On Spanning-Tree Integrality and a new Branching Rule for the Maximum Cut Problem

Chris Cohadari[2] and Sven Mallach[1,2]

[1]Network & Data Science Management Group, University of Siegen, Germany
[2]University of Bonn, Germany

September 17, 2026

Abstract

State-of-the-art exact methods for the Maximum Cut problem are based on solving linear and semidefinite programming relaxations embedded into a branch-and-bound algorithm. For linear programming formulations, it was shown recently that it is thereby sufficient to enforce the integrality of the variables associated with the edges of a spanning tree. Our first contribution is to extend this result to the domain of semidefinite programming. We then present a new branching rule that is designed to exploit this theoretical foundation in practice. In a computational study, we demonstrate significant improvements compared to standard branching rules used in state-of-the-art solvers employing both kinds of relaxations.

1 Introduction

The Maximum Cut Problem (MaxCut) is a classical and well-studied combinatorial optimization problem: Given an undirected graph $G := (V,E)$ with edge weights $c : E \to \mathbb{R}$, determine a cut of G, i.e., an edge set $\delta(W) := \{\{u,v\} : u \in W, v \in V \setminus W\}$ for $W \subseteq V$, such that its value $c(\delta(W)) := \sum_{e\in\delta(W)} c_e$ is as large as possible. Besides several optimization problems that translate to MaxCut in a straightforward fashion [10], there is a long list of applications [14] that arise from its direct correspondence to Quadratic Unconstrained Binary Optimization (QUBO) [3, 11, 16]. At the same time, MaxCut is $\mathcal{NP}$-hard in the general case [21].

State-of-the-art exact methods to solve MaxCut are based on linear programming (LP) [9, 27] and semidefinite programming (SDP) [28, 23, 15, 17, 8] formulations and have evolved with several recent advancements. These formulations involve binary variables to model whether an edge of G is actually cut ("edge variables"), respectively which of the two partitions of a cut a vertex of G is assigned to ("node variables"). According LP and SDP *relaxations* then first neglect the integrality restriction of these variables, and solving such a relaxation provides an upper bound on the value of a maximum cut of G which is in turn exploited in a branch-and-bound or branch-and-cut fashion to successively re-enforce the integrality of the variables. Thereby, assuming w.l.o.g. that G is connected, it has been shown only recently by Mallach in [26] that it is sufficient to enforce integrality on the edges of a *spanning tree* of G for the canonical LP-based MaxCut formulation by Barahona and Mahjoub [4].

In this paper, we first extend this "spanning-tree integrality" result to the natural SDP relaxation that builds the basis of prevalent SDP-based MaxCut solvers, and thus more tangibly to the rich class of binary quadratic optimization problems that can be modeled by appending further constraints to the underlying SDP formulation. On this theoretical basis, we then design an accordingly inspired and novel branching rule that aims to exploit the sufficiency of spanning-tree integrality in practice, by guiding the actual subproblem generation in a branch-and-bound scheme for both LP- and SDP-based approaches. In an experimental study, we demonstrate sig-

nificant improvements in the required solution time and number of branch-and-bound subproblems until proven optimality when compared to standard and previously proposed branching rules used in state-of-the-art solvers. The results of our work are particularly clear in terms of advancing the state-of-the-art in SDP-based branching. In the case of LP-based branching, our *spanning-tree* rule further improves broadly on the standard rules commonly used for MaxCut formulations that require the integrality of edge variables. Considering also branching rules designed for node-variable branching, there is a more differentiated picture: We observe considerable performance gains and superiority for certain instance classes while rules designed for node-variable branching, in particular the problem-specific *degree-dynamic* rule proposed by Charfreitag et al. in [10], remain more effective on others.

The paper is organized as follows. In Section 2, we review LP- and SDP-based relaxations and branching rules commonly used in state-of-the-art solvers for MaxCut and addressed by this work. Moreover, we briefly recall the recent result on spanning-tree integrality in LP-based formulations that provides the theoretical foundation of our further developments. In Section 3, we then first formally extend this result to the SDP domain, before we derive our actual *spanning-tree* branching rule in Section 4. We report on our computational experiments in Section 5. Finally, a brief conclusion is given in Section 6.

2 Preparations and Related Work

We start by briefly recalling canonical LP- and SDP-based formulations for MaxCut and the result on spanning-tree integrality as shown in [26] in the domain of the former.

In reference to the introduction, we assume to be given an undirected and w.l.o.g. connected graph $G = (V,E)$ along with edge weights $c : E \to \mathbb{R}$, and state first the seminal binary linear programming formulation by Barahona and Mahjoub [4] which we refer to as (MC-ILP).

$$\max \sum_{e \in E} c_e x_e \qquad \text{(MC-ILP)}$$

$$\sum_{e \in S} x_e - \sum_{e \in C \setminus S} x_e \leq |S| - 1 \quad \text{for all cycles } C \subseteq E \text{ and all } S \subseteq C, |S| \text{ odd} \qquad (1)$$

$$0 \leq x_e \leq 1 \quad \text{for all } e \in E \qquad (2)$$

$$x_e \in \{0,1\} \quad \text{for all } e \in E$$

This formulation involves a binary variable x_e for each $e \in E$ that is equal to one if and only if the two endpoints of e are separated by the cut. It is further based on the well-known fact that an edge subset $F \subseteq E$ is a cut in G if and only if F intersects with every cycle in G in an *even* number of edges. This is precisely established by the (only non-trivial) constraints (1), called *odd-cycle inequalities*, which rule out any odd-cardinality intersection.

Now the following theorem is a reformulation (immediate consequence) of Theorem 1 in [26].

Theorem 1. *Let $\bar{x} \in \mathbb{R}^E$ be a solution to the LP relaxation of (MC-ILP), and let $T = (V, E_T)$ be an arbitrary spanning tree of G. If $\bar{x}$ is binary in the components $\bar{x}_e$ for all $e \in E_T$, then $\bar{x}_e \in \{0,1\}$ for the components $e \in E \setminus E_T$ as well and thus $\bar{x}$ is the incidence vector of a cut in G.*

Proof. By Theorem 1 in [26], it is sufficient in (MC-ILP) to explicitly enforce the integrality of the variables x_e that belong to the edges $e \in E_S$ of an arbitrary spanning tree S of G, in the sense that consistent binary values for the other variables (and thus the incidence vector of a cut) are then implied by the odd-cycle inequalities (1), even if these are further restricted to the fundamental cycles of S. Thus, by choosing $S = T$, $\bar{x} \in \{0,1\}^E$ follows directly from the given assumption $\bar{x}_e \in \{0,1\}$ for all $e \in E_T$. □

A common starting point to derive an SDP relaxation for MaxCut is the following problem representation, which we refer to as (MC-ISDP), see also e.g. [28].

$$\max_{X \in S_n} \frac{1}{4} \langle L, X \rangle \qquad \text{(MC-ISDP)}$$

$$\text{s.t. } X_{ii} = 1 \quad \text{for all } i = 1, \ldots, n \qquad (3)$$

$$\text{rank}(X) = 1 \qquad (4)$$

Here, S_n is the set of symmetric $n \times n$ matrices, and $L := \text{Diag}(We) - W$ is the *Laplace matrix* of G while e is the all-ones vector of dimension n and W is the weighted adjacency matrix of G (i.e., $w_{ij} = w_{ji} = c_{ij}$ for all $\{i,j\} \in E$ with $i < j$, and $w_{ij} = 0$ otherwise). Due to (4), a feasible solution satisfies $X = yy^\mathsf{T}$ for some $y \in \mathbb{R}^n$. This implies that X is positive semidefinite which we denote by $X \succeq 0$. In addition, (3) enforces that actually $y \in \{-1,1\}^n$, which encodes the two vertex partitions of a cut in G. Compared to x_{ij} in (MC-ILP), the interpretation of X_{ij} is therefore shifted in the sense that $X_{ij} = 1$ if i and j are assigned to the *same* partition of V and $X_{ij} = -1$ otherwise.

