A FRAMEWORK FOR SOLVING MIXED-INTEGER SEMIDEFINITE PROGRAMS

TRISTAN GALLY, MARC E. PFETSCH, AND STEFAN ULBRICH

ABSTRACT. Mixed-integer semidefinite programs arise in many applications and several problem-specific solution approaches have been studied recently. In this paper, we investigate a generic branch-and-bound framework for solving such problems. We first show that strict duality of the semidefinite relaxations is inherited to the subproblems. Then solver components like dual fixing, branching rules, and primal heuristics are presented. We show the applicability of an implementation of the proposed methods on three kinds of problems. The results show the positive computational impact of the different solver components, depending on the semidefinite programming solver used. This demonstrates that practically relevant mixed-integer semidefinite programs can successfully be solved using a general purpose solver.

1. INTRODUCTION

The solution of mixed-integer semidefinite programs (MISDPs), i.e., semidefinite programs (SDPs) with integrality constraints on some of the variables, received increasing attention in recent years. There are two main application areas of such problems: In the first area, they arise because they capture an essential structure of the problem. For instance, SDPs are important for expressing ellipsoidal uncertainty sets in robust optimization, see, e.g., Ben-Tal et al. [10]. In particular, SDPs appear in the context of optimizing robust truss structures, see, e.g., Ben-Tal and Nemirovski [11] as well as Bendsøe and Sigmund [12]. Allowing to change the topological structure then yields MISDPs; Section 10.1 will provide more details. Another application arises in the design of downlink beamforming, see, e.g., [58]. The second main area is the reformulation of combinatorial optimization problems, see, e.g., Rendl [61] and Sotirov [70] for surveys. Indeed, SDPs are attractive in this context since they often provide strong dual bounds. Integrality constraints then already appear in the original formulation.

A natural solution method for MISDPs is branch-and-bound, and several application-specific approaches have appeared in the literature. Examples are Rendl et al. [62] for the max-cut problem, as well as Armbruster et al. [7] and Anjos et al. [6] for graph partitioning. One key aspect for all these approaches is that although SDPs usually provide strong dual bounds, they take more time to solve than linear programs. Moreover, from a software engineering perspective, SDP-solvers are not yet as advanced as their linear counterparts, for instance, with respect to presolving and numerical stability.

Going beyond the just mentioned application-specific approaches, the goal of this article is to introduce a generic framework for solving MISDPs. The motivation and longterm goal is to provide black-box-software that can be used in versatile conditions, similarly to the state-of-the-art in mixed-integer linear programming. In this paper, we make one step in this direction.

In order to provide a generic framework, several theoretical and practical challenges have to be dealt with. We first discuss the theoretical question of when strong duality can be guaranteed for each of the SDPs arising from the original one by adding linear constraints, e.g., strengthened bounds on the variables. We show in Section 5 that if strong duality holds for an original feasible SDP-relaxation and the optimal set is compact, this also holds for the SDP-relaxation of feasible subproblems. This result is important, since strong duality is necessary for primal-dual approaches, whose applicability is further discussed in Section 6. We then introduce a dual fixing method in Section 7 that allows to fix variables based on an incumbent value and exploiting integrality conditions. In Section 8, we present further solver components for a generic MISDP-solver and discuss preprocessing, branching rules, and primal heuristics. Section 9 discusses implementation issues. We demonstrate the applicability of this approach on three show-cases, which are introduced in Section 10. Finally, in Section 11, we investigate the influence of different SDP-solvers, branching rules, dual fixing, and primal heuristics as well as the occurrence of numerical difficulties in our implementation of a branch-and-bound algorithm for mixed-integer semidefinite programming.

The only existing similar project that we are aware of is YALMIP [49, 50], which is based on MATLAB and provides an interface to a very large number of solvers. However, its branch-and-bound part is not (yet) designed to be competitive. In comparison, our SCIP-SDP framework is written in C/C++ and is build on SCIP [67]. It can be extended by additional solver components and is available in source code.

2. Notation

Throughout this paper, we will use the following notation. The set of the first n positive integers is written as $[n] := \{1,\dots,n\}$. For some $x \in \mathbb{R}$, we write $\lfloor x \rfloor := \max\{n \in \mathbb{Z} : n \leq x\}$ for the largest integer less or equal to x and similarly $\lceil x \rceil$ for the smallest integer greater or equal to x. The cardinality of a set I is $|I|$. For $x \in \mathbb{R}^n$, we define $\|x\|_0 := |\{i \in [n] : x_i \neq 0\}|$ as the size of the support of x.

The set of all symmetric $n \times n$ matrices is denoted by $S_n \subset \mathbb{R}^{n\times n}$ and $A \bullet B := \sum_{i,j\in[n]} A_{ij}B_{ij} = \mathrm{Tr}(A \cdot B)$ is the standard inner product on S_n, where $\mathrm{Tr}(X) := \sum_{i=1}^n X_{ii}$ is the trace of a matrix X. We write $A \succeq 0$ if the matrix A is symmetric and positive semidefinite and $A \succ 0$ if it is symmetric and positive definite. The $\frac{1}{2}n(n+1)$-dimensional vector of all upper triangular entries of $X \in S_n$ is written as $\mathrm{vec}(X)$. For $y \in \mathbb{R}^n$, we write $\mathrm{Diag}(y)$ for the $n \times n$ matrix having entries of y on the diagonal and 0 otherwise. Conversely, $\mathrm{diag}(X)$ is the vector consisting of all diagonal entries of X. For a matrix $X \in S_n$, $\|X\|_\infty := \max_{i,j\in[n]} |X_{ij}|$ is the maximum norm and $\|X\|_F := (\sum_{i,j\in[n]} |X_{ij}|^2)^{1/2}$ defines the Frobenius norm. For any index-set $I \subseteq [n]$ of size k and a vector $x \in \mathbb{R}^n$, x_I is the k-dimensional vector of all components of x indexed by I. Similarly, for $X \in S_n$, the matrix X_I is the $k \times k$ matrix of all entries of X with indices in $I \times I$. Furthermore, $I_n \in \mathbb{R}^{n\times n}$ is the identity matrix. For some condition B, the indicator function $1_B \in \{0,1\}$ is 1 if and only if B is true.

Moreover, e_i is the i-th unit vector in $\mathbb{R}^n$. For a vector space X, $\dim(X)$ is its dimension, and for some subset $S \subseteq X$, the linear span is written as $\mathrm{span}\,(S) := \{y \in X : \exists k \in \mathbb{N}, \lambda \in \mathbb{R}^k, x_1,\dots,x_k \in S, y = \sum_{i=1}^k \lambda_i x_i\}$. The range of an operator $\mathscr{A} : X \to Y$ is denoted by $\mathrm{Range}(\mathscr{A}) := \{\mathscr{A}(x) : x \in X\}$, and the preimage of $y \in Y$ is given by $\mathscr{A}^{-1}(y) := \{x \in X : \mathscr{A}(x) = y\}$.

3. Mixed-Integer Semidefinite Programs

In this paper, we discuss a framework for solving mixed-integer semidefinite programs (MISDP) of the form

$$\text{(D-MISDP)} \qquad \begin{aligned} \sup \quad & b^\top y \\ \text{s.t.} \quad & C - \sum_{i=1}^{m} A_i y_i \succeq 0, \\ & \ell_i \leq y_i \leq u_i \qquad \forall i \in [m], \\ & y_i \in \mathbb{Z} \qquad \forall i \in I, \end{aligned}$$

with $C \in S_n$, $b \in \mathbb{R}^m$, and $A_i \in S_n$, $\ell_i \in \mathbb{R} \cup \{-\infty\}$, $u_i \in \mathbb{R} \cup \{\infty\}$ for all $i \in [m]$. The set of indices of integer variables is given by $I \subseteq [m]$. The model (D-MISDP) arises by adding integrality constraints to a semidefinite program written in a form usually referred to as "dual". One could also consider the primal form:

$$\text{(P-MISDP)} \qquad \begin{aligned} \inf \quad & C \bullet X \\ \text{s.t.} \quad & A_i \bullet X = b_i \quad \forall i \in [m], \\ & X \succeq 0, \\ & X_{ij} \in \mathbb{Z} \qquad \forall i \in J, \end{aligned}$$

for a given set $J \subseteq [n] \times [n]$ of indices for the integrality constraints, and possibly with additional variable bounds. Using standard techniques, (D-MISDP) can be transformed to (P-MISDP) and conversely. Thus, as usual, it is somewhat arbitrary which model is primal or dual. We will use the dual form as the primary form in order to be consistent with most parts of the literature.

4. SDP-based Branch-and-Bound Algorithm

Mixed-integer semidefinite programs can be solved with a straight-forward branch-and-bound algorithm. Branch-and-bound was first proposed for integer linear programs by Land and Doig [46] in 1960. Already in 1965, Dakin [21] realized that the problems do not have to be linear, but that the same technique can also be applied to general problems with integrality constraints, as long as the subproblems obtained by relaxing the integrality conditions and possibly adding variable bounds can be solved to optimality. Applying this algorithm to mixed-integer semidefinite programs leads to Algorithm 1. Its basic properties can be formulated as follows.

Proposition 1. *If all problems in Step 4 can be solved to optimality or proven infeasibility and the set S is bounded, Algorithm 1 terminates after a finite number of steps with either the optimal solution or $L = -\infty$, if* (D-MISDP) *is infeasible.*

Remark 1.

(1) One of the requirements for Algorithm 1 is that one can compute optimal solutions $\hat{y}$ in Step 4, in particular, the optimal value is attained. This can theoretically be guaranteed, for instance, if the subproblems for $\hat{S}$ have a compact optimality set; see Section 5 for a discussion.

(2) The boundedness of S can, for instance, be guaranteed, if all variable bounds ℓ_i and u_i are finite. Note that Algorithm 1 need not terminate if the relaxation set S is unbounded, even in the special case of mixed-integer linear programs, e.g., for $\min\{x : x + y = \frac{1}{2},\ x \in \mathbb{Z}_+,\ y \in \mathbb{Z}\}$.

