P_j^{n_r}$'s being the extreme points of $\mathbb{P}^{n_r}$. Then, $X[C_r] \in \mathbb{P}^{n_r}$ can be decomposed as

$$X[C_r] = \sum_{j=1}^{N_r} \lambda_j^r P_j^{n_r} \text{ with } \sum_{j=1}^{N_r} \lambda_j^r = 1 \text{ and } \lambda_1^r, \ldots, \lambda_{N_r}^r \geq 0. \tag{1}$$

A natural question is when the relaxation (PR) becomes tight. The following two lemmas are useful for deriving the conditions of the tightness of (PR).

Lemma 3.1 *Assume that $Q \in \mathbb{S}^n(\mathcal{G})$, $\mathcal{T} = (\mathcal{U}, \mathcal{H})$ is a tree decomposition of $\mathcal{G}$, and C_s and C_t are two adjacent vertices in the tree decomposition. If $X[C_s] \in \mathbb{P}^{n_s}$, $X[C_t] \in \mathbb{P}^{n_t}$, and $|C_s \bigcap C_t| \leq 3$, then there exists a matrix $\bar{P} \in \mathbb{P}^{|C_s \bigcup C_t|}$ such that, for the matrix $\bar{X} = \mathcal{X}(\bar{P}, C_s \bigcup C_t)$, we have $\bar{X}[C_s] = X[C_s]$ and $\bar{X}[C_t] = X[C_t]$.*

Proof Denote $n_{s,t} = |C_s \bigcap C_t|$ and $N_{s,t} = 2^{n_{s,t}-1}$. Let $\mathbb{B}^{n_{s,t}}$ be the set of extreme points of $\mathbb{P}^{n_{s,t}}$, represented by $\mathbb{B}^{n_{s,t}} := \{P_j^{n_{s,t}} \,|\, j = 1, \ldots, N_{s,t}\}$. Let $r = s$ or t, we partition the set $\{1, 2, \ldots, N_r\}$ into $N_{s,t}$ disjoint sets defined by

$$\mathcal{D}_q^r = \{j \,|\ P_j^{n_r} \in \mathbb{B}^{n_r},\ \mathcal{X}(P_j^{n_r}, C_r)[C_s \bigcap C_t] = P_q^{n_{s,t}}\},\ q = 1, \ldots, N_{s,t}.$$

Then the decomposition of $X[C_r]$ in (1) can be reformulated as

$$X[C_r] = \sum_{q=1}^{N_{s,t}} \sum_{j \in \mathcal{D}_q^r} \lambda_j^r P_j^{n_r}, \ \sum_{j=1}^{N_r} \lambda_j^r = 1 \text{ with } \lambda_1^r, \ldots, \lambda_{N_r}^r \geq 0.$$

Based on the construction of $\mathcal{D}_q^r$, $r = s$ or t, we have

$$X[C_s \bigcap C_t] = \sum_{q=1}^{N_{s,t}} \left(\sum_{j \in \mathcal{D}_q^s} \lambda_j^s \right) P_q^{n_{s,t}} = \sum_{q=1}^{N_{s,t}} \left(\sum_{j \in \mathcal{D}_q^t} \lambda_j^t \right) P_q^{n_{s,t}}.$$

When $n_{s,t} \leq 3$, it is easy to check that the points in the set $\mathbb{B}^{n_{s,t}}$ are affinely independent. Therefore,

$$\sum_{j \in \mathcal{D}_q^s} \lambda_j^s = \sum_{j \in \mathcal{D}_q^t} \lambda_j^t, \ q = 1, \ldots, N_{s,t}.$$

For each $q = 1, \ldots, N_{s,t}$, let $\mu_q = \sum_{j \in \mathcal{D}_q^s} \lambda_j^s = \sum_{j \in \mathcal{D}_q^t} \lambda_j^t$. For each pair $i \in \mathcal{D}_q^s$ and $j \in \mathcal{D}_q^t$, there exists an extreme point in $\mathbb{B}^{|C_s \bigcup C_t|}$, denoted by $P_{i,j}^{|C_s \bigcup C_t|}$, such that

$$\mathcal{X}(P_{i,j}^{|C_s \bigcup C_t|}, C_s \bigcup C_t)[C_s] = P_i^s$$

and

$$\mathcal{X}(P_{i,j}^{|C_s \bigcup C_t|}, C_s \bigcup C_t)[C_t] = P_j^t.$$

Then, we can construct

$$\bar{P} = \sum_{q=1,\ldots,N_{s,t}, \mu_q \neq 0} \frac{1}{\mu_q} \sum_{i \in D_q^s} \sum_{j \in D_q^t} \lambda_i^s \lambda_j^t P_{i,j}^{|C_s \bigcup C_t|}$$

and $\bar{X} = \mathcal{X}(\bar{P}, C_s \bigcup C_t)$. It is not difficult to verify that $\bar{P} \in \mathbb{P}^{|C_s \bigcup C_t|}$, $\bar{X}[C_s] = X[C_s]$ and $\bar{X}[C_t] = X[C_t]$. □

Lemma 3.2 *Assume that $Q \in \mathbb{S}^n(\mathcal{G})$ and $\mathcal{T} = (\mathcal{U}, \mathcal{H})$ is a tree decomposition of $\mathcal{G}$. If $|C_s \bigcap C_t| \leq 3$ for all $s, t \in \{1, \ldots, k\}$ with $s \neq t$, then for a matrix $X \in \mathbb{R}^{n \times n}$, the following two statements are equivalent:*

- *(S1) There exists a matrix $\bar{X} \in \mathbb{P}^n$ such that the equation $\bar{X}[C_r] = X[C_r]$ holds for $r = 1, \ldots, k$.*
- *(S2) $X[C_r] \in \mathbb{P}^{n_r}$ for $r = 1, \ldots, k$.*

Proof If (S1) is true, then we naturally have $X[C_r] = \bar{X}[C_r] \in \mathbb{P}^{n_r}$ for all $r = 1, \ldots, k$, thus (S2) is true. We now show the reverse direction using the induction on k.

For $k = 2$, following Lemma 3.1, we can construct a matrix $\bar{X} \in \mathbb{P}^{|C_1 \bigcup C_2|} = \mathbb{P}^n$ such that $\bar{X}[C_1] = X[C_1]$ and $\bar{X}[C_2] = X[C_2]$. Hence (S1) holds for $k = 2$.

Now assume that (S1) can be derived from (S2) when $k \leq q$, where $q \geq 2$, and consider the case of $k = q + 1$. We choose a leaf vertex C_s from the

tree $\mathcal{T}$ and let C_t be the unique neighborhood of C_s. Using Lemma 3.1, we can construct a matrix $\bar{P} \in \mathbb{P}^{|C_s \bigcup C_t|}$ such that, for $\hat{X} = \mathcal{X}(\bar{P}, C_s \bigcup C_t)$, the equations $\hat{X}[C_s] = X[C_s]$ and $\hat{X}[C_t] = X[C_t]$ hold. Then, we generate a new matrix $\tilde{X}$ by assigning $\tilde{X}[C_s \bigcup C_t] = \bar{P}$ and $\tilde{X}_{ij} = X_{ij}$ if $i \notin C_s \bigcup C_t$ or $j \notin C_s \bigcup C_t$. Based on this construction, we know that if $\tilde{X}_{ij} \neq X_{ij}$, then either $i \in C_s \setminus C_t, j \in C_t \setminus C_s$ or $i \in C_t \setminus C_s, j \in C_s \setminus C_t$ must hold. Now we analyze the block $\tilde{X}[C_u]$ where $u \notin \{s, t\}$. Since $C_u \bigcap C_s \subseteq C_t$ (according to the running intersection property (P3) of tree decomposition), which implies $(C_u \bigcap C_s) \setminus C_t = \emptyset$, i.e., for any $i \in C_u$, we have $i \notin C_s \setminus C_t$. Hence, for any $i, j \in C_u$, the equation $\tilde{X}_{ij} = X_{ij}$ must hold, and thus $\tilde{X}[C_u] = X[C_u] \in \mathbb{P}^{|C_u|}$. Then, $\tilde{X}$ satisfies $\tilde{X}[C_r] \in \mathbb{P}^{|C_r|}$ for all vertices C_r in $\mathcal{T}/[C_s, C_t]$, and the contracted tree $\mathcal{T}/[C_s, C_t]$ has $q = k - 1$ vertices. Moreover, let C_e be the vertex contracted from $[C_s, C_t]$. Then, for a neighbor C_r of C_e, we have $C_r \bigcap C_e = C_r \bigcap (C_s \bigcup C_t) = C_r \bigcap C_t$, where the last equality holds due to the property $C_r \bigcap C_s \subseteq C_t$. Thus, we have shown that $|C_r \bigcap C_e| = |C_r \bigcap C_t| \leq 3$. Consequently, $\mathcal{T}/[C_s, C_t]$ is another tree decomposition of $\mathcal{G}$ such that for each pair of vertices C_s and C_t in $\mathcal{T}/[C_s, C_t]$, we have $|C_s \bigcap C_t| \leq 3$. By induction, we know that (S1) is true. □

Note that Lemma 2.1 shows that $\mathcal{T}/[C_s, C_t]$ is a tree decomposition, but not necessary a clique decomposition, of the chordal graph $\mathcal{G}$. In general, the set $C_e = C_s \bigcup C_t$ may not be a clique in $\mathcal{G}$. Hence, in Lemmas 3.1 and 3.2, we assume that $\mathcal{T}$ is a tree decomposition, rather than a clique decomposition, of $\mathcal{G}$. Furthermore, in the proof of Lemma 3.2, the induction procedure only requires $\mathcal{T}/[C_s, C_t]$ to be a tree decomposition. Now, we are ready to prove the conditions for the tightness of the polyhedral relaxation (PR).