To finally arrive at an SDP relaxation, the non-convex rank constraint (4) in (MC-ISDP) is relaxed to $X \succeq 0$. We will show in Section 3 that Theorem 1 can be extended to this relaxation. Moreover, it is frequently enriched by the following *triangle inequalities*:

$$-X_{ij} - X_{ik} - X_{jk} \leq 1 \quad \text{for all } 1 \leq i < j < k \leq n \quad (5)$$
$$-X_{ij} + X_{ik} + X_{jk} \leq 1 \quad \text{for all } 1 \leq i < j < k \leq n \quad (6)$$
$$X_{ij} - X_{ik} + X_{jk} \leq 1 \quad \text{for all } 1 \leq i < j < k \leq n \quad (7)$$
$$X_{ij} + X_{ik} - X_{jk} \leq 1 \quad \text{for all } 1 \leq i < j < k \leq n \quad (8)$$

Since we are going to derive branching rules for both LP and SDP relaxations, we now proceed with discussing related work in this field.

For approaches based on LP-based formulations with edge variables like (MC-ILP), it is so far common to rely on generic branching rules for integer linear programming such as reliability pseudo-cost or pseudo-cost branching (see e.g. [1, 13]). The former is for instance the default branching rule in the optimization framework SCIP [19], and the latter has been employed by the problem-specific QuBowl solver by Rehfeldt et al. [27]. Thereby, a new branching strategy is explicitly mentioned as a potentially promising ingredient for further improvements in the conclusion of [27]. When using additional node variables, like in the MaxCut solver McSparse [9], a study by Charfreitag et al. [10] reveals that their MaxCut-specfic *degree-dynamic* branching rule can be superior in certain cases. We will come back to this in more detail when presenting our computational results in Section 5.

State-of-the-art SDP-based approaches for MaxCut rely on (enriched) relaxations of (MC-ISDP). Thereby, the employed branching rules are essentially based entirely on the entries of the matrix obtained from solving these relaxations. Such rules have been investigated in [18, 28], resulting in the popular strategy of branching on a least decided (or "most infeasible") variable, i.e. one that has a value furthest from ± 1. In fact, this is the default branching rule used in BiqMac [28], BiqBin [15] and BiqCrunch [23].

3 Spanning-Tree Integrality for SDP Relaxations of MaxCut and Related Problems

In this section, we present an extension of Theorem 1 to the SDP relaxation as described in Section 2. Although the proof may also be laid out more independently and briefly when using additional algebraic arguments as noted below, we first provide a complete translative and constructive one that takes the triangle inequalities into account and that highlights additional relationships which may be of utility also in other contexts. In particular, we emphasize that these results straightforwardly extend to ± 1-SDP formulations of the rich class of binary quadratic optimization problems that can be and are modeled by appending constraints to (MC-ISDP) that are linear in X [29].

Theorem 2. *Let $\bar{X} \in S_n$ be a solution to the SDP relaxation of (MC-ISDP) enriched with the triangle inequalities (5)–(8), and let $T = (V, E_T)$ be an arbitrary spanning tree of G (or its completion). If $\bar{X}_{ij}$ is ± 1 in the components for all $\{i, j\} \in E_T$, then $\bar{X}_{ij} \in \{-1, 1\}$ for all $i, j \in \{1, \ldots, n\}$, $i \neq j$. Consequently, $\bar{X}$ encodes the incidence vector of a cut in G (and its completion) and* $\operatorname{rank}(\bar{X}) = 1$.

Proof. Since the SDP relaxation of (MC-ISDP) has a variable $X_{ij} = X_{ji}$ for all $i, j \in \{1, \ldots, n\}$, $i < j$, we first consider to augment $G = (V, E)$ to a complete graph $G' = (V, E')$ where each edge $\{i, j\} \in E' \setminus E$ receives the weight $c_{ij} = L_{ij} = L_{ji} = 0$. In fact, solving (MC-ISDP) (or a relaxation of it) for G means precisely solving it for G'.

Let now $\bar{X}$ be a feasible solution to the SDP relaxation of (MC-ISDP) where $\bar{X}_{ij} \in \{-1, 1\}$ for the edges $\{i, j\} \in E_T$ of the spanning tree $T = (V, E_T)$ of G'. Using the bijection $x_e = \frac{1}{2}(1 - X_e)$, we obtain a vector $\bar{x} \in \mathbb{R}^{E'}$ such that $\bar{x}_e \in \{0, 1\}$ if and only if $\bar{X}_e \in \{-1, 1\}$ and $0 < \bar{x}_e < 1$ otherwise. Moreover, applying this variable transformation to the triangle inequalities (5)–(8), one obtains their following LP-based pendants (9)–(12).

$$x_{ij} + x_{jk} + x_{ik} \leq 2 \quad \text{for all } i, j, k \in V, i < j < k \quad (9)$$
$$x_{ij} - x_{jk} - x_{ik} \leq 0 \quad \text{for all } i, j, k \in V, i < j < k \quad (10)$$
$$-x_{ij} + x_{jk} - x_{ik} \leq 0 \quad \text{for all } i, j, k \in V, i < j < k \quad (11)$$
$$-x_{ij} - x_{jk} + x_{ik} \leq 0 \quad \text{for all } i, j, k \in V, i < j < k \quad (12)$$

Since $\bar{X}$ satisfies (5)–(8), $\bar{x}$ satisfies (9)–(12) which are the particular special cases of the odd-cycle inequalities (1) for $|C| = 3$. Moreover, since G' is complete, these triangles are the only cycles in G' that are chordless, and therefore (9)–(12) are the only instances of (1) that are facet-inducing for the cut polytope [4] regarding

G', i.e., irredundant. In other words, $\bar{x}$ satisfies (1), and is thus a feasible solution to the LP relaxation of (MC-ILP) for G' with $\bar{x}_{ij} \in \{0,1\}$ for all $\{i,j\} \in E_T$. Consequently, Theorem 1 applies whence $\bar{x}_{ij} \in \{0,1\}$ respectively $\bar{X}_{ij} \in \{-1,1\}$ also holds for all $\{i,j\} \in E' \setminus E_T$, i.e., the off-diagonal entries of $\bar{X}$ encode the incidence vector of a cut in G'. We conclude that with $\bar{X}_{ii} = 1$ for all $\{1,\dots,n\}$, $\bar{X}$ is thus a rank-1 matrix. □

For readers familiar with the theory of semidefinite optimization, we complement the proof by a shorter version that is based on a geometric argument and that shows in addition that the presence of the triangle inequalities is not even needed in the SDP case. Besides that, this result is stronger than a similar one in [25] (Theorem 4.3) in the sense that the connectedness of the subgraph defined by ± 1-edges (as here established by the spanning tree with a minimum number of edges) is sufficient to obtain a rank-1 matrix, rather than that this subgraph was required to be chordal.