Algorithm 1: SDP-based branch-and-bound algorithm

Input: bounded MISDP $\sup\{b^\top y : y \in S,\ y_i \in \mathbb{Z}\ \forall i \in I\}$,
$S := \{y \in \mathbb{R}^m : C - \sum_{i=1}^m A_i y_i \succeq 0,\ \ell_i \le y_i \le u_i\}$
Output: optimal solution y^* or "infeasible" if $L = -\infty$

```
1  set T := {S}, L := −∞;
2  while T ≠ ∅ do
3      choose Ŝ ∈ T, set T := T \ {Ŝ};
4      solve sup{b^⊤ y : y ∈ Ŝ} with an optimal solution ŷ if it exists;
5      if ŷ exists then
6          if b^⊤ ŷ > L then
7              if ŷ_i ∈ ℤ for all i ∈ I then
8                  y* := ŷ, L := b^⊤ ŷ;
9              else
10                 choose i ∈ I with ŷ_i ∉ ℤ;
11                 Ŝ^≤ := {y ∈ Ŝ : y_i ≤ ⌊ŷ_i⌋}, Ŝ^≥ := {y ∈ Ŝ : y_i ≥ ⌈ŷ_i⌉};
12                 T := T ∪ {Ŝ^≤, Ŝ^≥};
```

(3) The handling of unbounded problems is delicate for mixed-integer semidefinite problems. In the mixed-integer linear case, it is possible to show that for every problem with finite optimal value an optimal solution with a polynomial size in the encoding length of the problem exists. Thus, one can make the problem bounded by adding corresponding bounds on the variables. See, e.g., Schrijver [66] for a discussion.

For bounded semidefinite programs, Porkolab and Khachiyan [60] proved the existence of an optimal solution with encoding length that is exponential in the encoding length of the problem. It remains an open problem whether these bounds can be extended to the mixed-integer case. We will therefore always assume that Algorithm 1 terminates, e.g., because the problem is bounded.

5. Strong Duality after Adding Inequalities

The main work when applying branch-and-bound arises in solving the semidefinite relaxations in Step 4. Here SDPs of the form

$$\text{(D)}\quad \begin{aligned} \sup\ & b^\top y \\ \text{s.t.}\ & C - \sum_{i=1}^m A_i y_i \succeq 0, \\ & y \in \mathbb{R}^m, \end{aligned} \qquad \text{(P)}\quad \begin{aligned} \inf\ & C \bullet X \\ \text{s.t.}\ & A_i \bullet X = b_i \quad \forall i \in [m], \\ & X \succeq 0. \end{aligned}$$

have to be solved, and the supremum needs to be attained to proceed with Step 6 of Algorithm 1. The bounds on the y variables are omitted in this formulation, since these can also be integrated in the semidefinite constraints as additional rows and columns, which are zero apart from a diagonal entry of $y_i - \ell_i$ or $u_i - y_i$. The resulting matrix will be positive semidefinite if and only if these diagonal entries are nonnegative and the remaining part of the matrix is positive semidefinite.

A common approach to solve these problems are interior-point methods, see, e.g., Nesterov and Nemirovskii [56] and Ye [76]. Typically, these algorithms require strong duality for validity or convergence guarantees. An important characteristic of semidefinite programs ensuring strong duality is the existence of points in the relative interior of either (P) or (D), the so-called *Slater condition*. This is due to the following proposition, with the main result originally proven by Bellman and Fan [9] for slightly different versions of (P) and (D):

Proposition 2 (see, e.g., Helmberg [39], Todd [71])**.** *Let p^* and d^* be the optimal objective values of (P) and (D). Then the following holds:*

(1) If (P) has a strictly feasible point $X^* \succ 0$ and (D) is feasible, then $p^* = d^*$ and this value is attained for (D). Furthermore, if the A_i are linearly independent, then the optimal set of (D) is compact.

(2) If (D) has a strictly feasible point y^* with $C - \sum_{i=1}^m A_i y_i^* \succ 0$ and (P) is feasible, then $p^* = d^*$ and this value is attained for (P). Furthermore, the optimal set of (P) is compact.

(3) If (P) and (D) both have strictly feasible points, then $p^* = d^*$ and this value is attained for both problems. Furthermore, the optimal set of (P) is compact, and if the A_i are linearly independent, then the optimal set of (D) is also compact.

If neither (P) nor (D) has a relative interior point, strong duality no longer holds and there might be a nonzero duality gap between primal and dual optimal solutions, see, e.g., [39, 69, 71]. If only one problem is strictly feasible, then this problem might not attain its optimal value and therefore the existence of a KKT-point is not guaranteed. For this reason, interior-point methods in semidefinite programming in general require the existence of strictly feasible solutions for both (P) and (D), see, e.g., [76].

A natural question is whether strong duality or even the existence of strictly feasible solutions is inherited from the predecessor subproblems in a branch-and-bound tree. In this section, we investigate the general question of inheritance when adding linear inequalities. While only the results in dual form (D-MISDP) are relevant for Algorithm 1, we will discuss results in both primal and dual form, since they are interesting from a theoretical point of view and some assumptions necessary for the dual form can be dropped for (P-MISDP). We start with discussing the inheritance of strong duality for problems in primal form (P-MISDP), before extending the results to the dual form (D-MISDP) via duality. We then continue with the inheritance of strictly feasible solutions.

If a linear inequality is added to (P-MISDP), the corresponding slack variable increases the dimension. Therefore, we will allow a general affine constraint $A_{m+1} \bullet X = b_{m+1}$ with $A_{m+1} \in S_{n+1}$ to be added to (P-MISDP), which changes the SDP-relaxation (P) to

$$
\begin{aligned}
(\mathrm{P}_+\text{-P}) \qquad & \inf \quad \begin{pmatrix} C & 0 \\ 0^\top & 0 \end{pmatrix} \bullet X, \\
& \text{s.t.} \quad \begin{pmatrix} A_i & 0 \\ 0^\top & 0 \end{pmatrix} \bullet X = b_i \quad \forall i \in [m], \\
& \qquad A_{m+1} \bullet X = b_{m+1}, \\
& \qquad X \succeq 0.
\end{aligned}
$$

The dual problem to (P_+-P) can then be written as

$$\begin{aligned} & \sup && b^\top y + b_{m+1} y_{m+1} \\ (P_+\text{-D}) \qquad & \text{s.t.} && \begin{pmatrix} C & 0 \\ 0^\top & 0 \end{pmatrix} - \sum_{i=1}^{m} \begin{pmatrix} A_i & 0 \\ 0^\top & 0 \end{pmatrix} y_i - A_{m+1} y_{m+1} \succeq 0, \\ & && y \in \mathbb{R}^{m+1}. \end{aligned}$$

In the following, we show that this change has no influence on strong duality and compactness of the set of optimal solutions, which implies that the supremum is attained and there exists an optimal solution. So if strong duality and compactness of the optimal set hold in the root node, e.g., because of strict feasibility of the root node relaxation, they will hold for all feasible nodes.

Theorem 1. *Let p^* be the optimal objective of (P) and d^* the optimal objective of (D). If $p^* = d^* > -\infty$, the optimal set of (P) is compact, and problem (P_+-P) is feasible, then the optimal set of (P_+-P) is also compact and $p_+^* = d_+^*$ also holds for the optimal objective values of (P_+-P) and (P_+-D), respectively.*

Proof. W.l.o.g. assume that

$$\inf\{C \bullet X : A_i \bullet X = b_i \; \forall i \in [m+1]\} < \inf\{C \bullet X : A_i \bullet X = b_i \; \forall i \in [m+1], X \succeq 0\},$$

since otherwise the redundant SDP-constraint can be dropped and the result follows from strong LP-duality.

Because of the finiteness of (P) and the feasibility of (P_+-P), the value of (P_+-P) is finite. Moreover, assume that there exists an unbounded minimizing sequence $(X_k) \subset S_{n+1}$ for (P_+-P). After restriction of X_k to S_n and possibly selecting a subsequence, $R := \lim_{k\to\infty} (X_k)_{[n]} / \|(X_k)_{[n]}\|_F$ satisfies $R \succeq 0$, $A_i \bullet R = 0$ for all $i \in [m]$, and $C \bullet R = 0$. Therefore, given an optimal solution $\tilde{X}$ of (P), $\tilde{X} + \lambda R$ is optimal for any $\lambda \geq 0$, contradicting the compactness of the optimal set of (P). Hence, all minimizing sequences are bounded and therefore the optimal set of (P_+-P) is nonempty, bounded and consequently compact.

Clearly, (P_+-D) is feasible and by weak duality $p_+^* \geq d_+^*$. Define the sets $G_1 := \{X \in S_{n+1} : X \succeq 0\}$ and $G_2 := \{X \in S_{n+1} : A_i \bullet X = b_i \, \forall i \in [m+1], C \bullet X \leq p_+^* - \varepsilon\}$ for small enough $\varepsilon > 0$, such that G_2 is feasible (because of the assumption of non-redundancy of the SDP-constraint, such an ε exists). These closed convex sets G_1 and G_2 are disjoint and

$$\inf\{\|Y - W\|_F : Y \in G_1, W \in G_2\} > 0$$

holds. Otherwise there would exist sequences

$$W_k = Y_k + U_k, \; Y_k \in G_1, \; W_k \in G_2, \; U_k \to 0.$$

If (W_k) is bounded, we get $W := \lim_{k\to\infty} W_k$ feasible for (P_+-P) with $C \bullet W \leq p_+^* - \varepsilon$, which is impossible. If (W_k) is unbounded, after restriction to S_n and selecting a subsequence, we obtain $R := \lim_{k\to\infty} (W_k)_{[n]} / \|(W_k)_{[n]}\|_F$ satisfying $R \succeq 0$, $A_i \bullet R = 0$ for all $i \in [m]$, and $C \bullet R = 0$. This again contradicts the compactness of the optimal set of (P). Hence, G_1 and G_2 have no common direction of recession and thus there exists a strictly separating hyperplane, i.e., there are $S \in S_{n+1}$, $\sigma \in \mathbb{R}$ such that

$$S \bullet X > \sigma \quad \forall X \in G_1, \quad S \bullet X \leq \sigma \quad \forall X \in G_2.$$

Since $0 \in G_1$, we must have $\sigma < 0$. Therefore, $S \bullet X \geq 0$ for all $X \succeq 0$, which, by self-duality of the cone of positive semidefinite matrices, implies $S \succeq 0$.