Theorem 3.1 *Assume that $Q \in \mathbb{S}^n(\mathcal{G})$ and $\mathcal{T} = (\mathcal{U}, \mathcal{H})$ is a tree decomposition of $\mathcal{G}$. If $|C_s \bigcap C_t| \leq 3$ for all $s, t \in \{1, \ldots, k\}$ with $s \neq t$, then the polyhedral relaxation* (PR) *is tight.*

Proof Since (PR) is a relaxation of (RF2), we only need to show that, for every feasible solution X of (PR), there is a feasible solution $\bar{X}$ of (RF2) such that both X and $\bar{X}$ give the same objective value. Following Lemma 3.2, we know that X is feasible to (PR) if and only if there is a matrix $\bar{X} \in \mathbb{P}^n$ such that $\bar{X}[C_r] = X[C_r]$ for $r = 1, \ldots, k$. Since $Q \in \mathbb{S}^n(\mathcal{G})$, we have $Q \cdot \bar{X} = \sum_{[i,j] \in \mathcal{E}} Q_{ij} \bar{X}_{ij}$ and $Q \cdot X = \sum_{[i,j] \in \mathcal{E}} Q_{ij} X_{ij}$. Note that $\mathcal{T} = (\mathcal{U}, \mathcal{H})$ is a tree decomposition of $\mathcal{G}$, then, for any $[i, j] \in \mathcal{E}$, there is a $C_r \in \mathcal{U}$, $r \in \{1, ..., k\}$, such that $i, j \in C_r$. Therefore, we have $Q \cdot \bar{X} = Q \cdot X$. Consequently, (PR) is equivalent to (RF2), and thus tight. □

Corollary 3.1 *Assume that $Q \in \mathbb{S}^n(\mathcal{G})$ and $\mathcal{T} = (\mathcal{U}, \mathcal{H})$ is a tree decomposition of $\mathcal{G}$. If the width of $\mathcal{T}$ is no more than* 3*, then the polyhedral relaxation* (PR) *is tight.*

Proof When the width of $\mathcal{T}$ is no more than 3, we have that $|C_s \bigcap C_t| \leq 3$ for all $s, t \in \{1, \ldots, k\}$ with $s \neq t$. The conclusion then follows from Theorem 3.1. □

Based on Theorem 3.1 and Corollary 3.1, we can see that relaxation (PR) is tight if the treewidth of the sparsity pattern $\mathcal{G}$ corresponding to Q is no more than three, or if the intersection set $C_s \bigcap C_t$ among different maximal cliques has no more than three elements. These theoretical results give the sufficient conditions for the tightness of relaxation (PR).

4 The hierarchy branching strategy

The sufficient conditions for the tightness of problem (PR) provide important insights on how to design an effective branch-and-bound rule – if the treewidth of the sparsity pattern is small, and the intersections of different cliques have very few vertices, then (PR) is likely to be a tight relaxation for problem (MC). Motivated by this intuitive idea, we exploit the clique decomposition structure in the sparsity pattern of problem (MC) to design a new branching rule, such that the sparsity patterns of subproblems in the deep level of the proposed branch-and-bound algorithm have small treewidth and small clique intersection. The new features of the proposed algorithm include:

- A new branching rule, which aims to manipulate the sparsity pattern of the problem towards a sparsity pattern that has a low treewidth and few vertices in the intersections of different cliques.
- A new cutting-plane selection scheme that only involves the valid inequalities on variables in $X[C_r]$ for $r = 1, \ldots, k$.

To further explain the ideas of the proposed branch-and-bound techniques, we first analyze the relation between the proposed polyhedral relaxation and the classical semidefinite relaxations of the max-cut problem. Then we develop a new semidefinite relaxation based branch-and-bound algorithm

4.1 Semidefinite relaxations

If there is a clique $C_r \in \mathcal{U}$ with large $n_r = |C_r|$, then it is computationally expensive to solve (PR) directly. A practical way is to further relax $X[C_r] \in \mathbb{P}^{n_r}$ to a computationally efficient relaxation. One straightforward way is to relax $X[C_r] \in \mathbb{P}^{n_r}$ to $X[C_r] \succeq 0$, and then add valid inequalities on $X[C_r]$. One set of such valid inequalities is $X_{ii} = 1$ for $i = 1, \ldots, n$. Then, we arrive at the following semidefinite relaxation:

$$\begin{aligned} \max \quad & Q \cdot X \\ \text{s.t.} \quad & X_{ii} = 1, \ i = 1, \ldots, n, \\ & X[C_r] \succeq 0, \ r = 1, \ldots, k. \end{aligned} \tag{CSDR}$$

This relaxation has actually appeared in [12]. It is a compact reformulation of the following classical semidefinite relaxation for (MC):

$$\begin{aligned} \max \quad & Q \cdot X \\ \text{s.t.} \quad & X_{ii} = 1, \ i = 1, \ldots, n, \\ & X \succeq 0. \end{aligned} \tag{SDR}$$

To see the equivalence between (CSDR) and (SDR), we cite the following result of [18]:

Lemma 4.1 ([18], Theorem 7) *Assume that $\mathcal{G}$ is a chordal graph and $X \in \mathbb{S}^n(\mathcal{G})$. Let $\mathcal{T} = (\mathcal{U}, \mathcal{H})$ be a clique decomposition of $\mathcal{G}$, where $\mathcal{U} = \{C_1, \ldots, C_k\}$ is the set of all maximal cliques in $\mathcal{G}$. Then the following two statements are equivalent:*
(S1) $X[C_r] \succeq 0$ for $r = 1, \ldots, k$.
(S2) There exists an $\bar{X} \in \mathbb{S}^n_+$ such that $\bar{X}[C_r] = X[C_r]$ for $r = 1, \ldots, k$.

As indicated in Theorem 3.1 and Corollary 3.1, if the treewidth of sparsity pattern is no more than three or the intersection of any two cliques has no more than three vertices, then the relaxation (PR) is tight. Since (CSDR) is derived by further relaxing (PR), it can be regarded as an approximation of (PR). Hence, when the treewidth of the sparsity pattern is reduced, and the number of elements in the intersection of cliques is small, it is expected that the gap between (CSDR) and (MC) is small. Intuitively speaking, the gap between the polyhedral constraint $X[C_r] \in \mathbb{P}^{n_r}$ and its semidefinite relaxation $X[C_r] \succeq 0$ depends on the size of C_r. When n_r is small, $\mathbb{P}^{n_r}$ is a low-dimension polyhedral, and the gap between $\mathbb{P}^{n_r}$ and its semidefinite relaxation is small [21]. In this sense, when the treewidth of the sparsity pattern is small, the semidefinite relaxation (CSDR) or (SDR) may involve a tight relaxation to each sub-matrix constraint $X[C_r] \in \mathbb{P}^{n_r}$ such that it is likely to provide a tight bound.

This motivates us to design a new branching rule such that the treewidth of the sparsity pattern and the intersection of any two cliques are reduced as the branch-and-bound tree traverses deep. To achieve this purpose, we propose a new branching rule based on the clique decomposition of sparsity pattern in problem (MC) in the next section.

4.2 A hierarchy branching rule

In this subsection, we will develop a new branching rule based on the clique decomposition of the sparsity pattern of problem (MC). In the existing three semidefinite relaxation based branch-and-bound algorithms [20,33,25], two branching rules are widely adopted: the easy first rule (called rule R2), and the difficult first rule (called rule R3). For an optimal solution X of a semidefinite relaxation, rule R2 selects i and j such that their rows of X are closest to a $\{-1, 1\}$ vector, while rule R3 picks the one which minimizes $|X_{ij}|$.

As we can see that both rules R2 and R3 are numerical-driven rules, which only consider the information of the optimal solution of current relaxation problem. However, the branching procedure has significant impacts on the sparsity pattern of the problem. Note that by assigning X_{ij} to either $X_{ij} = 1$ or $X_{ij} = -1$, the constraint $x_i = x_j$ or $x_i = -x_j$ is added implicitly in the child nodes. This is equivalent to say that the vertices i and j is contracted to a new vertex in the sparsity pattern of the child problems. Viewing from this perspective, the branching rules R2 and R3 are actually vertex contraction on the sparsity pattern of the problem. Here, different from the above two branching rules, we design a new rule that takes the sparsity pattern of the problem into consideration.

We first analyze how the branching procedure will affect the sparsity pattern of a problem. For a problem with $Q \in \mathbb{S}^n(\mathcal{G})$, when we select a variable X_{ij} to branch, the branching procedure will eliminate the variable x_j by replacing it with x_i or $-x_i$ in the child nodes, the dimension of the problem reduces by one, and the matrix Q in the objective function can be updated by Algorithm 1 as given below, where $p = -1$ in case $x_i = -x_j$, and $p = 1$ in case $x_i = x_j$. The set $\mathcal{V}_Q := \{\ell_1, \ldots, \ell_r\} \subseteq \{1, \ldots, n\}$ in Algorithm 1 defines an ordered sequence, in which, for each $s = 1, \ldots, r$, ℓ_s represents the vertex in $\mathcal{G}$ that the s-th row (or column) of Q corresponds to. In a branch-and-bound algorithm, $\mathcal{V}_Q$ is initialized as the set of all vertices in the root node. After some branching steps, some variables have been eliminated, and the number of undetermined variables in the objective function will be reduced to r with $r < n$. In general, $\mathcal{V}_Q$ in an enumeration node is a subset of $\mathcal{V}$. Algorithm 1 provides a general procedure for the case $Q \in \mathbb{S}^r$ that corresponds to a subset of the vertices. The new vertex generated by contracting i and j is denoted by e. Then, we update $\mathcal{V}_Q$ to a new ordered sequence $\mathcal{V}_{Q'}$ according to the following $\mathcal{V}$-Procedure:

$\mathcal{V}$-Procedure: Replace the vertex i in the ordered sequence $\mathcal{V}_Q$ by e, and delete the vertex j from the sequence to generate a new sequence $\mathcal{V}_{Q'}$.

Algorithm 1 Update the matrix Q for vertex merging

1: **Input:** $Q \in \mathbb{S}^r$, $i, j \in \mathcal{V}_Q := \{\ell_1, \ldots, \ell_r\}$ and $p \in \{-1, +1\}$.
2: **Output:** A new matrix $Q' \in \mathbb{S}^{r-1}$ and a new ordered sequence $\mathcal{V}_{Q'}$.
3: $Q_{:i} \leftarrow Q_{:i} + pQ_{:j}$
4: $Q_{i:} \leftarrow Q_{i:} + pQ_{j:}$
5: Set $C = \{1, \ldots, r\} \setminus \{t\}$ and $Q' := Q[C]$, where t is the index that $\ell_t = j$.
6: Use $\mathcal{V}$-Procedure to generate $\mathcal{V}_{Q'}$.
7: Return Q' and $\mathcal{V}_{Q'}$.