The proof is as follows: Since $\bar{X}$ is positive semidefinite, it can be decomposed as $\bar{X} = U^\mathsf{T} U$, i.e. it is the Gram matrix of the column vectors $u_1,\dots,u_n$ of U. As $\bar{X}$ has an all-ones diagonal, the vectors have unit length and $\bar{X}_{ij} = u_i^\mathsf{T} u_j$ for any $i,j \in \{1,\dots,n\}$. For each edge $\{i,j\} \in E_T$ of the given spanning tree T, the inner product $u_i^\mathsf{T} u_j$ attains either the value -1 or 1, and thus the Cauchy-Schwarz inequality is satisfied with equality. This means that the vectors u_i and u_j must be equal or diametrically opposed on the unit n-sphere. Now consider, for every $i,j \in \{1,\dots,n\}$, $i \neq j$, the edges of the unique path P_{ij} connecting i and j in the spanning tree. Since $\bar{X}_{vw} = u_v^\mathsf{T} u_w \in \{-1,1\}$ for all $\{v,w\} \in P_{ij}$, the vectors of all the vertices in between i and j (if any) are equal or diametrically opposed as well. Substitution along each P_{ij} thus forces $u_i = \pm u_j$ and $\bar{X}_{ij} = \pm 1$ for every distinct pair i,j. Consequently, there exists a vector $s \in \{-1,1\}^n$ such that $U = u_1 s^\mathsf{T}$, i.e. $u_j = s_j u_1$, and the initial decomposition can be made explicit: $\bar{X} = U^\mathsf{T} U = s(u_1^\mathsf{T} u_1) s^\mathsf{T} = ss^\mathsf{T}$. Thus, we have $\operatorname{rank}(\bar{X}) = 1$, and s encodes the incidence vector of a cut in G (or its completion).

4 Branching Rule Design

This section is devoted to the presentation of our novel *spanning-tree* branching rule for the LP- and SDP-based MaxCut formulations from Section 2. We start by giving our conceptual motivation and key design ideas, before providing the formal definitions of the ingredients used to make the actual branching decisions. Then, we present a precise description of the algorithmic steps to select a branching variable from a given LP or SDP relaxation solution, followed by a discussion of important implementation details and side aspects.

A common and simplistic viewpoint on the solution of a relaxation which is particularly relevant for the design of branching rules is that it provides an *indication* on "tendencies" in terms of value assignments in an optimum integer feasible solution, and on the "degree of decidedness" with respect to the (two) actual options expressed by a (binary) variable. A key idea behind our approach is therefore to foster the construction of a feasible solution, respectively of a spanning tree of binary edge variables, *dynamically* and informed by the successively observed relaxation solutions (rather than statically choosing such a spanning tree to enforce upfront).

Conversely, an according *spanning-tree* branching rule should also be designed to guide the generation of subproblems towards a reasonable expansion of intermediate tree components (subgraphs with decided edge variables) by additional (more structural) means than just interpreting the relaxation solution. In this respect, a central ingredient for the original spanning-tree integrality proof from [26] is the observation that the integrality of a non-tree edge $e = \{i,j\} \in E \setminus E_T$ is implied by the odd-cycle inequalities that refer to the fundamental cycle in terms of T that is closed by e. More precisely, let P_{ij} be the set of edges on the unique path connecting i and j in T. Then the odd-cycle inequalities (1) for $C = P_{ij} \cup \{e\}$ imply that $x_e = 0$ if the number of partition changes on P_{ij}, i.e., $\sum_{f \in P_{ij}} x_f$ where all summands are binary, is even, and $x_e = 1$ otherwise [26].

More generally, given Theorems 1 and 2, any branching process for the formulations (MC-ILP) and (MC-ISDP) can be thought of as growing a spanning forest of G respectively G'. Specifically, at each subproblem, a certain set of edges F that forms a subgraph (V,F) of G or G' has been branched on. From the discussion in the previous paragraph, we further conclude that, in the presence of odd-cycle respectively triangle inequalities, an edge $e = \{i,j\} \in E$ respectively $e = \{i,j\} \in E'$ will not be a branching candidate if F contains a path connecting i and j, as then x_e is already implied integral.

A major design principle for our branching rule is therefore to (greedily) expand the current forest F in a way that supports to reach many edges whose associated variables are not implied integral yet or as a consequence of the current expansion step, respectively to reach many vertices after this expansion step which are not yet matched by any edge in F. Thereby, we do not truly "count" such edges but consider their "undecidedness" in the current relaxation solution as well as their absolute weight, which is reflected in the weighted infeasibility measure defined below. Besides that, a few further ingredients influence our branching candidate selection.

Formalization: To formalize this, we denote with $d_F(v)$ the degree of $v \in V$ w.r.t. the edge set F of the current spanning forest (V,F), and with $d_E(v)$ the degree of v with respect to G. Analogously, we use $\delta(v) \coloneqq \delta(\{v\}) = \{\{v,w\} : \{v,w\} \in E\}$ as a shorthand reference to the adjacent edges of $v \in V$. Moreover, we denote with $c_F(v)$ the connected component of the subgraph (V,F) that contains v. The current solution of the LP (SDP) relaxation for which we make a branching decision is referred to as $\bar{x}$ ($\bar{X}$). It naturally defines the set of general branching candidates belonging to E which we denote by $E_b \coloneqq \{e \in E : 0 < \bar{x}_e < 1\}$ respectively $E_b \coloneqq \{e \in E : -1 < \bar{X}_e < 1\}$. Based on this, we finally define the (absolute) *weighted infeasibility* $\omega_e \coloneqq |c_e| \cdot (\frac{1}{2} - |\frac{1}{2} - \bar{x}_e|)$ respectively $\omega_e \coloneqq |L_e| \cdot (1 - |\bar{X}_e|)$ of the variable associated with an edge $e \in E_b$ that we use as a local estimate of the impact on the objective when the variable would be set to its closer bound.

We are now ready to describe the main principles of our *spanning-tree* branching rule while we will discuss further implementation details afterwards.

Initialization: At the beginning ($d_F(v) = 0$ for all $v \in V$), we first determine a vertex $u \in V$ whose adjacent edges have the largest total weighted infeasibility, i.e., $u \coloneqq \operatorname{argmax}\{w \in V : \sum_{e \in \delta(w) \cap E_b} \omega_e\}$. Then we pick the edge $\{u,v\} \in \delta(u) \cap E_b$ such that v has the largest total weighted infeasibility $\sum_{e = \in \delta(v) \cap E_b} \omega_e$ among the neighbors of u in terms of $\delta(u) \cap E_b$. Clearly, these totals tend to be larger if $d_E(u)$ and $d_E(v)$ are large, whence this choice supports to gain many options for picking an adjacent follow-up edge in general.

Expansion: We then proceed with expanding a current forest edge set $F \neq \emptyset$ in three principal ways, one of which will be chosen in each case as explained hereafter.

1. Expand a non-singleton connected component (tree) of (V,F): Pick an edge $\{u,v\} \in E_b$ such that $d_F(u) > 0$, $d_F(v) = 0$, and such that the primary score
$$\sum_{\substack{e=\{v,w\} \in E_b: \\ d_F(w)=0}} \omega_e,$$
i.e. the total weighted infeasibility of all the edges in $\delta(v) \cap E_b$ whose other endpoint is not yet matched by (V,F), is maximum. This is illustrated in Figure 1. If two edges have the same primary score, a tie breaking applies. For $\{u,v\}$, the secondary score is calculated as
$$\sum_{\substack{e=\{v,w\} \in E_b: \\ w \neq u,\ c_F(w)=c_F(u)}} \omega_e,$$
and thus captures the total weighted infeasibility of all the edges adjacent to w that would be implied integral as a consequence of closing a cycle with the tree path P_{vw} which results from branching on $\{u,v\}$, see again Figure 1. The decision is then made based on a larger secondary score, and in case of another tie, the edge encountered first is picked.

2. Initiate a new non-singleton component of (V,F) by repeating the initialization step in a slightly adapted way. More precisely, pick an edge $\{u,v\} \in E_b$ such that $d_F(u) = d_F(v) = 0$. Thereby, determine u and v in the same fashion as in the initial step except for that the sums over the ω_e are only built over those $e = \{i,j\}$ where $d_F(i) = d_F(j) = 0$ holds in addition. A conceptual example regarding this expansion strategy is illustrated in Figure 2.

3. Connect two non-singleton components of (V,F), i.e., pick an edge $\{u,v\} \in E_b$ such that $d_F(u) > 0$ and $d_F(v) > 0$[1]. We consider the primary score of such edges to be zero, and take an edge $\{u,v\}$ such that the total weighted infeasibility of all edges between $c_F(u)$ and $c_F(v)$ (secondary score) is as large as possible. Again, in case of a tie, the edge encountered first is taken.