Hence,

$$S \bullet X \leq \sigma \quad \forall X \text{ with } A_i \bullet X = b_i,\ i = 1,\dots m+1,\ C \bullet X \leq p_+^* - \varepsilon,$$

and the optimal value of the LP

$$\max \{S \bullet X : A_i \bullet X = b_i,\ i = 1,\dots,m+1,\ C \bullet X \leq p_+^* - \varepsilon\}$$

is at most σ. The dual of this LP is

$$\min \{(p_+^* - \varepsilon)t - b^\top z : Ct - \sum_{i=1}^{m+1} A_i z_i = S,\ t \geq 0\}.$$

By strong LP-duality, there exist $y \in \mathbb{R}$, $\eta \geq 0$ such that

$$C\eta - \sum_{i=1}^{m+1} A_i y_i = S, \quad (p_+^* - \varepsilon)\eta - b^\top y \leq \sigma.$$

Let $\hat{X}$ be an optimal solution of (P$_+$-P). We must have $\eta > 0$, since for $\eta = 0$ we have $-b^\top y \leq \sigma < 0$ and

$$-b^\top y = -(A_1 \bullet \hat{X}, \dots, A_{m+1} \bullet \hat{X})^\top y = \hat{X} \bullet (-\sum_{i=1}^{m+1} A_i y_i) = \hat{X} \bullet S \geq 0,$$

which is a contradiction. Thus, $\eta > 0$ and we can define $\hat{y} = y/\eta$. Then

$$C - \sum_{i=1}^{m+1} A_i \hat{y}_i = \tfrac{1}{\eta} S \succeq 0, \quad b^\top \hat{y} \geq p_+^* - \varepsilon - \sigma/\eta \geq p_+^* - \varepsilon.$$

Thus, $\hat{y}$ is feasible for (P$_+$-D) with objective function value $\geq p_+^* - \varepsilon$. By taking $\varepsilon \to 0$, this shows $p_+^* = d_+^*$. □

By choosing

$$A_{m+1} = \begin{pmatrix} \tilde{A}_{m+1} & 0 \\ 0^\top & 1 \end{pmatrix}, \qquad A_{m+1} = \begin{pmatrix} \tilde{A}_{m+1} & 0 \\ 0^\top & -1 \end{pmatrix}, \quad \text{or} \quad A_{m+1} = \begin{pmatrix} \tilde{A}_{m+1} & 0 \\ 0^\top & 0 \end{pmatrix},$$

Theorem 1 allows to add inequalities $\tilde{A}_{m+1} \bullet \tilde{X} \leq b_{m+1}$, $\tilde{A}_{m+1} \bullet \tilde{X} \geq b_{m+1}$, or the equation $\tilde{A}_{m+1} \bullet \tilde{X} = b_{m+1}$ for $\tilde{A}_{m+1} \in S_n$:

Corollary 1. *Let p^* be the optimal objective of (P) and d^* the optimal objective of (D). If $p^* = d^* > -\infty$, the optimal set of (P) is compact, and the problem (P$_+$), generated by adding a constraint $\tilde{A}_{m+1} \bullet X \leq b_{m+1}$, $\tilde{A}_{m+1} \bullet X \geq b_{m+1}$, or $\tilde{A}_{m+1} \bullet X = b_{m+1}$ for $\tilde{A}_{m+1} \in S_n$ to (P), is feasible, then strong duality holds for (P$_+$) and its dual. Moreover, the optimal set of (P$_+$) is compact.*

Remark 2. The compactness of the optimal set of (P) is not only necessary to ensure compactness of the optimal set after branching, but also to keep strong duality, as shown by the example given by Friberg [31] for a mixed-integer second order cone problem, i.e., a special case of MISDP.

Via duality, the above results can also be extended to (D-MISDP), except that we have to restrict the optimal set to $Z := C - \sum_{i=1}^m A_i y_i$ to obtain compactness. In this case, we want to show that strong duality still holds after adding a linear constraint $\sum_{i=1}^m a_i y_i \leq c$, $\sum_{i=1}^m a_i y_i \geq c$, or $\sum_{i=1}^m a_i y_i = c$ to (D). For example, in the case of $\sum_{i=1}^m a_i y_i \leq c$, we obtain

$$
\begin{array}{lll}
\sup & b^\top y & \\
\text{s.t.} & C - \sum_{i=1}^m A_i y_i \succeq 0, & \\
& \sum_{i=1}^m a_i y_i \le c, & \\
& y \in \mathbb{R}^m, &
\end{array}
\qquad (\mathrm{D}_+\text{-D})
\qquad
\begin{array}{lll}
\inf & C \bullet X + c\, x_{n+1} & \\
\text{s.t.} & A_i \bullet X + a_i x_{n+1} = b_i & \forall i \in [m], \\
& \begin{pmatrix} X_{11} & \dots & X_{1n} & x_1 \\ \vdots & \ddots & \vdots & \vdots \\ X_{1n} & \dots & X_{nn} & x_n \\ x_1 & \dots & x_n & x_{n+1} \end{pmatrix} \succeq 0, &
\end{array}
\qquad (\mathrm{D}_+\text{-P})
$$

while for the equality constraint we would have to apply the procedure twice, once with positive and once with negative sign.

Defining

$$\hat{A}_i = \begin{pmatrix} A_i & 0 \\ 0^\top & a_i \end{pmatrix}, \quad \hat{C} = \begin{pmatrix} C & 0 \\ 0^\top & c \end{pmatrix},$$

the dual constraint of (D$_+$-D) can be written as $\hat{C} - \sum_{i=1}^m \hat{A}_i y_i \succeq 0$.

Theorem 2. *Let p^* be the optimal objective of (P) and d^* the optimal objective of (D). If $p^* = d^* < \infty$, the set of optimal $Z = C - \sum_{i=1}^m A_i y_i$ of (D) is compact, and (D$_+$-D) is feasible, then strong duality holds for (D$_+$-D) and (D$_+$-P). Moreover, the set of optimal $\hat{Z} = \hat{C} - \sum_{i=1}^m \hat{A}_i y_i$ of (D$_+$-D) is compact.*

Proof. To apply Theorem 1 to (D) and (D$_+$-D), we need to transform them to primal form. To do so, we will use the same technique as Todd in [71]. Since (P) is feasible, choose $D \in S_n$ such that $A_i \bullet D = b_i$ for all $i \in [m]$, let $G_1, \dots, G_k$ be a basis of the orthogonal complement of span$(A_1, \dots, A_m)$ in S_n, and define $h_i := C \bullet G_i$ for all $i \in [k]$. Then (D) is equivalent to

$$
\begin{array}{lll}
C \bullet D - \inf & D \bullet Z & \\
\text{s.t.} & G_i \bullet Z = h_i & \forall i \in [k], \\
& Z \succeq 0, &
\end{array}
\qquad (\text{T-P})
$$

since

$$
\begin{aligned}
G_i \bullet Z = h_i \quad \forall i \in [k] \quad &\Leftrightarrow \quad G_i \bullet (Z - C) = 0 \quad \forall i \in [k] \\
&\Leftrightarrow \quad Z - C \in \operatorname{span}(A_1, \dots, A_m).
\end{aligned}
$$

Thus, $Z \succeq 0$ is feasible for (T-P) if and only if $Z = C - \sum_{i=1}^m A_i y_i$ for some $y \in \mathbb{R}^m$ and

$$C \bullet D - D \bullet \Big(C - \sum_{i=1}^m A_i y_i\Big) = C \bullet D - \Big(D \bullet C - \sum_{i=1}^m D \bullet A_i y_i\Big) = \sum_{i=1}^m b_i y_i = b^\top y.$$

This implies that (T-P) is feasible if and only if (D) is feasible. Furthermore, the optimal set of (T-P) is compact if and only if the set of optimal $Z = C - \sum_{i=1}^m A_i y_i$ for (D) is compact and the optimal objective values agree. The dual problem to (T-P) can be written as

$$
\begin{array}{lll}
C \bullet D - \sup & h^\top w & \\
\text{s.t.} & D - \sum_{i=1}^k G_i w_i \succeq 0, &
\end{array}
\qquad (\text{T-D})
$$

which is equivalent to (P), since

$$\begin{aligned} A_i \bullet X = b_i \quad \forall i \in [k] \quad &\Leftrightarrow \quad A_i \bullet (X - D) = 0 \quad \forall i \in [k] \\ &\Leftrightarrow \quad X - D \in \operatorname{span}(G_1, \ldots, G_m). \end{aligned}$$

Thus, X is feasible for the equality constraint in (P) if and only if $X = D - \sum_{i=1}^{k} G_i w_i$ for some $w \in \mathbb{R}^m$, and

$$C \bullet X = C \bullet D - \sum_{i=1}^{k} (C \bullet G_i) w_i = C \bullet D - h^\top w.$$

When adding the linear inequality $\sum_{i=1}^{m} a_i y_i \leq c$ to (D), we have $\hat{Z} \in S_{n+1}$ in (D_+-D) instead of $Z \in S_n$ in (D). Thus, the dimension of $\operatorname{span}\left(\hat{A}_1, \ldots, \hat{A}_n\right)^\perp \subseteq S_{n+1}$ is larger than the dimension of $\operatorname{span}(A_1, \ldots, A_n)^\perp \subseteq S_n$ by at most $n+1$. Furthermore, there exists a basis $\hat{G}_1, \ldots, \hat{G}_{\hat{k}}$ of the orthogonal complement of the span, such that the first k matrices $\hat{G}_i$ equal G_i, except for an added zero row and column. For this reason, also $\hat{h}_i = h_i$ for $i = 1, \ldots, k$. Since b does not change, we can also choose $\hat{D}$ as D with added zero row and column. Thus, the new problem (T-P_+) arises by adding at most $n+1$ constraints $\hat{G}_i \bullet \hat{Z} = \hat{h}_i$, $i = k+1, \ldots, \hat{k}$, to (T-P), while the remaining problem is only extended by zero rows and columns. Therefore, Theorem 1 can be applied for each added $\hat{G}_i$. Thus, strong duality holds for (T-P_+) and (T-D_+), and the optimal set of (T-P_+) is compact, which directly transfers to (D_+-D) if we restrict the optimal set to $\hat{Z} = \hat{C} - \sum_{i=1}^{m} \hat{A}_i y_i$. □

Remark 3. From Theorem 2 we only get compactness of the set of optimal $\hat{Z} = \hat{C} - \sum_{i=1}^{m} \hat{A}_i y_i$, but not of the set of optimal y_i. To also ensure compactness of this set, we further need to assume that the $\hat{A}_i$ are linearly independent, since in this case the y_i are uniquely determined by $\hat{Z}$, see Todd [71].