To simplify the notations, we use $Q_{\mathcal{V}_Q(s,t)}$ to represent the entry of Q in the row that corresponds to s and the column that corresponds to t, where s and t are two vertices in $\mathcal{V}_Q$. The next lemma shows how the branching procedure affects the sparsity pattern of the problem.

Lemma 4.2 *For $Q \in \mathbb{S}^r(\mathcal{G})$, if Q' is the matrix updated by Algorithm 1 with a given pair of indices $i, j \in \mathcal{V}_Q$, then $Q' \in \mathbb{S}^{r-1}(\mathcal{G}/[i,j])$.*

Proof Let e denote the vertex contracted from i and j and $\mathcal{G}' = \mathcal{G}/[i,j]$. For two vertices $s, t \in \mathcal{V}_{Q'}$, if $s \neq e$ and $t \neq e$, then the entity $Q'_{\mathcal{V}'_Q(s,t)}$ is nonzero if and only if $Q_{\mathcal{V}_Q(s,t)}$ is a nonzero of Q because the row and column in Q, corresponding to s and t, respectively, do not change in this case. Since $Q_{\mathcal{V}_Q(s,t)} \neq 0$, $[s,t]$ is an edge of $\mathcal{G}$. Therefore, it is also an edge of $\mathcal{G}'$. On the other hand, when one of the two vertices is e, say, $s = e$ and $t \neq e$, then $Q'_{\mathcal{V}'_Q(s,t)}$ is nonzero if either $Q_{\mathcal{V}_Q(i,t)}$ or $Q_{\mathcal{V}_Q(j,t)}$ is nonzero, i.e., either $[i,t]$ or $[j,t]$ is an edge of $\mathcal{G}$. In this case, according to the definition of vertex contraction, $[e,t]$ is an edge of $\mathcal{G}'$. Consequently, we have $Q' \in \mathbb{S}^{r-1}(\mathcal{G}')$. □

Lemma 4.2 shows that the branching procedure is in fact a vertex contraction on the sparsity pattern of the problem. Thus, the sparsity pattern will be changed in general. However, the chordal structure in the sparsity pattern will be kept when it is a chordal graph and an appropriate variable is carefully selected to branch. We prove this result in the next theorem.

Theorem 4.1 *Let $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ be a chordal graph and $[i,j] \in \mathcal{E}$, then the graph $\mathcal{G}' = \mathcal{G}/[i,j]$ remains to be a chordal graph.*

Proof Let e represent the new vertex in $\mathcal{G}'$ that is contracted from vertices i, j. Let $\ell_1, \ldots, \ell_r, \ell_1$ be a cycle in $\mathcal{G}'$ of length $r \geq 4$, we show that there is a chord in the cycle. Consider the following situations: If $e \notin \{\ell_1, \ldots, \ell_r\}$, then $\ell_1, \ldots, \ell_r, \ell_1$ is also a cycle in $\mathcal{G}$ of length r, and the chord of the cycle in $\mathcal{G}$ is also a chord of the same cycle in $\mathcal{G}'$. Otherwise, when $e \in \{\ell_1, \ldots, \ell_r\}$, without loss of generality, the cycle can be represented by $e, \ell_2, \ldots, \ell_r, e$. There are four possible cases to be discussed.

Case 1: The sequence $i, \ell_2, \ldots, \ell_r, i$ is a cycle in $\mathcal{G}$. In this case, there exists a chord of the cycle in graph $\mathcal{G}$. If the chord is constructed by $[\ell_s, \ell_t]$ with $\ell_s, \ell_t \in \{\ell_2, \ldots, \ell_r\}$, then $[\ell_s, \ell_t]$ is also a chord of the cycle $\ell_1, \ldots, \ell_r, \ell_1$ in $\mathcal{G}'$. Otherwise, the chord is constructed by $[i, \ell_t]$ for some $\ell_t \in \{\ell_3, \ldots, \ell_{r-1}\}$, then $[e, \ell_t]$ is a chord of the cycle $\ell_1, \ldots, \ell_r, \ell_1$ in $\mathcal{G}'$.

Case 2: The sequence $j, \ell_2, \ldots, \ell_r, j$ is a cycle in $\mathcal{G}$. Using the same arguments as Case 1, we can show that there exists a chord of the cycle $\ell_1, \ldots, \ell_r, \ell_1$ in $\mathcal{G}'$.

Case 3: The sequence $i, j, \ell_2, \ldots, \ell_r, i$ is a cycle of length $r+1$ in $\mathcal{G}$. In this case, there exists a chord of the cycle in graph $\mathcal{G}$. If the chord is constructed by $[\ell_s, \ell_t]$ with $\ell_s, \ell_t \in \{\ell_2, \ldots, \ell_r\}$, then $[\ell_s, \ell_t]$ is also a chord of the cycle $\ell_1, \ldots, \ell_r, \ell_1$ in $\mathcal{G}'$. Otherwise, the chord is constructed by $[w, \ell_t]$ for $w \in \{i, j\}$ and $\ell_t \in \{\ell_2, \ldots, \ell_r\}$. We further consider the three subcases: (i) $\ell_t \in \{\ell_3, \ldots, \ell_{r-1}\}$. In this subcase, $[e, \ell_t]$ is a chord of the cycle $\ell_1, \ldots, \ell_r, \ell_1$ in $\mathcal{G}'$; (ii) $[w, \ell_t] = [i, \ell_2]$. In this subcase, $i, \ell_2, \ldots, \ell_r, i$ is another cycle in $\mathcal{G}$, then this subcase is reduced to Case 1, in which we have shown that there exists a chord of $\ell_1, \ldots, \ell_r, \ell_1$ in $\mathcal{G}'$; (iii) $[w, \ell_t] = [j, \ell_r]$. In this subcase, $j, \ell_2, \ldots, \ell_r, j$

becomes a cycle in $\mathcal{G}$, and this subcase is reduced to Case 2, thus there exists a chord of $\ell_1, \ldots, \ell_r, \ell_1$ in $\mathcal{G}'$.

Case 4: The sequence $j, i, \ell_2, \ldots, \ell_r, j$ is a cycle of length $r+1$ in $\mathcal{G}$. Using the same arguments as Case 3, we can show that there exists a chord of the cycle $\ell_1, \ldots, \ell_r, \ell_1$ in $\mathcal{G}'$.

In all, we conclude that for any cycle in $\mathcal{G}'$ of length $r \geq 4$, there must exist a chord in the cycle. Hence $\mathcal{G}'$ is chordal. □

Theorem 4.1 shows that the chordal structure of the sparsity pattern will be kept if we only select the variables X_{ij} with $[i,j] \in \mathcal{E}$ as candidates for branching. As we have mentioned, the purpose of our new branching rule is to reduce the treewidth of sparsity pattern, if possible, after branching. To do this, we need the next theorem.

Theorem 4.2 *Let $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ be a chordal graph, for any $[i,j] \in \mathcal{E}$, if $\mathcal{T} = (\mathcal{U}, \mathcal{H})$ with $\mathcal{U} = \{C_1, \ldots, C_k\}$ being a clique decomposition of $\mathcal{G}$, then $\mathcal{T}' = (\mathcal{U}', \mathcal{H}')$ is a tree decomposition of $\mathcal{G}' = \mathcal{G}/[i,j]$ with*

$$C'_r = \begin{cases} C_r, & \text{if } i \notin C_r \text{ and } j \notin C_r, \\ C_r \bigcup \{e\} \setminus \{i,j\}, & \text{otherwise,} \end{cases} \quad r = 1, \ldots, k, \tag{2}$$

where e is the new vertex generated by the contraction of the vertices i and j in $\mathcal{G}$, and for two vertices $C'_s, C'_t \in \mathcal{U}'$, $[C'_s, C'_t] \in \mathcal{H}'$ if and only if $[C_s, C_t] \in \mathcal{H}$.

Proof It is easy to verify that $\mathcal{T}'$ satisfies properties (P1) and (P2) in Definition 2.2, we only need to show that it satisfies property (P3').

For a vertex w in $\mathcal{G}'$, if $w \neq e$, then the set of vertices $\{C'_t \,|\, w \in C'_t, C'_t \in \mathcal{U}'\}$ is the same as the set of vertices $\{C_t \,|\, w \in C_t, C_t \in \mathcal{U}\}$, thus property (P3') holds automatically. Otherwise, we consider the case that $w = e$. Since $[i,j] \in \mathcal{E}_{\mathcal{G}}$, there is a vertex $C_r \in \mathcal{U}$ such that both i and j are in C_r. Based on property (P3') for $\mathcal{T}$, the sets $\mathcal{U}_i := \{C_t \,|\, C_t \in \mathcal{U}, i \in C_t\}$ and $\mathcal{U}_j := \{C_t \,|\, C_t \in \mathcal{U}, j \in C_t\}$ induce connected subgraphs in $\mathcal{T}$, denoted by $\mathcal{T}[\mathcal{U}_i]$ and $\mathcal{T}[\mathcal{U}_j]$, respectively. Since $C_r \in \mathcal{U}_i \bigcap \mathcal{U}_j$, the intersection of the two connected subgraphs $\mathcal{T}[\mathcal{U}_i]$ and $\mathcal{T}[\mathcal{U}_j]$ is nonempty. Hence, the subgraph $\mathcal{T}[\mathcal{U}_i \bigcup \mathcal{U}_j]$ is also a connected subgraph in $\mathcal{T}$. Note that for each $C'_t \in \mathcal{U}'$, we have $e \in C'_t$ if and only if $i \in C_t$ or $j \in C_t$, i.e., $C_t \in \mathcal{U}_i \bigcup \mathcal{U}_j$. Then the set of vertices in $\mathcal{U}'$ that contains e induces a connected subgraph of $\mathcal{T}'$. Therefore, property (P3') holds for $\mathcal{T}'$. □

Theorem 4.2 shows that it is possible to reduce the treewidth of the problem if the indices i and j of the selected variable X_{ij} are both in the same maximum clique of the $\mathcal{G}$. Moreover, we have the following two observations:

(i) If $i, j \in C_r$ for some $r \in \{1, \ldots, k\}$, then $|C'_r| = |C_r| - 1$.
(ii) If $i, j \in C_s \bigcap C_t$ for some $s, t \in 1, \ldots, k$, then $|C'_s \bigcap C'_t| = |C_s \bigcap C_t| - 1$.