[1] In this case, we necessarily have $c_F(u) \neq c_F(v)$ as otherwise $\{u,v\}$ were implied integral and thus not in E_b.

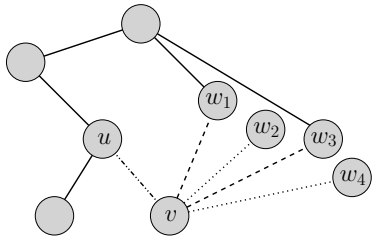


Figure 1: Expanding an example tree (solid edges) according to the first strategy. The primary score of the branching candidate $\{u,v\}$ (dash-dotted) is determined by the weighted infeasibilities associated with the dotted edges to so-far unmatched neighbors of v. The secondary (tiebreak) score is determined by the weighted infeasibilities associated with the dashed edges to already matched neighbors of v (including u). Only edges relevant in this context are displayed while further ones could exist.

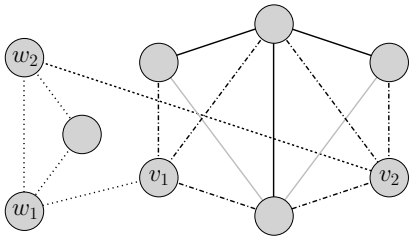


Figure 2: An example situation where the second expansion strategy is likely to apply. Here, the successive expansion of the tree edges (solid black) has the consequence that the gray solid edges are implied integral. Moreover, the primary scores of the branching candidates relevant for the first expansion strategy (dash-dotted) depend on the weighted infeasibilities of only a *single* edge – either on ω_{v_1,w_1} or on ω_{v_2,w_2}. At the same time, their secondary scores depend on the weighted infeasibilities of the respective other *two* dash-dotted edges incident to v_1 respectively v_2. This reflects the already relatively high level of implication around vertices that could be attached to the tree, and increases the chance that the primary scores are lower than the secondary ones. If this is true, the second expansion strategy will select one of the dotted edges that have no endpoint matched by forest edges so far. As opposed to Figure 1, it is here assumed that only the displayed edges exist.

Selection: For every branching decision after initialization, first an incumbent edge according to the first expansion strategy is derived (if any), to support the aforementioned major design principle. In cases where the conditions of the first strategy do not apply or the provided incumbent has a primary score that is smaller than its secondary one, the second expansion strategy is taken into account and may override the incumbent edge if an edge satisfying its conditions is found. In the seldom case that neither the first nor the second strategy applies, the third expansion rule finally provides an edge to branch on.

With the second option, we provide a mechanism to "escape" from a successive local expansion of a non-singleton tree component as performed by the first strategy, respectively to enforce branching decisions in so far undecided parts of the graph when this appears more promising based on the scoring. In case of an incumbent $\{u,v\}$ stemming from the first expansion strategy, we consider this to be true if the weighted total infeasibility of v-adjacent edges implied integral (secondary score) has become larger than the weighted total infeasibility of so far unmatched v-adjacent edges (primary score). Thereby, we consciously assign only a low priority to the third expansion strategy, i.e. the connection of two non-singleton components of (V,F), because we expect the odd-cycle (respectively triangle) inequalities involving edges of both components to already have a relatively strong impact on those that connect the two components.

Implementation details: Given the above description, an according algorithmic implementation of the *spanning-tree* branching rule is immediate and well-defined: Based on the available representation of the current forest (V,F), select a branching variable from the current set of E_b-candidates by means of the described successive consideration of the three expansion strategies while all the required weighted infeasibility values are available from the current relaxation solution. Specifically, the if-then-else case distinction is thereby precisely as follows: If the first strategy gives a candidate, then it also has a primary and secondary score, and it is taken if the primary score is at least as large as the secondary score. If no candidate is returned or the primary score is smaller than the secondary score, determine the best candidate according to the second strategy (if any). If still no candidate is determined, retrieve it with the third strat-

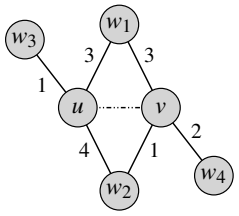
egy. However, depending on the problem representation that is used, a few details have to be taken into account in practice.

First of all, in a representation where only variables for edges in E are present, as is customary especially for LP-based approaches, our principle to branch only on original edges is implicitly established. To achieve this also for SDP relaxations, which usually act on a matrix X involving variables for any pair of vertices as discussed above, one needs to admit only variables associated with non-zero off-diagonal entries in the matrix L as branching candidates *explicitly*.

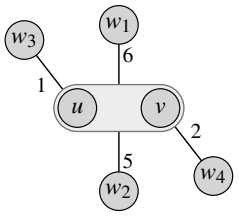
The implementation of the *spanning-tree* branching rule is further straightforward if, at each subproblem of the branch-and-bound procedure, a problem representation is available that provides access to all the original variables that are fixed to a certain value. This applies in particular to general purpose solvers and makes it possible in addition to consider edges as members of F whose associated variable has obtained equal lower and upper bounds by other means than branching, such as e.g. propagation or bound tightening techniques. We follow this principle in our implementation.

In more problem-specific representations of the branch-and-bound procedure, which are particularly common for SDP-based MaxCut solvers, subproblems are however often built by contracting the branching edge. The graph and the Laplace matrix L considered in any subproblem other than the root are thus not the original ones, and each vertex represents a (possibly singleton) connected component of the forest (V,F) while no F-edge is ever present.

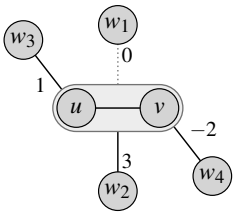
During contraction, the edges incident to vertices that are adjacent to both endpoints of the branching edge are bundled into one, and the weight of that remaining edge is updated by a (signed) aggregation of all the weights participating in the contraction. Figure 3 shows both children arising from an example subproblem. In the special case where this aggregated weight is zero, as for $\{\{u,v\},w_1\}$ in Figure 3(c), the separate weights of the bundled edges thus cancel out for any cut that is compatible with the current (partitioning of the) forest (V,F). According to our principle of branching only on edge variables with a non-zero L-entry, such variables will not be considered as branching candidates even though their value might be different from ± 1. This is meaningful because such edges have indeed no impact on the objective value. It is also



(a) Subproblem before branching on $\{u,v\}$.



(b) Child subproblem obtained by setting $X_{uv} = 1$, i.e. excluding $\{u,v\}$ from the cut.



(c) Child subproblem obtained by setting $X_{uv} = -1$, i.e. including $\{u,v\}$ into the cut.

Figure 3: Contracting the branching edge $\{u,v\}$ (dash-dotted) into the two child subproblems, in which u and v are meld into one component of the forest (V,F) (shaded). In (c), the signs of the edges incident to v are flipped (upon adding a constant to the objective), before contracting $\{u,v\}$ under the removal of v. Since w_1 and w_2 are adjacent to both endpoints, each respective pair of edges is bundled into a single one whose weight is the signed aggregation $L_{uw_i} + L_{vw_i}$ in (b) and $L_{uw_i} - L_{vw_i}$ in (c) for $i = 1,2$, whereas the edges to w_3 and w_4 and their weights are inherited. In (c), the aggregated weight of $\{\{u,v\},w_1\}$ vanishes, so the separate weights cancel out for every cut compatible with (V,F) and the corresponding L-entry is zero. The associated variable is hence not considered as a branching candidate even though it may be undecided. Only edges relevant for the displayed contraction operations are displayed while further ones could exist.

feasible because, in the only critical and hypothetical case where L is zero in all off-diagonal entries and the primal bound is not large enough to cut off the current subproblem, branching on any variable $\bar{X}_{ij} \neq \pm 1$ is fine to successively produce a feasible solution of precisely the current relaxation's objective value. Therefore, we then just resort to branching on a most undecided variable.