While the compactness of the optimal set suffices to guarantee that it is possible to solve the problems in Step 4 of Algorithm 1 to optimality, in practice, one might want to solve these problems using interior-point solvers. Since the convergence proofs of most interior-point solvers for SDPs require the existence of strictly feasible solutions for the primal and dual problem, we will now turn to the question of inheritance of such solutions. If instead of strong duality and compactness of the set of optimal Z, we require the stronger condition of strict feasibility for the dual problem, we can also show that strict feasibility is maintained after adding a linear inequality or equation.

Proposition 3. *Assume that (D) is strictly feasible. Then the dual of the problem obtained by adding a linear constraint $A_{m+1} \bullet X = b_{m+1}$, $A_{m+1} \bullet X \geq b_{m+1}$, or $A_{m+1} \bullet X \leq b_{m+1}$ to (P) is strictly feasible as well.*

Proof. After adding an equality constraint to (P), the dual problem reads

$$\text{($P_=$-D)} \qquad \begin{aligned} \sup \quad & b^\top y \\ \text{s.t.} \quad & C - \sum_{i=1}^{m+1} A_i y_i \succeq 0, \\ & y \in \mathbb{R}^{m+1}. \end{aligned}$$

Thus, if $\hat{y}$ is strictly feasible for (D), then $(\hat{y}, 0)$ is strictly feasible for ($P_=$-D).

Now assume that we w.l.o.g. add an inequality constraint $A_{m+1} \bullet X \geq b_{m+1}$. Then we have to add a slack variable to (P), and the dual problem becomes

$$\text{(P}_{\geq}\text{-D)} \qquad \begin{array}{ll} \sup & b^\top y \\ \text{s.t.} & \begin{pmatrix} C & 0 \\ 0^\top & 0 \end{pmatrix} - \sum_{i=1}^{m+1} \begin{pmatrix} A_i & 0 \\ 0^\top & 0 \end{pmatrix} y_i - \begin{pmatrix} 0 & 0 \\ 0^\top & -1 \end{pmatrix} y_{m+1} \succeq 0, \\ & y \in \mathbb{R}^{m+1}. \end{array}$$

Let $\hat{y}$ be a strictly feasible solution of (D). Then a strictly feasible solution to ($\text{P}_{\geq}$-D) is given by $(\hat{y}, \delta)$ for sufficiently small $\delta > 0$, since continuity implies $C - \sum_{i=1}^m A_i \hat{y}_i - A_{m+1}\delta \succ 0$ and therefore also

$$\begin{pmatrix} C - \sum_{i=1}^m A_i \hat{y}_i - A_{m+1}\delta & 0 \\ 0^\top & \delta \end{pmatrix} \succ 0. \qquad \square$$

For strict feasibility we cannot extend the results easily via duality. In this case, we need an additional assumption to prove the corresponding result for the primal problem:

Proposition 4. *Define $\mathscr{A} : S_n \to \mathbb{R}^m$, $X \mapsto (A_i \bullet X)_{i\in[m]}$. If (P) is strictly feasible and $a \in \text{Range}(\mathscr{A})$, then the primal of the problem formed by adding a linear constraint $\sum_{i=1}^m a_i y_i \leq c$, $\sum_{i=1}^m a_i y_i \geq c$, or $\sum_{i=1}^m a_i y_i = c$ to (D) is strictly feasible as well.*

Proof. We will only show the case $\sum_{i=1}^m a_i y_i \leq c$; the other cases are obtained by changing the sign of a and c or by adding both inequalities in the case of the equality constraint. The modified dual and primal problems are then (D_+-D) and (D_+-P) from above, respectively. Let $\hat{X}$ be a strictly feasible solution for (P). Then a strictly feasible solution $(\tilde{X}(\delta), \tilde{x}(\delta))$ for (D_+-P) is given by $\tilde{x}(\delta)_{n+1} = \delta$, $\tilde{x}(\delta)_i = 0$ for $i \in [n]$, $\tilde{X}(\delta) = \hat{X} - \delta Y$ for some $Y \in \mathscr{A}^{-1}(a)$ and sufficiently small δ, since

$$\begin{aligned} (A_i \bullet \tilde{X}(\delta) + a_i \tilde{x}(\delta)_{n+1})_{i\in[m]} &= \mathscr{A}(\tilde{X}(\delta)) + \tilde{x}(\delta)_{n+1} \cdot a = \mathscr{A}(\hat{X} - \delta Y) + \delta a \\ &= \mathscr{A}(\hat{X}) - \delta \mathscr{A}(Y) + \delta a = b - \delta a + \delta a = b, \end{aligned}$$

and by continuity also

$$\begin{pmatrix} \hat{X} - \delta Y & 0 \\ 0^\top & \delta \end{pmatrix} \succ 0. \qquad \square$$

Remark 4.

(1) The assumption $a \in \text{Range}(\mathscr{A})$ in Proposition 4 is necessary, even for linear programs, as the following example shows:

$$\text{(1-D)} \quad \begin{array}{ll} \sup & 2y_1 + 4y_2 \\ \text{s.t.} & y_1 + 2y_2 \leq 1, \end{array} \qquad \text{(1-P)} \quad \begin{array}{ll} \inf & x_1 \\ \text{s.t.} & x_1 = 2, \\ & 2x_1 = 4, \\ & x_1 \geq 0. \end{array}$$

A strictly feasible solution for (1-D) is given by $y = (0.1, 0.1)$. Indeed, we have $1 - y_1 - 2y_2 = 0.7 > 0$. Moreover, $x_1 = 2 > 0$ is strictly feasible for (1-P). But if we add the constraint $y_1 \leq 1$, we get

$$(1_+\text{-D}) \quad \begin{aligned} \sup \quad & 2y_1 + 4y_2 \\ \text{s.t.} \quad & y_1 + 2y_2 \leq 1, \\ & y_1 \leq 1, \end{aligned} \qquad (1_+\text{-P}) \quad \begin{aligned} \inf \quad & x_1 + x_2 \\ \text{s.t.} \quad & x_1 + x_2 = 2, \\ & 2x_1 = 4, \\ & x_1,\, x_2 \geq 0. \end{aligned}$$

The only remaining feasible solution for (1_+-P) is $x = (2, 0)$, which is no longer strictly feasible. In this example, the assumption of Proposition 4 does not hold, since we have

$$a = \left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right) \notin \operatorname{span}\left(\left(\begin{smallmatrix} 1 \\ 2 \end{smallmatrix}\right)\right) = \operatorname{Range}(\mathscr{A}).$$

(2) The assumption $a \in \operatorname{Range}(\mathscr{A})$ is implied by linear independence of the matrices $A_1, \ldots, A_m$, because then $\dim(\operatorname{Range}(\mathscr{A})) = m$, i.e., $\operatorname{Range}(\mathscr{A}) = \mathbb{R}^m$. Assuming linear independence of the matrices is a natural assumption in the context of semidefinite programs, since it is also assumed by interior-point codes to ensure the existence of unique solutions for the Newton-steps on the central path.

Remark 5. Proposition 4 (and equivalently Proposition 3) only ensures strict feasibility in the primal problem, not for the dual problem. Therefore, it does not necessarily ensure the existence of a Karush-Kuhn-Tucker-Point, since the primal problem might not attain its optimum if the dual problem does not contain a relative interior point. This is shown by the following example, which is an extension of a well-known example for SDP-Duality (see, e.g., [39, Example 2.2.8] or [69, Example 4.1.1]):

$$(2\text{-D}) \quad \begin{aligned} \sup \quad & 2y_1 - y_2 \\ \text{s.t.} \quad & \begin{pmatrix} 0.5 & -y_1 \\ -y_1 & y_2 \end{pmatrix} \succeq 0, \end{aligned} \qquad (2\text{-P}) \quad \begin{aligned} \inf \quad & 0.5 X_{11} \\ \text{s.t.} \quad & \begin{pmatrix} X_{11} & 1 \\ 1 & 1 \end{pmatrix} \succeq 0, \end{aligned}$$

where the primal results from the constraints $X_{12} + X_{21} = 2$ and $-X_{22} = -1$. In this case, there exists a strictly feasible point given by $X_{11} = 2$ with eigenvalues 0.38 and 2.62 for the primal and $y = (0, 0.5)$ for the dual, i.e., the scaled identity matrix. Thus, strong duality holds with an optimal objective value of 0.5, which is obtained for $X_{11} = 1$ and $y = (0.5, 0.5)$.