Hence, if we select a suitable pair $[i,j] \in \mathcal{E}$, then we may reduce not only the number of vertices in some cliques, but also the number of vertices in the intersections of different maximal cliques.

On the other hand, if we arbitrarily select a pair $[i,j] \notin \mathcal{E}$ for contracting, then the chordal structure of the sparsity pattern may be destroyed and the treewidth of graph $\mathcal{G}$ may even increase. For example, consider a simple graph $\mathcal{G}$ with 5 vertices $\{1,\ldots,5\}$ and 4 edges

$$\{[1,2],[2,3],[3,4],[4,5]\}.$$

It is easy to see that $\mathcal{G}$ is a tree (and also a chordal graph), with treewidth being 1. Let $\mathcal{G}' = \mathcal{G}/[1,5]$. Then $\mathcal{G}'$ becomes a circle with vertices $\{e,2,3,4\}$ and edges

$$\{[e,2],[e,4],[2,3],[3,4]\}.$$

It is straightforward to verify that $\mathcal{G}'$ is not a chordal graph and the treewidth of $\mathcal{G}'$ is increased to 2. Hence, the sparsity pattern of the problem may become more complicated if the pair i,j to be contracted is not well selected.

The above simple example indicates that we should carefully choose a variable to branch in the algorithm to keep the chordal structure intact. In our branch-and-bound algorithm, we only select the variables X_{ij} with $[i,j] \in \mathcal{E}$ as candidates to branch. An immediate question is whether such an entry exists when the relaxation (SDR) is not tight. The following theorem gives a positive answer to this question.

Theorem 4.3 *Assume that $\mathcal{G} = \{\mathcal{V},\mathcal{E}\}$ is a connected chordal graph. Let X^* be an optimal solution of* (SDR). *If $|X^*_{ij}| = 1$ for all $[i,j] \in \mathcal{E}$, then problem* (SDR) *is tight.*

Proof We prove the theorem by showing that X^* is a rank-one matrix. Since $X^* \succeq 0$, it can be decomposed as $X^* = V^\top V$, where $V := [v_1,...,v_n] \in \mathbb{S}^n$ and $v_i \in \mathbb{R}^n$ for $i = 1,...,n$. We have $v_i^\top v_i = X^*_{ii} = 1$ for all $i = 1,...,n$, thus each column of V is a unit vector. For any entry X^*_{ij} with $[i,j] \in \mathcal{E}$, since $|X^*_{ij}| = 1$, we have $|v_i^\top v_j| = |X^*_{ij}| = 1$. Following the Cauchy-Schwartz inequality $|v_i^T v_j| \le \|v_i\|\|v_j\|$, together with $\|v_i\| = 1$, $\|v_j\| = 1$, the equality $|v_i^T v_j| = 1$ holds if and only if $v_i = v_j$ or $v_i = -v_j$. Also note that $\mathcal{G}$ is connected, then for any $j \in \{2,\ldots,n\}$, there exists a path $i_0,\ldots,i_r$ with $i_0 = 1$ and $i_r = j$, such that $(i_t,i_{t+1}) \in \mathcal{E}$ for $t = 0,1,...,r-1$. Hence, $v_1 = v_j$ or $v_1 = -v_j$ for any $j \in \{2,...,n\}$. It follows that V is a rank-one matrix, so is X^*. □

Theorem 4.3 shows that if the relaxation (SDR) is not tight, then there must exist an edge $[i,j] \in \mathcal{E}$ such that $|X^*_{ij}| < 1$, where X^* is an optimal solution of (SDR). Therefore, it is valid to design a branching rule that only selects variables X_{ij} with $[i,j] \in \mathcal{E}$ to branch.

Obviously, there could be multiple candidates X_{ij} with $[i,j] \in \mathcal{E}$. To pick the one that is effective in reducing the treewidth and the number of vertices in the intersections of maximal cliques, the proposed branching rule needs to consider the priorities of the edges $[i,j]$ in $\mathcal{E}$. That is, in order to reduce the treewidth of the sparsity pattern, the edges in the maximum cliques should have higher priorities. Similarly, in order to reduce the number of vertices in

the intersections of cliques, the edges in many different maximal cliques will have higher priorities. For this purpose, we define two priority values for each $[i,j] \in \mathcal{E}$:

- $P_{ij}^1 = \max_{i,j \in C_r} |C_r|$ is the largest cardinality of the maximal cliques that contain both vertices i and j.
- $P_{ij}^2 = |\{C_r \,|\, i,j \in C_r,\ r = 1,\ldots,k\}|$ is the number of different maximal cliques that contain both vertices i and j.

A good candidate $[i,j] \in \mathcal{E}$ should have large values in both P_{ij}^1 and P_{ij}^2. We then define the unified priority value P_{ij} of $[i,j]$ as the cardinality of the set $\mathcal{U}_{i,j}$, where

$$\mathcal{U}_{i,j} = \{v \in \mathcal{V} \,|\, [i,v] \in \mathcal{E},\ [j,v] \in \mathcal{E}\}\,.$$

Note that $\mathcal{G}$ is assumed to be a chordal graph (or have been expanded to a chordal graph) and the set $\mathcal{U}_{i,j}$ consists of all vertices v that are adjacent to both i and j in $\mathcal{G}$, P_{ij} actually denotes the number of vertices that are located in the same cliques with i and j. When P_{ij} is large for some $[i,j] \in \mathcal{E}$, there are many vertices v adjacent to both i and j. If these vertices v belong to one maximal clique, then the cardinality of this maximal clique is large and P_{ij}^1 is large. On the other hand, if these vertices v belong to different maximal cliques, then P_{ij}^2 is large. Therefore, if P_{ij} is large, then either P_{ij}^1 or P_{ij}^2 should be large. In this sense, P_{ij} simultaneously takes the priority values P_{ij}^1 and P_{ij}^2 into consideration. We prefer to select the variable X_{ij} with the largest priority value P_{ij} and $[i,j] \in \mathcal{E}$ as the one to branch.

Based on the above discussion, we now propose our new branching rule, referred to as **hierarchy branching rule** (rule HB in short): From the set $\big\{[i,j] \in \mathcal{E} \,\big|\, |X_{ij}| < 1\big\}$, select the pair $[i,j]$ that achieves the largest priority value P_{ij} for branching. If there are multiple edges that achieve the largest priority, then we break the tie by using the conventional branching rule R3, i.e., we select the variable such that $|X_{ij}|$ achieves the smallest value among the candidate edges $[i,j]$ that have the largest priority. If there still exist more than one candidate, then we randomly select one among them.

4.3 A sparsity-pattern driven cutting-plane scheme

In this subsection, we develop a cutting-plane selection scheme based on the sparsity pattern to further improve the efficiency of the branch-and-bound method for the max-cut problem.

Cutting planes are usually added into the relaxation problem (SDR) to obtain a tighter upper bound for problem (MC). One set of such cutting planes are the so-called triangle inequalities proposed in [20]:

$$\left.\begin{array}{r} X_{ij} + X_{it} + X_{jt} \geq -1 \\ X_{ij} - X_{it} - X_{jt} \geq -1 \\ -X_{ij} + X_{it} - X_{jt} \geq -1 \\ -X_{ij} - X_{it} + X_{jt} \geq -1 \end{array}\right\} \text{ for all } i < j < t, \text{ and } i,j,t \in \mathcal{V}. \qquad (3)$$

Let X^* be an optimal solution of (SDR) before adding any triangle inequalities in (3). It is not difficult to find the most violated γ inequalities from (3) by X^* and add them into (SDR). We call this process as traditional cutting-plane (TCP in short) scheme.

To keep the algorithm efficient it is extremely important to select just a small number of promising inequalities from the vast set of violated triangle inequalities. Hence, we derive a heuristic criterion based on the clique decomposition to select the violated triangle inequalities from a subset of (3). By the equivalent formulation of (SDR), i.e., problem (CSDR), we can see that variable X_{ij} can be eliminated whenever i and j do not simultaneously belong to any clique C_r, $r = 1, ..., k$. In this case, the triangle inequalities involving X_{ij} would be expected to have little improvement in the bound. To avoid selecting these triangle inequalities, we only enumerate triangle inequalities from the following set:

$$\left.\begin{array}{r} X_{ij} + X_{it} + X_{jt} \geq -1 \\ X_{ij} - X_{it} - X_{jt} \geq -1 \\ -X_{ij} + X_{it} - X_{jt} \geq -1 \\ -X_{ij} - X_{it} + X_{jt} \geq -1 \end{array}\right\} \text{ for all } i < j < t, \text{ and } i, j, t \in C_r, \tag{4}$$

for each clique C_r, $r = 1, ..., k$, and add no more than γ triangle inequalities that are most violated by $X^*[C_r]$ into (SDR). We call this process as sparsity driven cutting-plane (SCP in short) scheme. Note that SCP is different from TCP in that the entries i, j and t are required to be in the same clique C_r.

Following the SCP scheme, the added valid inequalities have the form $A_i \cdot X \leq b_i$, $i = 1, \ldots, m \leq \gamma$ with $A_i \in \mathbb{S}^n(\mathcal{G}[C_r])$, and $\mathcal{G}[C_r]$ being the subgraph of $\mathcal{G}$ induced by C_r. Thus, as a byproduct of SCP, the sparsity pattern of matrix A_i is in fact a subgraph of $\mathcal{G}$. That is, SCP will not complicate the sparsity pattern of subproblems in the branch-and-bound tree.