Finally, while the primary score used for the first expansion rule is unaffected by contractions, as only edges between singleton connected components count towards it, they require an adaption of the secondary score computation. More precisely, the contractions have the consequence that the edges that would be implied integral when branching on an edge $e \in E_b$, i.e. those relevant for the secondary score, have already been bundled into e itself. Therefore, the secondary score of a branching candidate e then corresponds to, and is in our implementation considered to be, the weighted infeasibility ω_e of e alone.

Experiments with variations: We also experimented with variants of our branching rule. In particular, we tested the effect of using the weighted infeasibility of only the branching candidate itself as the secondary score for the LP-based setting. Moreover, we tried to use the *unweighted* infeasibility instead for the calculation of the primary scores. However, this did not lead to better results for the diverse set of instances considered for our computational experiments in Section 5. Rather, the first expansion strategy and the presented primary score turned out to be the dominant ingredients for the sustained performance.

Connections to other branching rules: We close this section by briefly highlighting that our branching rule integrates aspects of common general as well as problem-specific ones. For instance, the weighted infeasibility scoring and the expansion towards vertices with many unmatched neighbors resembles typical (weighted) "most infeasible" respectively "most undecided" strategies. Since a high primary score for edges $\{u, v\}$ as considered in the first expansion strategy is more likely in case of a high value $d_E(v) - d_F(v)$, there is also some relation to the *degree-dynamic* rule mentioned in Section 2. Even more, fixing a node variable can be seen as fixing the edge variable to a predetermined reference vertex that is adjacent to all other vertices and thus induces a star-type spanning tree [10]. Whereas *degree-dynamic* then tends to fix a node variable of large undecided vertex degree, our rule tends to fix an edge variable that newly connects a vertex of high degree to a connected component (tree) of the forest.

5 Computational Experiments

To demonstrate the practical effect of our new rule, we conduct a computational study that closely follows the one in [10], where the *degree-dynamic* rule was introduced, but that also integrates experiments with SDP relaxations. Overall, we evaluate our branching rule as designed in Section 4 against the standard and problem-specific rules used in related work as addressed in Section 2 and taken up below.

5.1 Experimental Setup

The experimental setup has a few LP- and SDP-specific aspects that we describe first while it is kept as uniform as possible apart from these.

LP-specific Setup: We implemented our spanning-tree branching rule using the online[2] available code used in [10], which also provides the *degree-dynamic* rule applied to the formulation (RE) with node variables from [10] as well as the state-of-the-art branching rules *reliability pseudocost branching (scip-default)* and *pseudocost branching (scip-pseudocost)* against all of which we compare. It is written in C++20 and compiled with GCC 11.4, linking to NetworKit 10.1 [2] for graph representation, and to SCIP 10.0.1 [19] as the MIP-solver combined with the commercial LP-solver CPLEX 22.1.2 [20]. Naturally, due to the more recent software versions and the different hardware (see below), the results slightly deviate from those in [10]. Apart from this, we precisely adopt the setup from this reference. In particular, the odd-cycle inequalities (1) are not all present from the beginning but added (separated) iteratively if violated by the current relaxation solved in a subproblem of the branch-and-bound procedure. Moreover, the general purpose presolve as well as separation and conflict analysis components of SCIP are turned off, further isolating the performance impact of the branching rules.

SDP-specific Setup: Here, we implemented the proposed branching rule in an early Julia implementation[3] of

[2] `https://github.com/CharJon/ScientificMaxcutSolver`

[3] This preliminary BiqMac implementation is not yet runtime optimized. Therefore, the reported CPU times are not compatible with other publications of BiqMac, and only viable as a coarse indicator of relative performance. In the present context, this is negligible, as all competing SDP-based branching rules are implemented under the same conditions. Moreover, the number of branch-and-bound subproblems is accurate.

BiqMac [28], with the adaptions for a contraction-based generation of subproblems as described in the implementation details in Section 4. The code is executed with Julia 1.12.5 and uses OpenBLAS 0.3.29 for linear algebra computations, which we explicitly restricted to a single thread. BiqMac solves the SDP relaxation discussed in Section 2, enriched with triangle inequalities, using a dual bounding procedure. Again, the triangle inequalities are thereby taken into account in an iterative manner if violated by the current relaxation solved in a subproblem of the branch-and-bound procedure [28] and we do not apply any preprocessing. By the default, the solver uses the "most infeasible" branching rule, called R3 in [28], which is standard also in other SDP-based solvers (see Section 2). It further provides the "easiest first" branching rule, called R2 in [28]. We compare our rule to both R2 and R3, which were first experimentally evaluated in [18].

Instance Selection: For our experiments we consider a broad set of instances (obtained from [7]) varying in size, density, and weight distribution, comprising instances stemming from native QUBO as well as native MaxCut problems, and building in particular a strict superset of the ones used in [10]. Each of the according instance classes is detailed below and consists of 10 distinct instances. For the LP-based experiments, we slightly extend the instance selection of [10]. This earlier selection contains the QUBO instance classes **be250** and **be120.3** generated by Billionnet and Elloumi [6], whose respective original matrices have $n = 250$ with density 0.1 and $n = 120$ with density 0.3, each taking integer values in $[-100, 100]$ on the diagonal and $[-50, 50]$ on the off-diagonal entries. It further contains the QUBO instance class **bqp250** generated by Beasley [5], where the original matrices have $n = 250$ and density 0.1 taking integer values in $[-100, 100]$. As native MaxCut instance classes, the (Erdös-Renyi-Gilbert) random graphs **pm1s** with $|V| = 100$, edge weights ± 1 and density 0.1 are included. To add more variation in the magnitude of the edge weights, we further append the instance set **w01** to our selection, which differs from the previous class in having integer edge weights in $[-10, 10]$. For the SDP-based experiments, we then further complement our selection by medium-sized instance classes with density at least 0.5, as these are well-known to be more routinely solved with SDP- rather than LP-based methods. Specifically, we add **be120.8** from [6], which differs from **be120.3** in having density 0.8, the class **g05** of unweighted random graphs with $|V| = 100$ and density 0.5 and **pm1d**, which differs from **pm1s** in having density about 0.99.

Performance Metrics: Like in [10], we run each solver configuration three times using different random seeds, and then each run is treated as a separate data point in order to mitigate the effects of performance variability [24]. Then, the experimental results are aggregated per branching rule and instance class, as well as over all runs. For each rule, we report statistics on the CPU times in seconds as well as on the number of branch-and-bound subproblems, each summarized by the arithmetic mean, the median, the standard deviation (std), maximum, and shifted geometric mean (sgm). The latter is computed with a shift of 1s for the CPU times and unshifted for the number of branch-and-bound subproblems. Runs that reach the time limit count towards the CPU times statistics with the time limit itself, but are penalized for the branch-and-bound subproblem statistics regarding mean, median, std and sgm. More precisely, if no rule solves such a run, it is excluded from the statistics. Otherwise, if a run reaches the time limit, the corresponding number of branch-and-bound subproblems is replaced by the largest number of branch-and-bound subproblems among all the rules that solved this run within the time limit regarding the mentioned metrics. This yields more meaningful statistics, as it does not reward "fewer subproblems at the occurrence of a timeout", albeit deviating from [10]. Finally, the number of wins and timeouts are displayed, where a run is counted as a win for every rule reaching the minimum CPU time / number of branch-and-bound subproblems, as in [10].

Primal Heuristics & Solutions: All primal heuristics are turned off, and an optimum solution is passed as an incumbent to the root problem instead. This isolates the effects of the branching rule from those of subproblem selection as well as the effectiveness of the primal heuristic, see e.g. [12, 22], and again complies with [10].