Now assume that y_2 should be integral and we therefore want to branch on y_2. Adding the constraint $y_2 \leq 0$ through a new diagonal entry $-y_2$ in (2-D) changes the primal constraint corresponding to y_2 from $-X_{22} = -1$ to $-X_{22} + X_{33} = -1$, which leads to the dual problem

$$(2_+\text{-D}) \quad \begin{aligned} \sup \quad & 2y_1 - y_2 \\ \text{s.t.} \quad & \begin{pmatrix} 0.5 & -y_1 & 0 \\ -y_1 & y_2 & 0 \\ 0 & 0 & -y_2 \end{pmatrix} \succeq 0 \end{aligned}$$

and the primal problem

$$(2_+\text{-P}) \quad \begin{aligned} \inf \quad & 0.5 X_{11} \\ \text{s.t.} \quad & \begin{pmatrix} X_{11} & 1 & X_{13} \\ 1 & X_{22} & X_{23} \\ X_{13} & X_{23} & X_{22} - 1 \end{pmatrix} \succeq 0. \end{aligned}$$

The two diagonal entries for y_2 imply $y_2 = 0$ and therefore also $y_1 = 0$. Thus, the dual no longer has a strictly feasible solution, and the optimal objective value for the dual problem becomes 0. The primal problem still has a relative interior solution, e.g., $X_{11} = X_{22} = 2$, $X_{13} = X_{23} = 0$ with eigenvalues 1, 1, and 3. Thus, strong duality still holds by Proposition 2 and the infimum has to be zero, which is indeed the case since the sequence $X_{11} = 1/k$, $X_{22} = k$, $X_{13} = X_{23} = 0$ is feasible for all $k \geq 1$ and has objective value $1/(2k) \to 0$.

However, an optimal solution with objective 0 would have $X_{11} = 0$, which leads to a principle minor of $0 \cdot X_{22} - 1^2 = -1$ for the index set $\{1,2\}$, i.e., such a solution cannot be positive semidefinite. It follows that after adding a linear constraint to the above problem with primal and dual strictly feasible solutions, the resulting problem does not have a KKT-point, since the primal optimal objective is no longer attained.

6. Solving the SDP-Relaxations

We have seen in the last section that all feasible SDP-relaxations appearing in a branch-and-bound tree attain their optimal value, as long as the root node does, so they can in theory be solved to optimality. However, we have also seen that the existence of KKT-points may get lost by branching. Furthermore, the results in Section 5 assume the resulting problems to be feasible, while in a branch-and-bound approach infeasible problems will appear frequently, so detecting infeasibility is also an important task.

In this section, we will discuss checking feasibility of semidefinite programs and how to solve SDPs without both primal and dual relative interior points using interior-point solvers. Finally, we will discuss different solution approaches for semidefinite relaxations apart from interior-point methods. In the following, we will concentrate on semidefinite programs in dual form, since this is the type of problem we have to solve in Algorithm 1.

6.1. Interior-Point SDP-Solvers. Interior-point methods are the most frequently used type of algorithms to solve SDPs. However, most interior-point SDP-solvers are not tuned for usage in a branch-and-bound process, and, as discussed in Section 5, the constraint qualifications under which the solvers guarantee convergence may not be satisfied. Another problem of many SDP-solvers and interior-point solvers, in particular, is that they are not tuned to detect infeasibility. Since infeasible problems cannot satisfy the dual Slater condition by definition, such problems can lead to difficulties for the solvers.

One possibility to check the feasibility of a semidefinite relaxation is to use an idea in [51] and solve

$$\text{(FC)} \qquad \begin{aligned} \inf \quad & r \\ \text{s.t.} \quad & C - \sum_{i=1}^{m} A_i y_i + I_n r \succeq 0, \\ & \ell_i \leq y_i \leq u_i \qquad\qquad \forall i \in [m]. \end{aligned}$$

If problem (D) has a feasible solution $\tilde{y}$, then $(\tilde{y}, 0)$ is feasible for (FC), so its optimal objective is nonpositive. On the other hand, if the optimal objective of (FC) is strictly positive, then no y with $C - \sum_{i=1}^{m} A_i y_i \succeq 0$ exists. Thus, (D) is infeasible

and the node can be cut off. Note, however, that the solution of (FC) might fail, since changing the objective from (D) to (FC) can harm the primal Slater condition.

If (FC) showed that the problem is feasible, or we failed to solve (FC), a new upper bound on (D) and preferably even a feasible solution is needed. We have seen in Section 5, that primal strict feasibility is preserved by branching, but strictly feasible solutions in the dual problem may get lost. To also ensure the dual Slater condition, a penalty formulation can be used as, for instance, explained by Benson and Ye [14]. In this case, we solve the problems

$$\text{(}\Gamma\text{-P)}\quad \begin{aligned} \inf\quad & C \bullet X \\ \text{s.t.}\quad & A_i \bullet X = b_i \quad \forall i \in [m], \\ & I_n \bullet X \leq \Gamma, \\ & X \succeq 0, \end{aligned} \qquad \text{(}\Gamma\text{-D)}\quad \begin{aligned} \sup\quad & b^\top y - \Gamma r \\ \text{s.t.}\quad & C - \sum_{i=1}^{m} A_i y_i + I_n r \succeq 0, \\ & y \in \mathbb{R}^m,\ r \geq 0 \end{aligned}$$

for some $\Gamma > 0$ instead of (P) and (D). Then (Γ-D) has a strictly feasible solution $y = 0$ and r larger than the negative of the smallest eigenvalue of C. The Slater condition for the primal problem is also preserved, as long as $\Gamma \geq I_n \bullet X^* = \operatorname{Tr}(X^*)$, where X^* is a strictly feasible primal solution for (P). Such a solution is guaranteed by Proposition 4, if the root node relaxation is strictly feasible.

Since (Γ-D) and (Γ-P) both satisfy the Slater condition, (Γ-D) can now provably be solved to optimality by interior-point algorithms. This yields an upper bound on the value of (D) and therefore also (D-MISDP) for any Γ, since the feasible set of (D) is contained in that of (Γ-D) and the objective functions agree on the feasible set of (D). On the other hand, the feasible set of (Γ-D) includes further solutions with $r > 0$, which may lead to a gap between (D) and (Γ-D). If an optimal solution for (Γ-D) has $r = 0$, however, then this solution is also feasible for (D), so the optimal objective values of (Γ-D) and (D) are equal in this case.

If $r > 0$, by complementarity $\operatorname{Tr}(X) = \Gamma$. We can therefore try to increase Γ to force the solutions of (Γ-D) towards feasibility for (D). This is done until a feasible solution for our original problem is obtained and therefore the optimal objective values of (D) and (Γ-D) agree. However, we cannot guarantee this for all relaxations, as the following example shows:

Example 1. The primal and dual optimal objective value of (2_+-P) from Section 5

$$\text{(}2_+\text{-P)}\quad \begin{aligned} \inf\quad & 0.5 X_{11} \\ \text{s.t.}\quad & \begin{pmatrix} X_{11} & 1 & X_{13} \\ 1 & X_{22} & X_{23} \\ X_{13} & X_{23} & X_{22} - 1 \end{pmatrix} \succeq 0, \end{aligned}$$

with dual problem

$$\text{(}2_+\text{-D)}\quad \begin{aligned} \sup\quad & 2y_1 - y_2 \\ \text{s.t.}\quad & \begin{pmatrix} 0.5 & -y_1 & 0 \\ -y_1 & y_2 & 0 \\ 0 & 0 & -y_2 \end{pmatrix} \succeq 0. \end{aligned}$$

is zero. When solving the problem using the penalty formulation (Γ-P) and (Γ-D), the linear constraint $X_{11} + X_{22} + X_{22} - 1 \leq \Gamma$ is added to the primal problem. The primal (and dual) objective value, however, is $\frac{1}{4}(\Gamma + 1) - \frac{1}{2}\sqrt{(\Gamma+1)^2/4 - 2}$ for

$\Gamma > -1+2\sqrt{2}$, which will never reach zero. Since the computed bound gets better with increasing Γ, it might still be worthwhile to increase Γ in case of $\mathrm{Tr}(X^*) = \Gamma$.

Since Example 1 shows that the described approach is not guaranteed to terminate, a practical implementation will have to stop at a given threshold for the penalty parameter. In this case, the best solution found so far can still be used as an upper bound for (D). Even if neither the original problem nor the penalty formulation produced a valid bound, one can still continue if the current node is not the root node: One can take the bound of the parent node and proceed with branching. If the problems occur in the root node, the solution process has to terminate.

6.2. Alternative Solution Approaches. One alternative approach to handle SDPs is by outer approximation via linear programs, in which cutting planes are dynamically added. This approach is followed, e.g., by Krishnan and Mitchell [44, 45] and Mars [51]. The advantage of this approach is that very efficient linear programming solvers and, in particular, warm-starting of the dual simplex method can be used, while warm-starting interior-point methods is much more difficult. Furthermore, it is independent of the existence of relative interior points. The disadvantage is that, in general, an exponential number of cutting planes is needed for an ε-approximation, see Braun et al. [19]. Depending on the problem type, this is also confirmed by slow performance in practice, see [51].

A second approach is the spectral bundle method of Helmberg and Rendl [41], which is applicable to semidefinite programs with constant trace, i.e., $\mathrm{Tr}(X)$ is constant for all feasible points X of (P). This method is regularly used for solving max-cut and partitioning problems that satisfy this condition, see, e.g., [6, 7, 62]. One advantage of this approach is its warm-starting capability. One could therefore automatically switch to the spectral bundle method, if the constant-trace property is satisfied.

7. Dual Fixing

In mixed-integer linear programming, the good performance of branch-and-cut algorithms mainly results from many different techniques used for strengthening the relaxations. One such method is dual (or reduced cost) fixing, see, e.g., Nemhauser and Wolsey [55].

The general idea of reduced cost fixing for mixed-integer linear programs is that the reduced costs describe the per unit change of the objective value when increasing the value of a non-basic variable. Therefore, if the (non-positive) reduced cost of a binary variable at its lower bound is smaller than the difference between the objective of the best known integer solution and the current relaxation, the binary variable has to be zero for any optimal solution and can therefore be removed from the problem.

In this section, we extend this method to mixed-integer semidefinite programs. However, since no reduced costs are available in this context, the transfer is not direct. For linear programs solved by interior-point methods, Mitchell [53] already extended reduced cost fixing by using the dual variables corresponding to the variable bounds instead of the reduced costs. Helmberg [38] extended dual fixing to primal (mixed-integer) semidefinite programs, and Vigerske [72] proposed it for general mixed-integer nonlinear programs. In the following, we apply dual fixing specifically to the mixed-integer dual semidefinite problem (D-MISDP).