Remark 4.1 We would like to point out that the redundancy of a variable X_{ij} in (CSDR) does not indicate that all triangle inequalities in (3) involving X_{ij} are ineffective in improving the tightness of (SDR). The effect of these triangle inequalities depends on the structure of the problem. More discussions on the effectiveness of the SCP and TCP schemes will be provided in Section 6.

5 A new branch-and-bound algorithm

In this section, we describe the main steps of the proposed semidefinite relaxation based branch-and-bound algorithm. The main framework of the proposed algorithm is similar to the ones in Biq Mac [33], except that the proposed one uses the new branching rule and cutting-plane selection scheme described in Subsections 4.2 and 4.3, respectively. The main steps and some implementation details of the proposed algorithm are provided as follows.

Preprocessing: Before solving the problem, we first construct a graph to represent the sparsity pattern of the matrix Q, and expand the graph to a

chordal graph by using the greedy fill-in heuristic chordal extension algorithm in [24]. The expanded graph is denoted as $\mathcal{G}$. We also obtain the clique decomposition of $\mathcal{G}$ by using Algorithm 3.1 in [38, Section 3.5]. The details of Algorithm 3.1 can be referred to [38, Section 4].

Problem enumeration procedure: The problem being enumerated by the branch-and-bound algorithm is represented by a tree. In each enumeration step, we choose the leaf-node in the tree that has the largest upper bound. If there are multiple leaf-node that achieve the same largest upper bound, we break the tie arbitrary.

Iterative upper bound procedure: We solve the relaxation (SDR) by Mosek [30], a commercial interior-point based solver. Meanwhile, an iterative cutting-plane method that is similar to the one in [20] is adopted as follows: Valid triangle inequalities are iteratively added into (SDR) after solving the semidefinite relaxation, and then the relaxation with newly added triangle inequalities is re-optimized by Mosek. Following the SCP scheme, we only select the violated triangle inequalities from (4). In our implementation, we add at most 250 valid triangle inequalities in each iteration, and the number of iterations is limited to 30. Moreover, we set an early-stop rule to terminate the iteration if one of the following two conditions holds: (i) Let ℓ^* be the best-known lower bound and u_j be the upper bound obtained in the j-th iteration. Under the assumption that the entries in Q are all integers, the iteration terminates immediately when $u_j < \ell^* + 0.99$. (ii) If the improvement in the successive iterations is marginal, then we terminate the iteration. In our implementation, when $\frac{u_j - u_{j+1}}{u_0 - \ell^*} < 0.1$ for some j, the iteration terminates, where u_0 is the upper bound obtained by solving the initial relaxation (SDR) without adding any triangle inequality.

Lower bound procedure: For each enumeration node, after obtaining the optimal solution of relaxation (SDR), we use the classical semidefinite relaxation based rounding scheme in [16] to generate a feasible solution of problem (MC). The best-known feasible solution is recorded, and its corresponding objective value ℓ^* serves as a global lower bound of problem (MC).

Branching rule: We compute the priority value P_{ij} for each $[i, j] \in \mathcal{E}$, and select the variable X_{ij} to branch following the rule HB described at the end of Subsection 4.2.

We would like to point out that the framework of the proposed branch-and-bound algorithm is flexible. For example, the SCP scheme can be replaced by TCP scheme in the iterative upper bound procedure, and other branching rules, such as rule R2, instead of rule HB, can be applied in branching step.

Remark 5.1 Since it is not always more efficient to solve (CSDR) than (SDR) by using an interior-point based solver when the maximal cliques in the clique decomposition have large intersections with each other, we use (SDR) rather than (CSDR) for computing the upper bound. In fact, we introduce (CSDR) for analyzing its connection to (PR) and explaining how the chordal sparsity pattern affects the tightness of (SDR), rather than for computational purpose.

6 Computational experiments

In this section, we test the proposed algorithm with different options on randomly generated instances. The algorithm is implemented using Matlab R2017a on a personal computer with Intel Core i7-9700 CPU and 16 GB RAM. We adopt the commercial solver Mosek [30] (version 9.2) to solve the semidefinite relaxations in the proposed algorithm. We first describe different option settings of the proposed algorithm, and the data sets used in our experiment, and then present and analyze the numerical results. More experiments on larger instances and discussions on the feature of the proposed algorithm are presented at the end of this section.

6.1 Experiment settings and data sets

To study the effects of rule HB and SCP scheme, we implemented the proposed algorithm with three different variants as follows.
HB-SCP algorithm. This is the exact algorithm described in Section 5, which applies both of rule HB and SCP scheme.
HB algorithm. This algorithm uses TCP, instead of SCP, scheme in the upper bound procedure. The purpose here is to study the effectiveness of SCP scheme.
R2 algorithm. This algorithm turns off both rule HB and SCP scheme, but applies rule R2 and TCP scheme. This option is similar to Biq Mac [33], except that we solve the semidefinite relaxations using Mosek while Biq Mac uses the conic bundle algorithm.

We have also implemented another variant called "R3 algorithm", which uses rule R3 as the branching rule. However, our preliminary numerical results show that this variant performed worse than the R2 algorithm. This phenomenon has also been observed in other works, such as [20, 25] and [33]. Hence, we use R2 algorithm as a benchmark to study the effectiveness of rule HB and SCP scheme. Since all three algorithms use the same solver and run on the same computer, the comparison between these three algorithms is fair, and can reflect the effects of rule HB and SCP scheme.

Moreover, we have tested BiqCrunch in our experiment. BiqCrunch (the second release) [26] is an improved version of the branch-and-bound algorithm proposed in [25] that has achieved the state-of-the-art performance on solving binary quadratic programming problems. BiqCrunch applies a BFGS algorithm to solve the semidefinite relaxation, the main procedures are implemented in C, and some sub-procedures (the steps for BFGS iterations) are implemented in Fortran. As far as we know, BiqCrunch is currently the best semidefinite relaxation based branch-and-bound solver for solving the max-cut problem. It outperforms Biq Mac on many binary quadratic programming test

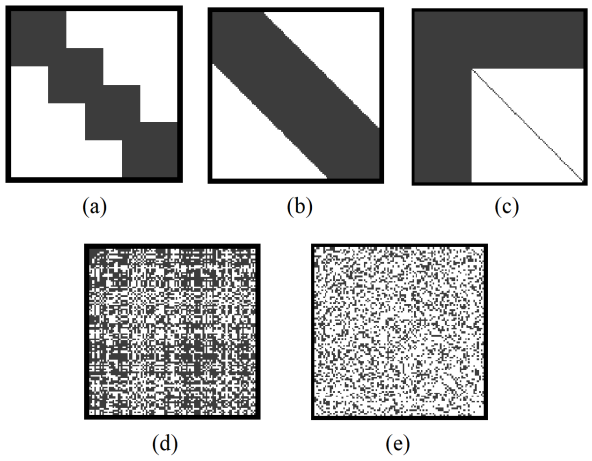

Fig. 2 Illustrations of five sparsity patterns: (a) overlapping block diagonal; (b) banded; (c) block arrow; (d) disk graph; (e) generic random graph.

instances[2]. The code of BiqCrunch[3] was compiled by GCC compiler and ran on the same computer.

Since most of the public benchmark test sets, such as Biq Mac Library[4], do not have instances with any chordal sparsity pattern, we generate random instances of five different sparsity patterns with wide applications for our experiment.

Set A: Ten instances with the *overlapping block diagonal* sparsity pattern are randomly generated in this set. In this sparsity pattern, there are k blocks, each of which contains w fully connected vertices. The two adjacent blocks are overlapped, with s vertices in their intersection. See Figure 2(a) for an illustration. The vertices in the overlapping block diagonal sparsity pattern is local-clustered and graph-connected. Such a graph often appears in a social network, where each vertex is an individual and an edge represents the relation between two individuals. The individuals in a common community are connected with each other (locally clustered) while individuals in different communities are connected via others [5]. In our experiment, we set $k = 4$, $w = 30$ and $s = 10$. The generated instances are indexed by "Block_w_s_k_id", where id$\in \{1, \ldots, 10\}$ denotes the index of the instance. Given the sparsity pattern with the above parameters, the graph has 90 vertices and $1,695$ edges. The density of the graph is 42.3%.

[2] See https://biqcrunch.lipn.univ-paris13.fr/BiqCrunch/results for detailed numerical results of BiqCrunch and Biq Mac.

[3] Available at `https://biqcrunch.lipn.univ-paris13.fr/BiqCrunch`. Our results is based on the second release of BiqCrunch.

[4] See `http://biqmac.uni-klu.ac.at/biqmaclib.html`.

Set B: Ten instances with the *banded* sparsity pattern are generated in this set. The banded sparsity often appears in the finite-difference and finite-element models for various practical problems [17,37]. In this sparsity pattern, two vertices $i, j \in \{1, \ldots, n\}$ are adjacent if and only if $|i - j| \leq w$, where w is a positive integer that denotes the bandwidth. In our experiment, we set $n = 100$ and $w = 30$. The generated instances are indexed by "Band_n_w_id". Each graph in this set has 100 vertices and $2,635$ edges with density being 53.2%.
Set C: Ten instances with the *block-arrow* sparsity pattern are generated in this set. The matrices of block-arrow sparsity pattern appear in Statistics [1] and power network [35]. In this sparsity pattern, there are n vertices in total, and the first k vertices are connected with each other while any vertex $i \in \{k+1, \ldots, n\}$ is adjacent to the first k vertices. We set $n = 120$ and $k = 40$ in our experiment. The generated instances are indexed by "Arrow_n_k_id". Each graph in this set has 120 vertices and $4,100$ edges with density being 57.4%.
Set D: Ten instances with the sparsity pattern derived from a *disk graph* are generated in this set. The disk graph appears in communication networks [10]. For each instance, we first generate a random disk graph following the procedure in [10, Section 5.1]: We sample n points $p_1, \ldots, p_n \in \mathbb{R}^2$ uniformly on the unit square. Edges are created between pairs of vertices i and j if $\|p_i - p_j\| \leq d$, where $d > 0$ is a constant parameter. The graph is then expanded with edges between pairs of vertices which have a neighbor in common. We set $n = 100$ and $d = 0.3$. The generated instances are indexed by "DiskGraph_n_id". Different from the previous three sparsity patterns that have deterministic structures in the positions of nonzero elements, the number and positions of nonzero elements in the disk-graph sparsity pattern are random. Hence, the density of different instances varies. Our Monte Carlo simulation shows that the expected density of the instances in this set is about 56.8%.
Set E: Ten instances with the sparsity pattern derived from a *random graph* are generated in this set. In a random graph with n vertices, each pair of vertices are adjacent with probability p. This type of sparsity pattern is a generic random sparsity pattern in which it does not have any deterministic structure. In our experiment, we set $n = 100$ and $p = 0.3$. The generated instances are indexed by "Rand_n_P_id", where $P = 100p$. The expected number of edges in such a graph is $1,485$, and the density is expected to be 30%.