Hardware & Operating System: The computational experiments are conducted on a machine equipped with an Intel Xeon E-2386G CPU with 3.50GHz, and 128 GB RAM running Ubuntu 22.04. All experiments are run single-threaded.

type	setting	cpu time [s] mean	median	std	max	sgm	wins	timeouts	#branch-and-bound subproblems mean	median	std	max	sgm	wins
be120.3	degree-dynamic	66.51	34.20	104.77	372.35	34.38	8	**0**	112.87	47.00	206.43	721	39.33	3
	scip-default	116.73	68.51	155.77	617.18	66.29	3	**0**	110.07	**29.00**	240.24	907	**28.94**	**19**
	E-spanning-tree	**62.45**	**29.49**	**104.12**	**369.91**	**31.53**	**19**	**0**	**88.60**	35.00	**170.24**	**589**	29.66	15
	E-scip-default	200.90	86.35	343.04	1380.56	89.35	0	**0**	185.73	70.00	365.04	1423	53.28	3
	E-scip-pseudocost	543.47	164.41	1052.57	3600.00	140.50	0	2	789.43	243.00	1528.61	5463	158.76	3
be250	degree-dynamic	840.53	411.34	**1103.55**	3600.00	343.26	8	**3**	399.07	289.00	460.62	1447	189.52	2
	scip-default	1091.11	596.19	1207.40	3600.00	569.90	3	**3**	419.00	**193.00**	583.57	2009	163.16	9
	E-spanning-tree	**808.15**	**353.71**	1132.51	3600.00	**301.40**	**25**	**3**	**317.30**	203.00	**424.92**	**1407**	**137.69**	**17**
	E-scip-default	1506.93	949.56	1448.58	3600.00	750.77	3	7	577.37	275.00	630.29	1719	254.47	0
	E-scip-pseudocost	2385.57	3502.33	1517.69	3600.00	1443.17	3	15	1472.44	1993.00	983.86	2543	875.41	0
bqp250	degree-dynamic	1335.60	485.19	1550.05	3600.00	433.88	9	**9**	261.86	101.00	337.22	985	109.57	0
	scip-default	1438.72	769.87	**1495.19**	3600.00	645.59	9	**9**	216.90	87.00	267.69	745	78.68	**12**
	E-spanning-tree	**1264.91**	**358.44**	1572.85	3600.00	**371.53**	**30**	**9**	**159.86**	**73.00**	**185.50**	**565**	**76.22**	10
	E-scip-default	1726.25	1098.35	1605.68	3600.00	769.53	9	11	369.33	99.00	527.70	1596	126.76	0
	E-scip-pseudocost	2232.02	3600.00	1555.91	3600.00	1182.57	9	16	1040.10	729.00	896.74	2455	511.18	0
pm1s	degree-dynamic	**6.78**	**6.00**	**3.51**	**12.35**	**5.99**	**27**	0	**159.27**	112.00	**102.21**	**353**	**129.16**	**18**
	scip-default	15.84	14.28	5.49	25.65	15.02	0	0	189.93	**102.00**	159.21	503	132.75	12
	E-spanning-tree	13.98	10.72	10.03	34.15	10.66	3	0	504.80	348.00	428.62	1339	319.55	0
	E-scip-default	36.60	30.89	27.57	99.36	27.13	0	0	837.87	751.00	714.55	2235	477.79	0
	E-scip-pseudocost	73.49	56.30	74.57	266.11	40.40	0	0	2831.00	2137.00	2897.99	10579	1458.24	0
w01	degree-dynamic	**2.43**	**2.09**	**1.57**	**6.95**	**2.19**	**26**	0	64.93	54.00	37.74	175	58.09	6
	scip-default	8.35	8.46	1.70	11.91	8.19	0	0	**53.40**	**51.00**	**32.38**	**159**	**44.34**	**27**
	E-spanning-tree	3.19	2.45	2.68	10.94	2.71	4	0	114.73	67.00	142.68	545	82.49	0
	E-scip-default	8.60	6.58	6.37	33.76	7.51	0	0	143.93	63.00	251.89	1179	81.51	1
	E-scip-pseudocost	10.79	4.43	19.92	94.27	5.55	0	0	503.80	112.00	1078.47	5109	182.12	0
Total	degree-dynamic	450.37	29.44	1002.45	3600.00	**40.16**	78	**12**	**191.20**	74.00	**287.64**	1447	88.13	29
	scip-default	534.15	66.94	1046.72	3600.00	80.78	15	**12**	191.81	**57.00**	329.89	2009	**72.22**	**79**
	E-spanning-tree	**430.54**	**28.60**	**1000.56**	3600.00	42.66	**81**	**12**	240.35	72.00	338.52	**1407**	96.82	42
	E-scip-default	695.86	86.35	1229.40	3600.00	103.62	12	18	422.98	96.00	581.35	2235	145.86	4
	E-scip-pseudocost	1049.07	121.36	1495.94	3600.00	144.67	12	33	1342.93	489.00	1882.34	10579	441.99	3

Table 1: Aggregated ILP-branching rule results per instance class. Each instance was run three times and the time limit was 3600 s. Maintaining compatibility with [10] we prefix branching rules on the edge model with E.

5.2 LP-based Branching Results

For the edge-based formulation (MC-ILP), we compare our novel rule to SCIP's pseudocost branching (*scip-pseudocost*) and reliability pseudocost branching (*scip-default*). In addition, we include the problem-specific rule *degree-dynamic* which operates on the (RE) model from [10] that employs and branches on variables which are technically node variables. Hence, comparing *spanning-tree* with *degree-dynamic* does not only refer to different branching rules but also to different MaxCut formulations. From now on, we thus adopt from [10] the addition of a prefix E whenever we refer to a branching rule with respect to (MC-ILP), including *E-spanning-tree* for our new rule, while no prefix means that the rule is applied to the (RE) model.

A summary of our results is displayed in Table 1. In total, our branching rule *E-spanning-tree* performs best regarding several metrics with respect to CPU time, and it achieves the most wins and a minimum number of timeouts. Moreover, it is also the clear overall winner when compared to the other E-rules. Across models, *degree-dynamic* and *scip-default* partly reach better results for some of the statistics such as the mean, median, and shifted geometric mean values in terms of branch-and-bound subproblems, and the shifted geometric mean (only *degree-dynamic*) running time, whereas *E-spanning-tree* reaches the best values otherwise.

A well differentiated picture is visible on the instance class level. For the QUBO instances (**be120.3**, **be250**, **bqp250**), *E-spanning-tree* outperforms the problem-

specific rule *degree-dynamic* in mean, median and shifted geometric mean runtime, the latter by up to 14%, and achieves by far the most wins. Among all rules, *E-spanning-tree* here leads to the fewest branch-and-bound subproblems in the metrics mean and max. For the native MaxCut instances (**pm1s**, **w01**), a strong impact of the underlying model on the mean number of branch-and-bound subproblems is clearly visible. Here, both rules based on the (RE) model need fewer of them than the E-models, and particularly *scip-default* needs significantly fewer than *E-scip-default*. In terms of running time, *degree-dynamic* outperforms *E-spanning-tree* in absolute terms on these instances, but the relative runtime improvement of these problem-specific rules to the respective general purpose rule *scip-default* and *E-scip-default* is comparable. Regarding the number of branch-and-bound subproblems, however, *E-spanning-tree* reduces this number considerably further than *degree-dynamic* does, namely by about 40% instead of 16% on **pm1s**, when compared to *E-scip-default* respectively *scip-default*. Notably for **w01**, the mean number of subproblems even increases by about 22% in *degree-dynamic* w.r.t. *scip-default*, whereas it decreases by about 20% in *E-spanning-tree* w.r.t. *E-scip-default*.