In this section, variable bounds are handled separately from the linear and semidefinite constraints in the relaxation of (D-MISDP). Therefore, define the sets

$$J_\ell := \{i \in [m] : \ell_i > -\infty\}, \quad J_u := \{i \in [m] : u_i < \infty\}$$

with $m_\ell := |J_\ell|$, $m_u := |J_u|$. The primal of the SDP-relaxation of (D-MISDP) is

$$\text{(B-P)} \qquad \begin{aligned} \inf \quad & C \bullet X + \sum_{i \in J_u} u_i V_{ii} - \sum_{i \in J_\ell} \ell_i W_{ii} \\ \text{s.t.} \quad & A_i \bullet X + 1_{\{i \in J_u\}} V_{ii} - 1_{\{i \in J_\ell\}} W_{ii} = b_i \quad \forall i = 1, \dots, m, \\ & \begin{pmatrix} X & U_1 & U_2 \\ U_1 & V & U_3 \\ U_2 & U_3 & W \end{pmatrix} \succeq 0, \end{aligned}$$

using the indicator function and with $V \in \mathbb{R}^{J_u \times J_u}$, $W \in \mathbb{R}^{J_\ell \times J_\ell}$. This formulation results from (P) and (D) via the $(n+m_u+m_\ell) \times (n+m_u+m_\ell)$-dimensional matrices

$$\tilde{A}_i = \begin{pmatrix} A_i & & \\ & \operatorname{diag}\big((e_i)_{J_u}\big) & \\ & & -\operatorname{diag}\big((e_i)_{J_\ell}\big) \end{pmatrix}, \quad \tilde{C} = \begin{pmatrix} C & & \\ & \operatorname{diag}(u_{J_u}) & \\ & & -\operatorname{diag}(\ell_{J_\ell}) \end{pmatrix}.$$

Extending the results on dual fixing in [53] to the SDP case, we can fix binary variables y_i, depending on the values of V or W corresponding to the variable bounds in the dual problem, and also enhance bounds for non-binary variables, which gives us the following theorem:

Theorem 3. *Let (X, V, W, U_1, U_2, U_3) be primal feasible for (B-P) with corresponding primal objective value $f = C \bullet X + \sum_{i \in J_u} u_i V_{ii} - \sum_{i \in J_\ell} \ell_i W_{ii}$ and let L be a lower bound on the optimal objective value of the mixed-integer semidefinite program (D-MISDP). Then for every optimal solution y of (D-MISDP) we have that*

$$(1) \qquad y_j \le \ell_j + \frac{f-L}{W_{jj}} \quad \forall\, j \in J_\ell \qquad \text{and} \qquad y_j \ge u_j - \frac{f-L}{V_{jj}} \quad \forall\, j \in J_u.$$

If y_j is an integer variable and $W_{jj} > f - L$, then for every optimal solution $y_j = \ell_j$; on the other hand, if $V_{jj} > f - L$, then every optimal solution satisfies $y_j = u_j$.

Proof. Let y be a dual feasible solution and $\tilde{Z} := \tilde{C} - \sum_{i=1}^m \tilde{A}_i y_i \succeq 0$. Furthermore, let

$$Y := \begin{pmatrix} X & U_1 & U_2 \\ U_1 & V & U_3 \\ U_2 & U_3 & W \end{pmatrix}.$$

By primal and dual feasibility we get that

$$\tilde{Z} \bullet Y = \Big(\tilde{C} - \sum_{i=1}^m \tilde{A}_i y_i\Big) \bullet Y = \tilde{C} \bullet Y - \sum_{i=1}^m y_i (\tilde{A}_i \bullet Y) = \tilde{C} \bullet Y - \sum_{i=1}^m y_i b_i = \tilde{C} \bullet Y - b^\top y$$

for $\tilde{A}_i$ and $\tilde{C}$. Thus,

$$\begin{aligned} b^\top y &= \tilde{C} \bullet Y - \tilde{Z} \bullet Y = f - \tilde{Z} \bullet Y \\ &= f - Z \bullet X - \sum_{i \in J_u} V_{ii}(u_i - y_i) - \sum_{i \in J_\ell} W_{ii}(y_i - \ell_i) \\ &\le f - \sum_{i \in J_u} V_{ii}(u_i - y_i) - \sum_{i \in J_\ell} W_{ii}(y_i - \ell_i), \end{aligned}$$

where we used that $Z \bullet X \geq 0$, since both matrices are positive semidefinite if they are feasible.

Now assume that

$$y_j > \ell_j + \frac{f - L}{W_{jj}}. \tag{2}$$

Using the fact that diagonal entries of positive semidefinite matrices are nonnegative and y is dual feasible, we get

$$b^\top y \leq f - \sum_{i \in J_u} V_{ii}(u_i - y_i) - \sum_{i \in J_\ell} W_{ii}(y_i - \ell_i) \leq f - W_{jj}(y_j - \ell_j) < L.$$

Thus, any solution satisfying (2) cannot be optimal and the first part of (1) follows. For the lower bound assume

$$y_j < u_j - \frac{f - L}{V_{jj}}. \tag{3}$$

Then we get

$$b^\top y \leq f - \sum_{i \in J_u} V_{ii}(u_i - y_i) - \sum_{i \in J_\ell} W_{ii}(y_i - \ell_i) \leq f - V_{jj}(u_j - y_j) < L.$$

Thus, any solution satisfying (3) cannot be optimal the second part of (1) follows.

The results for integer variables follow from the fact that $W_{jj} > f - L$ implies

$$y_j \leq \ell_j + \frac{f - L}{W_{jj}} < \ell_j + 1,$$

and similarly $y_j > u_j - 1$ for the upper bound. □

8. Solver Components

The Branch-and-cut framework SCIP and our extension SCIP-SDP allow to easily add further components to the base solvers. In this section, we highlight some of the components included in SCIP-SDP, which are all implemented as plugins. It is therefore easily possible to extend the code-basis by further user-written component plugins or to integrate some of the plugins in different projects.

8.1. Presolving. In the context of mixed-integer linear programs, a collection of presolving techniques for linear constraints are known, see for instance Savelsbergh [65]. For MISDPs, one can detect 1×1 SDP-blocks and transform them to linear constraints. Moreover, since the diagonal entries of a positive semidefinite matrix are nonnegative, one can add the following linear constraints:

$$C_{jj} - \sum_{i=1}^{m} (A_i)_{jj} y_i \geq 0 \qquad \forall\, j \in [n]\,.$$

These constraints do not strengthen the relaxation, but can be used to apply presolving techniques for linear constraints, e.g., bound tightening, see Mars [51].

8.2. Branching Rules and Node-Selection. The development of branching rules for general MISDPs is less advanced compared to the mixed-integer linear case. In the literature, branching rules for MISDPs are often problem-specific. However, some obvious possibilities are the following:

- *Integer Infeasibility*: Branch on the variable with fractionality closest to 0.5. Ties can be broken according to the absolute value of the objective function coefficients.
- *Objective:* Choose a fractional variable with largest absolute value of its objective coefficient. Ties can then be broken according to integer infeasibility.
- *Objective & Infeasibility:* A combination of the last two branching rules is possible by multiplying the objective coefficient with the integer infeasibility.

A further general rule is *inference branching*, see Achterberg [2]. Here historical information about fixings resulting from previous variable branchings is used to choose new branching variables. The larger the number of implied fixings, the higher the impact of the variable and the higher its chance of being selected.

A drawback of inference branching is that at the beginning of the solution process, there is little historical information. In principle, this can be alleviated by using strong branching techniques, in which tentative variable branching steps are performed and the resulting subproblems are (partially) solved to estimate the change in the objective. However, such approaches are usually very time consuming. Even if improvements like reliability branching (Achterberg et al. [3]) are applied, this will likely take too much time to be applied for SDPs.

In contrast, the rules for choosing the next node to be handled are universal for any branch-and-bound solver. A typical default node-selection rule is best-first search, in which the node with weakest relaxation value is chosen first. Of course, breadth- and depth-first search or combinations of these rules are also possible.

8.3. Primal Heuristics. Primal heuristics can help to speed up the solution process, since they possibly provide better lower bounds, allowing to cut off nodes of the tree in which no better solution can be found, see Step 6 of Algorithm 1. In the mixed-integer programming (MIP) context, many primal heuristics have been developed, see, e.g., Fischetti and Lodi [30] for a brief survey. Extensions of these as well as new methods for general mixed-integer nonlinear problems have also been developed, see, e.g., Berthold [15] for an overview. There are several obvious extensions/specializations for MISDPs:

- *Rounding:* Taking a solution of a subproblem relaxation, fractional values of integer variables can be rounded to the next integer value. The resulting solution then has to be tested for feasibility. However, this test is relatively time consuming and arbitrarily rounding a solution will only seldomly be successful. An improvement is to determine beforehand whether a fractional variable y_j in (D-MISDP) can be rounded up or down without violating feasibility, see, e.g., [51]. If A_j is positive semidefinite, then y_j can safely be rounded down, since we then add the positive semidefinite matrix $(y_j - \lfloor y_j \rfloor)A_j$ to the already positive semidefinite $C - \sum_{i=1}^{m} A_i y_i$. Similarly, if A_j is negative semidefinite, y_j can be rounded up. The rounding heuristic then only uses these valid directions. If both directions are invalid, the heuristic terminates.

- *Randomized Rounding:* The seminal paper of Goemans and Williamson [36] already contains the idea to obtain a feasible solution for the max cut problem by rounding the solution of an SDP-relaxation using a random hyperplane. Pilanci et al. [59] apply randomized rounding for each binary variable, i.e., a binary variable is rounded to 0 uniformly at random with probability equal to its value in the relaxation. In several applications, the obtained solutions are always or often feasible and repeating the process yields high quality solutions.
- *Diving:* Diving sequentially rounds fractional variables and resolves the SDP until an integral solution is found. The choice of the next variable can be performed similar to the branching rules described in Section 8.2, e.g., by using integer infeasibility.
- *Large Neighborhood Search:* There are several primal heuristics that exploit the possibility to recursively use the branch-and-bound algorithm on problems of smaller size: *local branching* [29], *RINS* [22], *RENS* [16], *crossover* [64], and *DINS* [35]. This idea naturally extends to MISDPs.