Please refer to Figure 2 for an illustration of the five sparsity patterns. In addition, we generate another set of test instances by using RUDY [34] as follows.
Set F: Ten instances with the random-graph sparsity are generated in this set. The random graphs are generated by issuing the command "rudy -rnd_graph n d s1 -random -50 50 s2" in RUDY, where $n = 120$ is the number of vertices, $d \in \{20, 50\}$ represents the density, and s1 and s2 are two seeds that are both equal to id. Note that the first five instances in this set have a density of 20%, and the other five instances have a density of 50%. The generated instances are indexed by "Rudy_n_d_id".

For every instance with a given sparsity pattern $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we uniformly sample the entry Q_{ij} of the symmetric matrix Q from $\{-50, -49, \ldots, 50\}$ for each edge $[i, j] \in \mathcal{E}$. In total, we have 60 test instances, whose size ranges from 90 to 120 and density from 20% to 57.4%. The sparsity patterns in Sets A-C have the deterministic structure. The aim of the experiments on these instances is to test the proposed algorithms on problems with special structures arisen in various practical applications. The sparsity patterns in Sets D-F are derived from random graphs. The sparsity pattern for Set D contains some geometrical information: the vertices are defined by a set of points in the unit square, and a point is only connected with its nearby points. In comparison, the sparsity patterns for Sets E and F are derived from generic random graphs that do not have any structural information. By evaluating the performance of the proposed algorithm on these random sparsity patterns, we could assess whether the proposed algorithm remains to be efficient on general sparsity patterns.

6.2 Numerical results

The numerical results for Sets A-F are reported in Tables 1–6, respectively. All computational time is the wall clock time in seconds. The computation time for both HB-SCP algorithm and HB algorithm includes the time for chordal extension, clique decomposition and the branch-and-bound procedure. Since the time for applying chordal extension and clique decomposition is much less than the one for branch-and-bound procedure, we do not show them separately in the tables.

First, we evaluate the effectiveness of SCP scheme by comparing the results of HB-SCP and HB algorithms. The computational time in Tables 1 and 4 shows that HB-SCP runs faster than HB on all instances in Set A, and slightly faster on 9 out of 10 instances in Set D. However, Tables 2, 3, 5 and 6 show that HB-SCP performs worse than HB on almost all instances in Sets B, C, E and F. Based on this observation, we could conclude that the effectiveness of SCP scheme depends on the sparsity pattern. An intuitive reason for the above observation is that the sparsity patterns in Sets A and D are locally clustered while they are not in other sets. For example, in the overlapping block diagonal sparsity pattern in Set A, the vertices in the first block are not adjacent with the vertices in the third or fourth blocks. Specifically, for any two vertices i and j that are located in the first and fourth block, respectively, they do not have a common neighbor vertex. Therefore, the interaction between the two vertices is extremely weak. Adding valid triangle inequalities on vertices with weak interaction may lead to marginal improvement in the upper bound. Similarly, for the sparsity pattern in set D, a vertex in a disk graph is only adjacent to a small number of nearby vertices. On the other hand, the SCP scheme is less effective for the instances in Sets B, C, E and F because the vertices in these sets have strong connections with each other. Consider the block-arrow sparsity pattern in set C, each pair of vertices i and j in the

Table 1 Numerical results for Set A (overlapping block diagonal sparsity pattern).

Problem	Nodes Enumerated				Time (Seconds)			
Name	HB-SCP	HB	R2	BiqC	HB-SCP	HB	R2	BiqC
Block_30_10_4_1	9	11	23	11	**9.06**	13.17	35.56	33.69
Block_30_10_4_2	7	9	23	9	**9.16**	23.06	60.68	39.00
Block_30_10_4_3	9	17	63	13	**7.22**	18.94	104.70	40.82
Block_30_10_4_4	7	7	67	7	**8.42**	14.44	129.97	29.63
Block_30_10_4_5	7	11	79	11	**6.56**	13.42	99.02	43.34
Block_30_10_4_6	5	5	5	1	2.99	4.75	6.10	**1.82**
Block_30_10_4_7	3	5	5	1	1.14	2.65	2.99	**1.12**
Block_30_10_4_8	35	35	81	35	**35.08**	52.32	95.67	101.85
Block_30_10_4_9	9	11	55	15	**12.31**	16.03	73.48	42.98
Block_30_10_4_10	19	21	61	15	**15.00**	30.93	101.36	45.20

Table 2 Numerical results for Set B (banded sparsity pattern).

Problem	Nodes Enumerated				Time (Seconds)			
Name	HB-SCP	HB	R2	BiqC	HB-SCP	HB	R2	BiqC
Band_100_30_1	15	15	23	15	24.05	**22.06**	44.77	55.89
Band_100_30_2	53	35	75	35	76.25	**64.52**	138.08	105.15
Band_100_30_3	101	59	137	101	128.63	**124.79**	205.64	288.17
Band_100_30_4	91	71	105	59	126.38	**109.35**	175.03	185.00
Band_100_30_5	5	7	5	3	**5.31**	8.73	7.65	9.43
Band_100_30_6	35	23	51	29	45.04	**33.52**	106.22	98.16
Band_100_30_7	239	147	317	183	318.45	**254.93**	463.96	490.67
Band_100_30_8	201	131	343	135	**300.60**	326.20	494.97	423.68
Band_100_30_9	201	123	251	133	277.07	**236.25**	397.95	370.13
Band_100_30_10	3	7	9	1	3.12	12.17	7.04	**1.46**

Table 3 Numerical results for Set C (block-arrow sparsity pattern).

Problem	Nodes Enumerated				Time (Seconds)			
Name	HB-SCP	HB	R2	BiqC	HB-SCP	HB	R2	BiqC
Arrow_40_120_1	735	181	1145	1755	1073.92	**497.17**	2592.19	4729.22
Arrow_40_120_2	625	145	1073	1227	878.24	**400.58**	2303.16	3409.52
Arrow_40_120_3	1055	211	2295	1839	1278.67	**558.61**	4764.95	4373.20
Arrow_40_120_4	709	139	1047	869	947.14	**427.30**	2222.04	2620.58
Arrow_40_120_5	181	53	285	307	270.43	**126.64**	643.87	922.64
Arrow_40_120_6	157	43	199	175	272.65	**162.92**	501.30	684.51
Arrow_40_120_7	429	87	627	815	638.44	**282.98**	1630.72	2857.61
Arrow_40_120_8	1279	291	—	—	1719.55	**867.58**	—	—
Arrow_40_120_9	989	219	2159	1453	1419.63	**597.99**	4956.09	4370.79
Arrow_40_120_10	615	149	731	677	934.17	**423.07**	1648.75	2227.66

Note: "—" means that the instance is not solved in 5,400 seconds.

graph is either adjacent, or has at least 40 common neighbor vertices. Based on the above analysis, we may conclude that the SCP scheme is effective on instances whose sparsity pattern is similar to a locally clustering graph. This is true when the intersections between different pairs of maximal cliques in a tree decomposition have only a few vertices. On the other hand, if the intersections

Table 4 Numerical results for Set D (disk-graph sparsity pattern).

Problem	Nodes Enumerated				Time (Seconds)			
Name	HB-SCP	HB	R2	BiqC	HB-SCP	HB	R2	BiqC
DiskGraph_100_1	157	113	511	217	**238.56**	267.98	724.47	672.88
DiskGraph_100_2	27	25	57	29	**43.00**	59.54	103.06	110.32
DiskGraph_100_3	101	81	149	71	**171.26**	192.91	232.24	236.49
DiskGraph_100_4	55	47	137	65	**94.93**	104.61	181.10	248.73
DiskGraph_100_5	909	549	921	633	**1090.99**	1178.79	1383.98	1811.57
DiskGraph_100_6	47	39	109	45	73.53	**64.49**	167.07	165.76
DiskGraph_100_7	113	87	147	99	**161.57**	190.39	211.00	329.68
DiskGraph_100_8	49	33	91	63	**69.00**	77.71	140.31	223.34
DiskGraph_100_9	199	155	489	311	**256.62**	258.00	593.79	898.29
DiskGraph_100_10	123	101	239	165	**153.02**	168.94	309.67	455.01

Table 5 Numerical results for Set E (generic random-graph sparsity pattern).

Problem	Nodes Enumerated				Time (Seconds)			
Name	HB-SCP	HB	R2	BiqC	HB-SCP	HB	R2	BiqC
Rand_100_30_1	31	21	31	15	46.04	47.26	42.06	**39.02**
Rand_100_30_2	39	27	47	25	63.41	**51.28**	79.15	68.67
Rand_100_30_3	49	31	71	98.31	87.49	**65.65**	97.08	98.31
Rand_100_30_4	59	47	97	61	86.87	**63.93**	133.55	138.87
Rand_100_30_5	143	91	181	107	**208.80**	213.42	254.34	265.01
Rand_100_30_6	37	27	45	31	66.12	**39.25**	83.31	94.77
Rand_100_30_7	137	99	135	127	210.52	**172.90**	201.38	317.68
Rand_100_30_8	107	79	117	67	173.05	**155.23**	162.35	203.78
Rand_100_30_9	381	253	697	327	598.83	**383.47**	894.54	695.65
Rand_100_30_10	97	73	105	63	139.63	**126.93**	178.03	172.06

Table 6 Numerical results for Set F (random-graph sparsity pattern generated by RUDY).