As far as robustness is concerned, we report that *E-spanning-tree* never times out on instances that can be solved by any of the other considered branching rules, independent of the underlying model. Finally, we report that the apparent superiority of *degree-dynamic*, respectively the underlying (RE) model, for ± 1-weighted instances such as **pm1s** also persists when compared to *E-spanning-tree* for accordingly weighted toroidal 2D square-grid graph instances as used in [10] as a showcase. Indeed, these instances can also be solved well with the generic *scip-default* rule on the (RE) model.

5.3 SDP-based Branching Results

In contrast to the LP-based experiments, all the three SDP-based rules *spanning-tree*, *R2-easiest-first* and *R3-most-infeasible* operate on the same underlying model, which is the SDP relaxation of (MC-ISDP) enriched with triangle inequalities as displayed in Section 2.

A summary of our results is described in Table 2. In total, *spanning-tree* is superior to both its competitors in almost every metric. It attains by far the most wins and greatly reduces the mean, median and shifted geometric mean for both CPU times and branch-and-bound subproblems. Compared to the current standard branching rule *R3-most-infeasible*, our new rule *spanning-tree* reduces the number of timeouts from 21 to 3. Furthermore, the number of branch-and-bound subproblems is reduced in the majority of the instances. Especially for the QUBO instances, *spanning-tree* is the best rule in essentially every metric and its advantage grows with the size of the considered instances, with the largest margins for **be250** and **bqp250**. For the native MaxCut instances **pm1s** and **w01**, both the mean CPU time and the mean number of branch-and-bound subproblems is roughly halved with *spanning-tree* compared to the best competing rule, whereas on **g05** and **pm1d**, *spanning-tree* is outperformed by *R2-easiest-first*. Notably, for **g05**, the number of branch-and-bound subproblems of *spanning-tree* exceeds that of *R2-easiest-first* by only about 7% but the runtime is about 38% higher. Following [18], a possible explanation lies in the long chains that *R2-easiest-first* creates in the branch-and-bound tree due to the distinct strategy of postponing difficult decisions. Hence, a large share of the branch-and-bound subproblems solved by *R2-easiest-first* reside deep in the branch-and-bound tree and, since every contraction shrinks the subproblem, are cheaper to solve than those of a more balanced branch-and-bound tree of similar size. Finally, as far as robustness is concerned, *spanning-tree* only times out on instances that also no other branching rule can solve within the time limit.

6 Conclusion

In this paper, we first extended a recent result on linear programming formulations for MaxCut to the domain of semidefinite programming formulations for MaxCut, and thus to the rich class of binary quadratic optimization problems which are modeled by appending further constraints to these: Enforcing integrality on the variables associated with the edges of a spanning tree is sufficient to obtain an overall consistent integer solution (i.e., the incidence vector of a cut).

Based on this theory, we have then devised a new *spanning-tree* branching rule that is explicitly designed to grow a spanning forest in terms of either problem representation, taking further common measures such

		cpu time [s]							#branch-and-bound subproblems					
type	setting	mean	median	std	max	sgm	wins	timeouts	mean	median	std	max	sgm	wins
be120.3	R2-easiest-first	64.15	60.78	52.52	163.63	34.33	1	0	39.20	31.00	39.70	131	14.54	9
	R3-most-infeasible	15.94	**9.11**	16.03	59.30	10.98	12	0	5.20	**3.00**	5.34	19	3.32	21
	spanning-tree	**12.12**	9.68	**8.76**	**33.62**	**9.77**	**17**	0	**3.80**	**3.00**	**2.91**	**11**	**2.86**	**30**
be120.8	R2-easiest-first	165.82	169.31	**27.99**	**202.17**	163.36	3	0	129.00	141.00	35.48	165	123.32	0
	R3-most-infeasible	167.63	104.07	147.40	507.15	114.11	2	0	54.40	34.00	50.49	167	35.65	3
	spanning-tree	**96.13**	**69.16**	75.69	260.25	**72.17**	**25**	0	**30.00**	**20.00**	**25.25**	**87**	**21.73**	**30**
be250	R2-easiest-first	2079.80	2213.66	**292.86**	2310.96	2055.73	0	**0**	256.80	280.00	71.35	377	244.67	0
	R3-most-infeasible	1953.18	1551.10	1291.98	3600.00	1441.84	0	9	134.00	71.00	127.54	162	78.52	0
	spanning-tree	**796.60**	**790.17**	440.02	**1483.45**	**630.55**	**30**	**0**	**36.00**	**36.00**	**20.04**	**69**	**28.36**	**30**
bqp250	R2-easiest-first	2253.93	2129.43	**471.87**	3600.00	2216.93	3	**3**	249.00	257.00	35.57	289	246.38	0
	R3-most-infeasible	1503.91	1255.35	1150.01	3600.00	1145.78	3	6	69.44	53.00	78.26	163	47.13	0
	spanning-tree	**896.39**	**532.80**	975.41	3600.00	**626.58**	**30**	**3**	**27.00**	**23.00**	**16.33**	**67**	**23.29**	**27**
g05	R2-easiest-first	**512.73**	**260.32**	**707.85**	**2466.20**	**273.06**	**30**	**0**	**630.00**	**305.00**	**931.15**	3231	**311.44**	**30**
	R3-most-infeasible	1138.99	684.82	1208.98	3600.00	656.69	0	3	1050.40	603.00	1155.25	**3155**	570.15	0
	spanning-tree	711.84	357.94	915.95	3232.12	411.49	0	**0**	677.00	318.00	932.08	3245	359.09	3
pm1s	R2-easiest-first	37.43	17.45	35.21	100.54	21.22	0	0	26.60	11.00	26.13	73	13.29	3
	R3-most-infeasible	46.68	12.76	57.79	173.24	20.24	2	0	26.60	**6.00**	34.37	103	10.17	15
	spanning-tree	**21.48**	**11.67**	**19.86**	**56.33**	**13.25**	**28**	0	**12.40**	**6.00**	**11.76**	**33**	**7.15**	**30**
pm1d	R2-easiest-first	**511.38**	**259.17**	**454.03**	**1478.10**	**335.39**	**27**	**0**	**603.80**	**285.00**	**559.10**	**1815**	**382.12**	**24**
	R3-most-infeasible	1396.41	660.96	1324.57	3600.00	758.41	0	3	1286.20	591.00	1237.08	3539	661.90	0
	spanning-tree	894.86	518.28	777.12	2491.28	554.33	3	**0**	843.60	472.00	762.32	2407	490.51	6
w01	R2-easiest-first	24.22	11.06	34.74	119.78	10.21	3	0	17.80	7.00	25.65	87	5.75	15
	R3-most-infeasible	20.03	**5.59**	42.79	145.73	6.74	9	0	11.60	**3.00**	26.35	89	3.19	21
	spanning-tree	**8.43**	5.85	**10.99**	**39.72**	**5.27**	**18**	0	**4.80**	**3.00**	**6.40**	**23**	**2.76**	**24**
Total	R2-easiest-first	706.18	164.42	934.80	3600.00	166.11	67	**3**	243.96	131.00	**448.53**	**3231**	73.71	81
	R3-most-infeasible	780.35	207.55	1148.61	3600.00	143.23	28	21	333.03	39.00	773.15	3539	38.91	60
	spanning-tree	**429.73**	**121.73**	**689.36**	3600.00	**93.57**	**151**	**3**	**206.57**	**23.00**	533.49	3245	**24.82**	**180**

Table 2: Aggregated SDP-branching rule results per instance class. Each instance was run three times and the time limit was 3600 s.

as (objective-weighted) infeasibilities of variables into account. Thereby, we also addressed implementation-specific details such as contractions, as these are frequently employed in branch-and-bound frameworks for MaxCut.

In an experimental study, we compared our *spanning-tree* rule against state-of-the-art generic as well as problem-specific competitors for both linear and semidefinite programming formulations on a broad set of established problem instances.