As should be apparent from this brief description, primal heuristics for MISDPs are likely to be time consuming. Thus, one has to carefully decide how often heuristics are run, i.e., in which nodes of the tree they are applied.

8.4. Cutting Planes. Cutting planes form one of the most important components of a MIP-solver. In the MISDP setting, research in this direction is only at its beginnings. An example for a problem-specific approach is the work by Helmberg and Rendl [40]. Moreover, cutting constraints for general conic problems have been investigated, see, for example, Çezik and Iyengar [20], Modaresi [54], Drewes and Pokutta [24], and Atamtürk and Narayanan [8]. But there is still a long way to go until cutting planes for mixed-integer semidefinite programming reach the same level as in its linear counterpart. Our implementation currently does not include the generation of general SDP cutting planes.

9. Implementation

We have implemented the branch-and-bound approach for solving MISDPs described above in C/C++ using SCIP [33, 67]. The implementation is freely available as SCIP-SDP [68]. The code is based on the implementation of Mars and Schewe [51, 52], which we partly reworked and extended. In this section, we describe several choices made in this implementation.

9.1. Blockstructure. The implementation allows for multiple SDP-constraints or SDP-blocks in (D-MISDP) instead of one. The theoretical results still hold, because if C and all A_i have a common blockstructure, then the positive semidefiniteness of $C - \sum_{i=1}^{m} A_i y_i$ is equivalent to the sum being positive semidefinite for each block. In practice, this has the advantage of smaller matrices being checked for positive semidefiniteness and appearing in Cholesky factorizations. Multiple SDP-blocks are also supported by the interfaced SDP-solvers.

9.2. Feasibility Check. The feasibility of a given solution is checked by computing the smallest eigenvalue of $C - \sum_{i=1}^{m} A_i y_i$ for each block using LAPACK [5]. If this eigenvalue is bigger than the negative feasibility tolerance, then the solution is

accepted for the semidefinite constraint. The default value for the feasibility tolerance in SCIP-SDP is 10^{-6}, while the default optimality tolerance for solving the SDPs is 10^{-4}.

9.3. Interface to SDP-Solvers. The implementation contains a generic interface and solver-specific interfaces. The generic interface takes care of removing locally fixed variables, since they cannot be handled by some SDP-solvers, as well as zero rows and columns in the dual semidefinite matrix $C - \sum_{i=1}^{m} A_i y_i$. These might occur by fixing all variables y_i to zero whose matrices A_i have entries in these rows and columns. Moreover, if all variables are fixed, then we check the resulting solution for feasibility by computing the minimal eigenvalue instead of passing it to the SDP-solver.

We use the penalty formulations described in Section 6 in order to increase the stability of the solution process as follows: After failing to solve an SDP-relaxation (D) to optimality, we first check the problem for feasibility using (FC). If (FC) could not prove infeasibility, we then try to solve the problem using (Γ-D) with a penalty parameter Γ between 10^5 and 10^{12}. The value is a multiple of the biggest objective coefficient of the problem, where the multiple depends on the chosen SDP-Solver. If the SDP-Solver did not converge, we increase Γ at most two times by a factor of 10^3, and we do the same if the penalty variable r in the resulting solution is bigger than the feasibility tolerance and $\mathrm{Tr}(X) = \Gamma$. This implies that Γ was too small and the optimal solutions of the original problem were cut off. On the other hand, if $\mathrm{Tr}(X) < \Gamma$, then we decrease the optimality tolerance of the SDP-solver with the aim to force r further towards zero, since we know that (Γ-D) is exact. After at most two changes to Γ and ε, we stop the solving process and continue with either the best dual bound computed via the penalty formulation or the dual bound of the parent node, whichever is better, and then branch on any variable that is not yet fixed.

There are currently two interfaces to SDP-solvers. The first interface is for DSDP [14], an implementation of the interior-point dual-scaling algorithm written in C and developed by Steve Benson, Yinyu Ye, and Xiong Zhang. In this interface, we explicitly use the penalty formulation provided by DSDP. By default, DSDP will start the solving process using the penalty formulation. However, once r is small enough, it will be fixed to 0 and removed from the formulation. DSDP can also be set to treat the penalty variable like a usual variable that has to be kept strictly positive during the interior-point iterations. We use this option after encountering numerical problems.

The other interfaced SDP-solver is SDPA [74, 75], which is an implementation of the primal-dual interior-point method in C++ developed by Katsuki Fujisawa, Makoto Yamashita and others. In this case, we implemented the penalty-formulation inside the interface. Since SDPA offers multiple sets of default parameter choices, we first try to solve the SDP with the fastest settings. If this does not work because of numerical difficulties, we switch to the two slower but more stable settings. Only if all three settings do not lead to stable results, we check for feasibility and resort to the penalty formulation. For efficiency, successful settings will be remembered, so that a child node will immediately start with the same settings that proved to be successful for the parent node (but we will never start with the penalty formulation or the feasibility check without first trying the original formulation with the most stable settings). In SDPA, the initial point X_0 is chosen as

$\lambda^* \cdot I_n$, with default value $\lambda^* = 100$. We try to compute a better guess for the initial point by first computing

$$\lambda = \max\left\{ S \cdot \max_{i\in[m]}\{|u_i|,|\ell_i|\} \cdot \max_{i\in[m]}\|A_i\|_\infty + \|C\|_\infty,\ \frac{\max_{i\in[m]}|b_i|}{S \cdot \min_{i,j,k:\,|(A_i)_{kj}|\neq 0}|(A_i)_{kj}|} \right\},$$

where S is a sparsity parameter computed by dividing the total number of upper-triangular nonzero entries of all A_i by the total number of upper triangular entries of a single matrix. The value of λ^* is then set to 1.5 if $\lambda < 10$ and to 10^5 otherwise, since this produced the best results in our tests.

9.4. Checking the Slater Condition. In order to evaluate constraint qualifications in practice, we also implemented the possibility to check every occurring SDP-relaxation for the primal and dual Slater condition.

To check the dual Slater condition, we solve the auxiliary problem (FC) that we also use to detect infeasibility. If the optimal objective value of (FC) is strictly negative, a dual solution y and $r < 0$ exist such that $C - \sum_{i=1}^m A_i y_i \succeq -I_n r \succ 0$, so the dual Slater condition is fulfilled. On the other hand, if the optimal objective is nonnegative, no such solution exists and therefore the dual Slater condition cannot be fulfilled. If it is strictly positive, the problem is infeasible. In our implementation, we do not check the Slater condition for the variable bounds.

For checking the primal Slater condition we use the formulation

$$\text{(P-Slater)} \qquad \begin{aligned} \sup\ & r \\ \text{s.t.}\ & A_i \bullet (X + I_n r) = b_i, \quad \forall i \in [m], \\ & X \succeq 0, \quad r \geq 0, \end{aligned}$$

where we now include all variable bounds in the A_i matrices. If the optimal objective value of (P-Slater) is strictly positive, there exist $X \succeq 0$ and $r > 0$ such that $X + I_n r$ fulfills the equality constraints in (P). Therefore, $X + I_n r \succeq I_n r \succ 0$ is a strictly feasible solution for (P), and the primal Slater condition is fulfilled. Conversely, if the optimal objective value is zero or the problem is infeasible, the primal Slater condition cannot be fulfilled, since for any strictly feasible solution X of (P), there exists $r > 0$ such that $(X - I_n r,\ r)$ is feasible for (P-Slater) with positive objective value.

Note that if every variable in (D) has finite bounds, (P-Slater) will be unbounded and the primal Slater condition is satisfied. This can be shown by observing that every bound on a dual variable leads to a nonnegative primal variable only appearing in the corresponding primal constraint. If every dual variable is bounded from above and below, these variables allow to generate a feasible solution to (P-Slater) for any value of r. Therefore, we only have to solve (P-Slater) if at least one dual variable is unbounded. Note that we cannot be sure whether the auxiliary problem itself satisfies the Slater condition, so we might not always be able to check the Slater condition.

9.5. Further Components. The presolving of linear constraints (including the objective bound) is implemented in SCIP and is described in Achterberg [1]. We implemented the branching rules described in Section 8.2, which was straightforward. Moreover, we implemented dual fixing as described in Section 7. The diving heuristic mentioned in Section 8.3 was implemented by adapting existing versions for the MIP case. The rounding heuristic is automatically performed by

SCIP based on "rounding locks" (see [1]), which we set up using the SDP constraints. The large neighborhood search heuristics are implemented in SCIP, see Berthold et al. [17] for a description on the performance for MINLP problems. In our implementation, we currently do not use these heuristics, because they would require more tuning effort in order to balance time against solution quality.

10. Applications

In the following, we want to demonstrate the usability of the MISDP-framework with three applications. In the first two applications, truss topology design and cardinality constrained least squares, the semidefinite constraints stem from quadratic or bilinear constraints or objectives, while the minimum k-partitioning problem can also be formulated as a MIP, but introducing a semidefinite constraint allows to reduce the size of the problem and increase the strength of the relaxation.

10.1. Truss Topology Design. Truss topology design determines truss structures that are both stable and lightweight. Historically, it was first formulated as a nonlinear program, see, e.g., Bendsøe and Sigmund [12]. Ben-Tal and Nemirovskii [11] introduced a robust SDP-formulation, which was extended by Mars [51] using binary variables for choosing components; we will follow this latter approach.