Problem	Nodes Enumerated				Time (Seconds)			
Name	HB-SCP	HB	R2	BiqC	HB-SCP	HB	R2	BiqC
Rudy_120_20_0	161	87	107	109	343.61	**193.90**	268.24	329.62
Rudy_120_20_1	205	101	173	85	613.53	381.47	**319.17**	386.47
Rudy_120_20_2	121	65	101	77	308.05	**201.25**	235.48	305.94
Rudy_120_20_3	307	149	231	113	658.90	456.95	518.60	**361.18**
Rudy_120_20_4	829	381	1371	581	1646.84	**1142.69**	2962.26	1631.47
Rudy_120_50_0	311	269	411	267	855.52	**830.62**	886.35	898.62
Rudy_120_50_1	423	349	571	395	1497.86	1494.57	**1185.30**	1694.03
Rudy_120_50_2	405	343	381	323	1022.64	971.03	**856.76**	1126.33
Rudy_120_50_3	95	75	103	65	241.70	220.18	**203.93**	219.56
Rudy_120_50_4	—	—	—	—	—	—	—	—

Note: "—" means that the instance is not solved in 5,400 seconds.

between different pairs of maximal cliques have many vertices, then the SCP scheme may filter some promising valid triangle inequalities, thus lead to only a small improvement in upper bounds.

Second, we study the effects of the new branching rule by comparing the HB algorithm with R2 algorithm. Note that both HB and R2 algorithms use

the same upper bound scheme, the only difference between the two algorithms is the branching rule. Hence, the comparison shows the effects of different branching rules. We first analyze the results for Sets A-C, whose instances have deterministic sparsity patterns. From Tables 1-3, we can see that HB runs faster than R2 on most test instances, except two easy instances in Set B that can be solved within no more than 10 enumerations. Especially, HB runs at least five times faster than R2 for the instances with the block-arrow sparsity pattern in Set C. These results show that, by exploiting the clique decomposition of sparsity pattern with a deterministic structure, the new branching rule could be very effective in improving the overall performance of a branch-and-bound algorithm. We then analyze the results of Sets D-F, whose sparsity patterns are random. In Table 4, we discover that HB performs better than R2 on all test instances, and the dominance is significant on several instances (including instances 1, 9 and 10 in set D). This result shows that rule HB is effective for test instances with disk-graph sparsity pattern. For the instances in Sets E and F, we observe from Tables 5 and 6 that HB runs faster than R2 on 9 out of 10 test instances in Set E, and 4 out of 5 instances with density 20% in Set F. This result indicates that rule HB is effective when the density of the random graph is relatively low. However, as shown in Table 6, HB performs worse than R2 for most of the instances with a density of 50%. The reason is that when the density of a random graph becomes high, after the chordal extension, the graph is close to a complete graph, and rule HB becomes almost equivalent to rule R3 in this case. Since R3 performs worse than R2 in general, the proposed rule HB is dominant by rule R2 for the dense random sparsity patterns. Lastly, for all instances in Tables 1-6, HB enumerates fewer nodes than R2. This is reasonable because rule HB makes the sparsity pattern in the child nodes more likely to satisfy the sufficient conditions in Theorem 3.1 and Corollary 3.1. Consequently, the bounds of child nodes under rule HB would be more possible to be tight, thus be pruned earlier in the HB.

Finally, we compare HB-SCP and HB with BiqCrunch. By comparing the computational time in Tables 1-6, we discover that HB-SCP might be the best choice for Sets A and D, and HB performs best on most test instances in Sets B, C and E, and the instances with density 20% in Set F. Since HB-SCP, HB and BiqCrunch use different methods to solve their semidefinite relaxations, and these algorithms are implemented in different programming language, it might be hard to conclude that our new branching rule is the key factor that leads to the better performance. However, by comparing the computational time of the proposed algorithms with BiqCrunch, we may conclude that HB-SCP and HB algorithms do perform better than BiqCrunch on randomly generated test instances that have certain types of sparsity patterns.

Remark 6.1 Although we listed the number of nodes enumerated by all algorithms in the tables, it is not a major indicator of algorithmic performance in this case. It is quite possible that an algorithm runs faster, but needs more enumerations than another algorithm, because the time for computing upper bounds depends on the number of iterations when using the iterative upper

bound scheme. One branching rule may generate nodes that can be easily pruned while others branching rules lead to vast nodes to explore. Therefore, the number of nodes does not reflect the overall performance of the proposed algorithm. Instead, we compare the computational time, which is directly related to the efficiency of every algorithm.

6.3 Larger instances with sparsity patterns

According to the numerical results in Subsection 6.2, we see that HB-SCP algorithm performs best for the banded and block-arrow sparsity patterns, and HB algorithm is the best one for the overlapping block diagonal and disk-graph sparsity patterns. In this subsection, we further evaluate the proposed algorithms by stress testing them on "hard" instances with the above four sparsity patterns. That is, we generate largest random instances that can be solved by HB-SCP or HB within 3 hours. The two sets of instances used in this subsection are as follows.

Set G: This set includes five instances of the banded sparsity pattern and five instances of the block-arrow sparsity pattern. For the instances of banded sparsity pattern indexed by "Band_n_w_id", we set $n = 120$ and $w = 40$. For the instances of block-arrow sparsity pattern indexed by "Arrow_n_k_id", we set $n = 180$ and $k = 5$. This set is used for testing algorithms HB, R2 and BiqCrunch.

Set H: This set includes five instances of the overlapping block diagonal sparsity pattern and five instances of the disk-graph sparsity pattern. For the instances of overlapping block diagonal sparsity pattern that are indexed by "Block_w_s_k_id", we set $w = 30$, $s = 10$ and $k = 8$.[5] For the instances of disk-graph sparsity pattern indexed by "DiskGraph_n_id", we set $n = 160$.[6] This set is used for testing algorithms HB-TIP, R2 and BiqCrunch.

The numerical results for Sets G and H are reported in Tables 7 and 8, respectively. We have the following two observations in this experiment:

(i) Same as the results in Tables 1-4, HB and HB-SCP are still more efficient than R2 and BiqCrunch on all, except two, test instances. Especially for the block-arrow sparsity pattern, HB solves all the five instances while BiqCrunch fails to solve any.

(ii) The efficiency of the proposed algorithms strongly depends on the types of sparsity patterns. The same observation also holds for BiqCrunch.

[5] We generated instances with different $k = 5, 6, 7, 8$ for the given w and s, and found that the proposed algorithm can solve the largest instance for $k = 8$ within 7 minutes, while the benchmark algorithm R2 already ran out of time limit. Hence, we did not try to find the largest possible instances that can be solved within 3 hours by our algorithm for this type of sparsity pattern.

[6] We found that, even for $n = 120$, some instances can not be solved by any of the algorithm within 3 hours. Hence, the step of graph augmentation is skipped in generating disk graphs in this set.

Table 7 Numerical results for Set G.

Problem	Nodes Enumerated			Time (Seconds)		
Name	HB	R2	BiqC	HB	R2	BiqC
Band_120_40_1	709	1283	935	**2559.75**	2757.74	3654.07
Band_120_40_2	43	133	99	**164.86**	245.62	533.10
Band_120_40_3	1607	2755	1791	**3743.70**	5437.97	5282.84
Band_120_40_4	199	385	343	**728.31**	848.29	1401.64
Band_120_40_5	53	103	65	**156.41**	261.33	287.43
Arrow_180_50_1	1049	—	—	**4595.36**	—	—
Arrow_180_50_2	173	731	—	**757.06**	2585.72	—
Arrow_180_50_3	75	281	—	**406.25**	1059.44	—
Arrow_180_50_4	1777	—	—	**9290.92**	—	—
Arrow_180_50_5	225	1059	—	**1266.86**	3860.91	—

Note: "—" means that the instance is not solved in 10,800 seconds.

Table 8 Numerical results for Set H.

Problem	Nodes Enumerated			Time (Seconds)		
Name	HB-SCP	R2	BiqC	HB-SCP	R2	BiqC
Block_30_10_8_1	15	377	21	**61.01**	1399.29	254.02
Block_30_10_8_2	111	—	139	**365.24**	—	1307.81
Block_30_10_8_3	75	—	93	**352.31**	—	957.70
Block_30_10_8_4	45	1275	65	**234.05**	5075.66	684.36
Block_30_10_8_5	13	231	3	47.65	823.45	**42.14**
DiskGraph_160_1	127	379	93	**676.65**	1774.17	766.25
DiskGraph_160_2	955	967	343	4161.82	4053.51	**2775.07**
DiskGraph_160_3	1417	—	1205	**6093.02**	—	8019.06
DiskGraph_160_4	337	1021	311	**1466.23**	4372.86	2370.96
DiskGraph_160_5	177	913	111	**742.03**	4042.55	933.00

Note: "—" means that the instance is not solved in 10,800 seconds.

The reason for the second observation deserves a further explanation. For the instances of banded and block-arrow sparsity patterns, Table 7 shows that HB can only solve up to 120 and 180 dimensions within 3 hours, respectively. In contrast, HB-SCP can easily solve the 180-dimensional instances of overlapping block diagonal sparsity pattern within 7 minutes as shown in Table 8. Similarly, BiqCrunch solves the instances of overlapping block diagonal sparsity pattern in at most 1,308 seconds but fails to solve any instance of block-arrow sparsity pattern within 3 hours. By comparing the structure of the above three sparsity patterns, we can see that the overlapping block diagonal sparsity pattern has two distinct features: the size of maximal clique is small (i.e., small cluster) and the intersection set between every pair of maximal clique has only a few vertices (i.e., small cluster intersection). These two features make the sufficient conditions in Theorem 3.1 and Corollary 3.1 easy to be satisfied, thus the semidefinite relaxations provide tight bounds for problem (MC). In this case, both HB-SCP and BiqCrunch are efficient to solve the instances of overlapping block diagonal sparsity pattern. In comparison, for the banded and block-arrow sparsity patterns, the intersection sets between two cliques could have many vertices (up to the same size of the treewidth), thus the instances of these two sparsity patterns are relatively hard to solve for those algorithms.