Our results show significant improvements in the required solution time and the number of branch-and-bound subproblems until proven optimality over standard and previously proposed branching rules used in state-of-the-art solvers, such as e.g. reliability pseudocost branching for linear programming and the most-infeasible branching rule for semidefinite programming.

The *spanning-tree* rule achieves a particularly clear advancement on the state-of-the-art rule for semidefinite programming, as well as a clear advancement on the previously best-performing edge-based branching rules for linear programming. The comparison to branching rules for node variables in linear programming is more nuanced. Here, *spanning-tree* achieves superior results for several instance classes and metrics again while *degree-dynamic* maintains an advantage on others. The overall runtime performance of *spanning-tree* and *degree-dynamic* is comparable. However, looking at the relative performance when compared to the respective (best) generic rule, *spanning-tree* reduces the number of branch-

and-bound subproblems considerably further than *degree-dynamic* does. Thus, also from this perspective, *spanning-tree* is a significant step ahead for purely edge-based branching, and an attractive alternative in general given its robustness across both relaxation approaches.

A future research direction in order to possibly even further improve the effectiveness of problem-specific edge-based branching rules such as *spanning-tree* is a rigorous analysis of concrete situations where branching on node variables can so far still lead to a better course of branching decisions. Our results suggest that particularly the **pm1s** instances could be an interesting class to study. Given the relations mentioned at the end of Section 4, it may then be possible to use respective insights in order to adjust the *spanning-tree* rule such that it almost resembles the behavior of *degree-dynamic* in according situations. A major challenge and open research question is whether this is possible without counteracting the here sustained robustness of *spanning-tree* across a broader set of instance classes.

Acknowledgements

We thank Christoph Helmberg for pointing us to the geometric viewpoint that inspired the corresponding alternative proof of Theorem 2. Moreover, we thank Angelika Wiegele, Johannes Schmucker and Daniel Brosch for fruitful discussions and enabling our experiments with the preliminary Julia version of BiqMac. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 537033028.

References

[1] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[2] E. Angriman, A. van der Grinten, M. Hamann, H. Meyerhenke, and M. Penschuck. Algorithms for large-scale network analysis and the networkit toolkit. In H. Bast, C. Korzen, U. Meyer, and M. Penschuck, editors, *Algorithms for Big Data: DFG Priority Program 1736*, pages 3–20. Springer Nature Switzerland, Cham, 2022.

[3] F. Barahona, M. Jünger, and G. Reinelt. Experiments in quadratic 0–1 programming. *Mathematical Programming*, 44(1):127–137, May 1989.

[4] F. Barahona and A. R. Mahjoub. On the cut polytope. *Mathematical Programming*, 36(2):157–173, 1986.

[5] J. E. Beasley. Or-library: Distributing test problems by electronic mail. *The Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

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

[7] Biq Mac Library – a collection of Max-Cut and quadratic 0-1 programming instances of medium size. `http://biqmac.aau.at/biqmaclib.html`, 2009.

[8] D. Brosch, J. Schwiddessen, and A. Wiegele. The augmented mixing method: computing high-accuracy primal-dual solutions to large-scale SDPs via column updates. *Optimization Methods and Software*, 41(3):754–796, May 2026.

[9] J. Charfreitag, M. Jünger, S. Mallach, and P. Mutzel. McSparse: Exact solutions of sparse maximum cut and sparse unconstrained binary quadratic optimization problems. In C. A. Phillips and B. Speckmann, editors, *2022 Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 54–66, 2022.

[10] J. Charfreitag, S. Mallach, and P. Mutzel. Integer programming for the maximum cut problem: A refined model and implications for branching. In J. Berry and D. B. Shmoys, editors, *Proc. of the 2023 SIAM Conf. on Applied and Computational Discrete Algorithms (ACDA23)*, pages 63–74, 2023.

[11] C. De Simone. The cut polytope and the boolean quadric polytope. *Discrete Mathematics*, 79(1):71–75, 1990.

[12] M. Fischetti and M. Monaci. Branching on nonchimerical fractionalities. *Operations Research Letters*, 40(3):159–164, 2012.

[13] G. Gamrath. *Enhanced Predictions and Structure Exploitation in Branch-and-Bound*. Dissertation, Technische Universität Berlin, Berlin, Germany, 2020.

[14] F. Glover, G. Kochenberger, and Y. Du. Applications and computational advances for solving the QUBO model. In A. P. Punnen, editor, *The Quadratic Unconstrained Binary Optimization Problem: Theory, Algorithms, and Applications*, pages 39–56. Springer International Publishing, Cham, 2022.

[15] N. Gusmeroli, T. Hrga, B. Lužar, J. Povh, M. Siebenhofer, and A. Wiegele. BiqBin: A parallel branch-and-bound solver for binary quadratic problems with linear constraints. *ACM Trans. Math. Softw.*, 48(2), jul 2022.

[16] P. L. Hammer. Some network flow problems solved with pseudo-boolean programming. *Operations Research*, 13(3):388–399, 1965.

[17] Q. Han, C. Li, Z. Lin, C. Chen, Q. Deng, D. Ge, H. Liu, and Y. Ye. A low-rank ADMM splitting approach for semidefinite programming. *INFORMS Journal on Computing*, 2025.

[18] C. Helmberg and F. Rendl. Solving quadratic (0, 1)-problems by semidefinite programs and cutting planes. *Mathematical programming*, 82(3):291–315, 1998.

[19] C. Hojny, M. Besançon, K. Bestuzheva, S. Borst, A. Chmiela, J. Dionísio, L. Eifler, M. Ghannam, A. Gleixner, A. Göß, A. Hoen, R. van der Hulst, D. Kamp, T. Koch, K. Kofler, J. Lentz, S. J. Maher, G. Mexi, E. Mühmer, M. E. Pfetsch, S. Pokutta, F. Serrano, Y. Shinano, M. Turner, S. Vigerske, M. Walter, D. Weninger, and L. Xu. The SCIP Optimization Suite 10.0. Technical report, Optimization Online, November 2025.

[20] IBM. Cplex optimization studio. `http://www.cplex.com`, 2025.

[21] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, New York*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

[22] F. Kılınç-Karzan, G. L. Nemhauser, and M. W. Savelsbergh. Information-based branching schemes for binary linear mixed integer problems. *Mathematical Programming Computation*, 1(4):249–293, 2009.

[23] N. Krislock, J. Malick, and F. Roupin. BiqCrunch: A semidefinite branch-and-bound method for solving binary quadratic problems. *ACM Trans. Math. Softw.*, 43(4), 3 2017.

[24] A. Lodi and A. Tramontani. Performance variability in mixed-integer programming. In *TutORials in Operations Research: Theory Driven by Influential Applications*, pages 1–12. INFORMS, 2013.

[25] C. Lu, Z. Deng, S.-C. Fang, and W. Xing. A new global algorithm for max-cut problem with chordal sparsity. *Journal of Optimization Theory and Applications*, 197(2):608–638, May 2023.

[26] S. Mallach. A family of spanning-tree formulations for the maximum cut problem. In A. Basu, A. R. Mahjoub, and J. J. Salazar González, editors, *Combinatorial Optimization*, pages 43–55, Cham, 2024. Springer Nature Switzerland.

[27] D. Rehfeldt, T. Koch, and Y. Shinano. Faster exact solution of sparse MaxCut and QUBO problems. *Mathematical Programming Computation*, Apr 2023.

[28] F. Rendl, G. Rinaldi, and A. Wiegele. Solving Max-Cut to optimality by intersecting semidefinite and polyhedral relaxations. *Mathematical Programming*, 121(2):307–335, 2010.

[29] N. Z. Shor. Quadratic optimization problems. *Soviet Journal of Circuits and Systems Sciences*, 25(6):1–11, 1987.