Moreover, for the block-arrow sparsity pattern, we discover that all the intersection sets of any two different maximal cliques are the same, i.e., all the maximal cliques share the common intersection set. Then, according to the rule HB, the edge that connects two vertices in this common intersection set has a high priority. By contracting the two vertices in the common intersection set, all the maximal cliques will have one less vertex in the child nodes. Therefore, rule HB reduces the treewidth of the sparsity pattern and the number of vertices in the intersection set. As shown in Table 7, HB successfully solves all the five instances with block-arrow sparsity pattern while BiqCrunch fails to solve any of them within the time limit. In comparison, in the banded sparsity pattern, each pair of maximal cliques has a different intersection set, so the rule HB is not so effective as in the case of block-arrow sparsity pattern.

7 Conclusions

In this paper, we design a new semidefinite relaxation based branch-and-bound algorithm for solving the classic max-cut problem. The main feature of the proposed algorithm lies in the utilization of the chordal sparsity patterns embedded in the max-cut problem. We first develop a polyhedral relaxation from the clique decomposition of the sparsity pattern, and then derive two sufficient conditions for the tightness of the polyhedral relaxation. Then, based on the discussions on the relation between (PR), (CSDR) and (SDR), we investigate how the chordal sparsity pattern affects the tightness of a semidefinite relaxation of the max-cut problem. Motivated by the theoretical results, a hierarchy branching rule is proposed to manipulate the sparsity pattern of the problem in order to reduce the treewidth of the sparsity pattern and the number of elements in the intersections of different cliques.

The computational results show that the proposed hierarchy branching rule is more effective than the conventional numerical-driven rule R2 on test instances with various chordal sparsity patterns. The proposed algorithm also outperforms BiqCrunch in terms of computational time on most instances with sparsity patterns. Meanwhile, the sparsity-pattern driven cutting-plane scheme is effective in improving the overall efficiency of the proposed algorithm for instances with certain types of chordal sparsity patterns. These results indicate that the chordal sparsity pattern of the problem may provide important information that can be utilized to improve the efficiency of a branch-and-bound algorithm.

Acknowledgements Lu's research has been supported by National Natural Science Foundation of China Grant No. 12171151. Deng's research has been supported by a grant from MOE Social Science Laboratory of Digital Economic Forecast and Policy Simulation at UCAS. Fang's research has been supported by the Walter Clark Endowment at NC State. Xing's research has been supported by National Natural Science Foundation of China Grant No. 11771243.

References

1. Andersen, M., Vandenberghe, L., Dahl, J.: Linear matrix inequalities with chordal sparsity patterns and applications to robust quadratic optimization. In: 2010 IEEE International Symposium on Computer-Aided Control System Design (CACSD), pp. 7–12 (2010)
2. Arnborg, S., Corneil, D., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM. J. Alg. Disc. Meth. **8**(2), 277–284 (1987)
3. Barahona, F., Grötschel, M., Mahjoub, A.R.: Facets of the bipartite subgraph polytope. Math. Oper. Res. **10**(2), 340–358 (1985)
4. Barahona, F., Mahjoub, A.R.: On the cut polytope. Math. Program. Ser. A **36**(2), 157–173 (1986)
5. Benati, S., Ponce, D., Puerto, J., Rodriguez-Chia, A.: A branch-and-price procedure for clustering data that are graph connected. Euro. J. Oper. Res. **297**, 817–830 (2021)
6. Billionnet, A., Elloumi, S.: Using a mixed integer quadratic programming solver for the unconstrained quadratic 0-1 problem. Math. Program. Ser. A **109**(1), 55–68 (2007)
7. Burer, S., Monteiro, R.D.C., Zhang, Y.: Rank-two relaxation heuristics for max-cut and other binary quadratic programs. SIAM J. Optim. **12**(2), 503–521 (2002)
8. Diestel, R.: Graph Theory (5th ed.). Springer (2017)
9. Dunning, I., Gupta, S., Silberholz, J.: What works best when? A systematic evaluation of heuristics for max-cut and QUBO. INFORMS J. Comput. **30**(3), 608–624 (2018)
10. Fairbrother, J., Letchford, A.N., Briggs, K.: A two-level graph partitioning problem arising in mobile wireless communications. Comput. Optim. Appl. **69**(3), 653–676 (2018)
11. Festa, P., Pardalos, P., Resende, M., Ribeiro, C.: Randomized heuristics for the max-cut problem. Optim. Methods Softw. **17**(6), 1033–1058 (2002)
12. Fukuda, M., Kojima, M., Murota, K., Nakata, K.: Exploiting sparsity in semidefinite programming via matrix completion I: General framework. SIAM J. Optim. **11**(3), 647–674 (2001)
13. Gaar, E., Rendl, F.: A computational study of exact subgraph based SDP bounds for Max-Cut, stable set and coloring. Math. Program. Ser. B **183**, 283–308 (2020)
14. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)
15. Garstka, M., Cannon, M., Goulart, P.: Cosmo: A conic operator splitting method for convex conic problems. J. Optimiz. Theory App. **190**, 779–810 (2021)
16. Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J. ACM **42**(6), 1115–1145 (1995)
17. Gosz, M.: Finite Element Method: Applications in Solids, Structures, and Heat Transfer. CRC Press, Boca Raton (2017)
18. Grone, R., Johnson, C.R., Sá, E.M., Wolkowicz, H.: Positive definite completions of partial Hermitian matrices. Linear Algebra Appl. **58**, 109–124 (1984)
19. Heggernes, P.: Minimal triangulation of graphs: A survey. Discrete Math. **306**(3), 297–317 (2006)
20. Helmberg, C., Rendl, F.: Solving quadratic (0,1)-problems by semidefinite programs and cutting planes. Math. Program. Ser. A **82**(3), 291–315 (1998)
21. Jarre, F., Lieder, F., Liu, Y., Lu, C.: Set-completely-positive representations and cuts for the max-cut polytope and the unit modulus lifting. J. Glob. Optim. **76**, 913–932 (2020). DOI 10.1007/s10898-019-00813-x
22. Jünger, M., Mallach, S.: Exact facetial odd-cycle separation for maximum cut and binary quadratic optimization. INFORMS J. Comput. **33**(4), 1419–1430 (2021)
23. Kim, S., Kojima, M.: Second order cone programming relaxation of nonconvex quadratic optimization problems. Optim. Methods Softw. **15**(3-4), 201–224 (2001)
24. Koster, A.C., Bodlaender, H.L., van Hoesel, S.P.: Treewidth: computational experiments. Electron. Notes Discrete Math. **8**, 54–57 (2001)
25. Krislock, N., Malick, J., Roupin, F.: Improved semidefinite bounding procedure for solving max-cut problems to optimality. Math. Program. Ser. A **143**(1), 62–86 (2014)
26. Krislock, N., Malick, J., Roupin, F.: Biqcrunch: a semidefinite branch-and-bound method for solving binary quadratic problems. ACM T. Math. Softw. **43**(4), 1–23 (2017)

27. Lasserre, J.B.: A MAX-CUT formulation of 0/1 programs. Oper. Res. Lett. **44**, 158–164 (2016)
28. Madani, R., Sojoudi, S., Fazelnia, G., Lavaei, J.: Finding low-rank solutions of sparse linear matrix inequalities using convex optimization. SIAM J. Optim. **27**(2), 725–758 (2017)
29. Martí, R., Duarte, A., Laguna, M.: Advanced scatter search for the max-cut problem. INFORMS J. Comput. **21**(1), 26–38 (2009). DOI 10.1287/ijoc.1080.0275
30. Mosek: Mosek aps. `http://www.mosek.com` (2020)
31. Muramatsu, M., Suzuki, T.: A new second-order cone programming relaxation for max-cut problems. J. Oper. Res. Soc. Jpn. **46**(2), 164–177 (2003)
32. Poljak, S., Rendl, F.: Solving the max-cut problem using eigenvalues. Discrete Appl. Math. **62**(1-3), 249–278 (1995)
33. Rendl, F., Rinaldi, G., Wiegele, A.: Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations. Math. Program. Ser. A **121**, 307–335 (2010)
34. Rinaldi, G.: Rudy. `http://www-user.tu-chemnitz.de/` (1998)
35. Sliwak, J., Andersen, E.D., Anjos, M.F., Létocart, L., Traversi, E.: A clique merging algorithm to solve semidefinite relaxations of optimal power flow problems. IEEE T. Power Syst. **36**(2), 1641–1644 (2021). DOI 10.1109/TPWRS.2020.3044501
36. Tarjan, R., Yannakakis, M.: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. SIAM J. Comput. **13**(3), 566–579 (1984)
37. Teng, J., Jakeman, A., Vaze, J., Croke, B., Dutta, D., Kim, S.: Flood inundation modelling: A review of methods, recent advances and uncertainty analysis. Environ. Modell. Softw. **90**, 201–216 (2017)
38. Vandenberghe, L., Andersen, M.S.: Chordal graphs and semidefinite optimization. Found. Trends Optim. **1**(4), 241–443 (2014)
39. Waki, H., Kim, S., Kojima, M., Muramatsu, M.: Sums of squares and semidefinite program relaxations for polynomial optimization problems with structured sparsity. SIAM J. Optim. **17**(1), 218–242 (2006)
40. Wang, J., Magron, V.: Exploiting sparsity in complex polynomial optimization. J Optimiz. Theory App. **192**, 335–359 (2022)
41. Wang, J., Magron, V., Lasserre, J.B.: Chordal-TSSOS: A moment-SOS hierarchy that exploits term sparsity with chordal extension. SIAM J. Optim. **31**(1), 114–141 (2021)
42. Wang, J., Magron, V., Lasserre, J.B., Mai, N.H.A.: CS-TSSOS: Correlative and term sparsity for large scale polynomial optimization. Arixv:2005.02828 (2021)
43. Zhang, R.Y., Lavaei, J.: Sparse semidefinite programs with guaranteed near-linear time complexity via dualized clique tree conversion. Math. Program. Ser. A **188**(1), 351–393 (2021)
44. Zheng, Y., Fantuzzi, G., Papachristodoulou, A., Goulart, P., Wynn, A.: Chordal decomposition in operator-splitting methods for sparse semidefinite programs. Math. Program. Ser. A **180**(1), 489–532 (2020)