The general idea is to start with a ground structure, i.e., a simple directed graph $D = (V,E)$ with n nodes $V = \{v_1,\dots,v_n\} \subseteq \mathbb{R}^d$. A subset $V_f \subset V$ of n_f nodes are free, while the remaining nodes are fixed. The goal is to optimize the cross-sectional areas $x \in \mathbb{R}^E_+$ of the possible bars in E such that the total volume of the truss is minimized, while still keeping it stable when a given force $f \in \mathbb{R}^{d_f}$, $d_f = d \cdot n_f$, is applied. The vector f consists of the d-dimensional forces that are applied for each of the n_f free nodes and causes displacements $u \in \mathbb{R}^{d_f}$. The node displacements can be computed from the equilibrium constraint $A(x)u = f$ utilizing the *stiffness matrix* $A(x) = \sum_{e\in E} A_e x_e$ with $A_e = b_e b_e^\top$, $b_e = (b_e(v))_{v\in V_f} \in \mathbb{R}^{d_f}$ and

$$b_e(v) = \begin{cases} \sqrt{\kappa}\,\frac{v_i - v_j}{\|v_i - v_j\|_2^{3/2}}, & \text{if } v = v_i, \\ \sqrt{\kappa}\,\frac{v_j - v_i}{\|v_i - v_j\|_2^{3/2}}, & \text{if } v = v_j, \\ 0 & \text{otherwise,} \end{cases} \qquad \text{for } e = (v_i, v_j),\ v \in V_f,$$

where κ is Young's modulus of the used material. The stability of the structure is measured by the *compliance* $\frac{1}{2} f^\top u$, which describes the potential energy stored in the deformed truss and has to be less than a given constant $C_{\max}$.

Using a robust optimization approach, the trusses can also be protected against perturbations of the load by computing a worst-case over all forces within a given uncertainty set, see, e.g., Ben-Tal et al. [10]. We use an ellipsoidal uncertainty set $\{f \in \mathbb{R}^{d_f} : f = Qg,\ \|g\|_2 \le 1\}$ for a given matrix $Q \in \mathbb{R}^{d_f \times k}$. This approach also allows to include given load scenarios. Alternatively, we apply multiple external forces f, which leads to one additional SDP-constraint per applied force.

Moreover, bars may only be available with certain cross-sectional areas from a given set $\mathscr{A} \subset \mathbb{R}_+$. We introduce binary variables x^a_e which model whether bar e

has cross-sectional area a. This finally leads to the volume minimizing MISDP

$$\text{(TT)}\qquad \begin{aligned} \inf\quad & \sum_{e\in E} \ell_e \sum_{a\in\mathscr{A}} a\, x_e^a \\ \text{s.t.}\quad & \begin{pmatrix} 2\tau I_k & Q^\top \\ Q & A(x) \end{pmatrix} \succeq 0, \\ & \sum_{a\in\mathscr{A}} x_e^a \le 1 && \forall e\in E, \\ & \tau \le C_{\max}, \\ & x_e^a \in \{0,1\} && \forall e\in E,\ a\in\mathscr{A}, \end{aligned}$$

where $A(x) = \sum_{e\in E}\sum_{a\in\mathscr{A}} A_e\, a\, x_e^a$, and ℓ_e is the length of bar $e\in E$.

One can show that the root node relaxation of Problem (TT) always satisfies the primal Slater condition. The dual Slater condition depends on the choice of $C_{\max}$ and the ground structure, but will be satisfied for all "reasonable" choices.

10.2. Cardinality Constrained Least Squares. Another application of MISDPs arises from the problem of cardinality constrained least squares, see Hocking [42]. We use the formulation as an MISDP of Pilanci et al. [59].

Given sample points $a^1,\dots,a^m\in\mathbb{R}^d$, collected as the rows of matrix $A\in\mathbb{R}^{m\times d}$, and corresponding measurements $b_1,\dots,b_m$, classical linear regression determines a vector $x\in\mathbb{R}^d$ such that $\|Ax-b\|_2$ is minimized. Its solution serves as a predictor for future measurements. In many applications, its quality can be improved by only allowing k of the components of x to be nonzero, i.e., the most important k of the d features are selected. This leads to the cardinality constrained quadratic program

$$\inf_{x\in\mathbb{R}^d}\left\{\frac{1}{2}\|Ax-b\|_2^2+\frac{1}{2}\rho\|x\|_2^2 \,:\, \|x\|_0\le k\right\},$$

where $\|x\|_0$ is the size of the support of x and $\frac{1}{2}\rho\|x\|_2^2$ is a regularization term for given positive $\rho\in\mathbb{R}$. As shown in [59], by introducing binary variables z for deciding on the support, this can be equivalently transformed to the MISDP

$$\text{(CLS)}\qquad \begin{aligned} \inf\quad & \tau \\ \text{s.t.}\quad & \begin{pmatrix} I_m+\frac{1}{\rho}A\,\mathrm{Diag}(z)\,A^\top & b \\ b^\top & \tau \end{pmatrix} \succeq 0, \\ & \sum_{j=1}^d z_j \le k,\ z\in\{0,1\}^d. \end{aligned}$$

The relaxation of the MISDP (CLS) satisfies the dual Slater condition, since $\frac{1}{\rho}A\,\mathrm{Diag}(z)\,A^\top$ is positive semidefinite for all $z\in[0,1]^d$. Thus, $I_m+\frac{1}{\rho}A\,\mathrm{Diag}(z)\,A^\top$ is positive definite. Therefore, choosing τ large enough, renders the whole matrix positive definite. The primal Slater condition also holds, since all variables are bounded, except for τ with a corresponding primal constraint $-X_{m+1,m+1}=-1$, so that $X=I_{m+1}$ is a strictly feasible primal solution for a suitable choice of the primal variables corresponding to the variable bounds.

10.3. Minimum k-Partitioning. In this section, we introduce the problem of finding a minimum k-partitioning of a weighted graph with bounded sizes of the parts. We first present the version without size constraints. Given an undirected graph $G=(V,E)$ with n nodes, edge-weights $c:E\mapsto\mathbb{R}$ and a positive integer $k\ge 2$, we

want to find a partitioning of V into k disjoint sets $V_1, \dots, V_k$ that minimizes the total weight of all edges within the parts

$$\sum_{i=1}^{k} \sum_{e \in E[V_i]} c(e).$$

This problem has applications in frequency planning, see, e.g., Eisenblätter [26], and in the layout of electronic circuits, see, e.g., Lengauer [47].

The minimum k-partitioning problem is closely connected to the minimum graph bisection and max-cut problems, which correspond to $k = 2$. It is NP-hard and also difficult to approximate, as shown by Kann et al. [43]. Many different semidefinite relaxations for this problem and the equi-partitioning variant, where all parts are required to be of the same size, have been proposed, for an overview see, e.g., Sotirov [70]. Here we want to use the formulation of Eisenblätter [25, 26], since it allows an exact formulation as an MISDP.

Assume w.l.o.g. that $V = \{1, \dots, n\}$ and define the cost matrix C with entries $c_{ij} := c(\{i, j\})$ for $\{i, j\} \in E$ and $c_{ij} = 0$ otherwise. Introducing variables $X_{ij} \in \{\frac{-1}{k-1}, 1\}$, where $X_{ij} = 1$ if and only if node i and j are in the same part, the problem is equivalent to

$$\begin{aligned} \inf \quad & \sum_{1 \le i < j \le n} c_{ij} \frac{(k-1)X_{ij} + 1}{k} & \\ \text{s.t.} \quad & X \succeq 0, & \\ & X_{ii} = 1 & \forall i = 1, \dots, n, \\ & X_{ij} \in \left\{\tfrac{-1}{k-1}, 1\right\} & \forall 1 \le i < j \le n. \end{aligned}$$

In order to obtain an MISDP, we introduce binary variables Y_{ij} such that $X_{ij} = \frac{-1}{k-1} + \frac{k}{k-1} Y_{ij}$. Therefore, $X_{ij} = \frac{-1}{k-1}$ if $Y_{ij} = 0$ and $X_{ij} = 1$ if $Y_{ij} = 1$. This leads to

$$\text{(Min-}k\text{)} \qquad \begin{aligned} \inf \quad & \sum_{1 \le i < j \le n} c_{ij} Y_{ij} \\ \text{s.t.} \quad & \tfrac{-1}{k-1} J + \tfrac{k}{k-1} Y \succeq 0, \\ & Y_{ii} = 1,\ Y \in S_n \cap \{0,1\}^{n \times n}, \end{aligned}$$

where J is the all-one matrix.

In the next step, we consider a weight function w for the nodes and add the constraint that the sum of weights for the nodes in every part lies in the interval $[\ell, u]$. Since the i-th row of Y has a 1 entry for each element in the same part as i (including the diagonal entry for i itself), we can enforce this through the linear constraint $\ell \le \sum_{j=1}^{n} w_j Y_{ij} \le u$. When using only upper triangular entries without the diagonal as variables, we obtain

$$\ell - w_i \le \sum_{j<i} w_j Y_{ji} + \sum_{j>i} w_j Y_{ij} \le u - w_i \qquad \forall i = 1, \dots, n. \tag{4}$$

The relaxation of (Min-k) satisfies the dual Slater condition for $k > 2$ with strictly feasible solution $Y = I_n$. For $k = 2$, it is not fulfilled, since all entries of the matrix have absolute value 1 and the 2×2 principal minors have value $1 - (\pm 1)^2 = 0$. Thus, no feasible matrix can be strictly positive definite. For larger k, the dual Slater condition is also lost after fixing a single variable to 1.

TABLE 1. Overview over all instances

problem-type	#	cont-vars	bin-vars	blocks	blocksize	LP-conss
truss topology	60	1	27 – 384	1 – 4	10 – 44	85 – 801
colon cancer	20	1	100	1	63	201
random CLS	45	1	32 – 128	1	43 – 367	65 – 256
VLSI	10	0	105 – 1128	1	15 – 48	240 – 2352
random min-k-part	59	0	120 – 2415	1	16 – 70	272 – 4970
total	194	0 – 1	27 – 2415	1 – 4	10 – 367	65 – 4970

If we fix X_{ij} to 1, we have $X_{ii} = X_{jj} = X_{ij} = 1$, so the principal minor of the corresponding 2×2 matrix is 0. Therefore, X can no longer be strictly positive definite and the relative interior of the dual problem becomes empty. Moreover, the dual Slater condition will also get lost when adding (4) for $\ell = u$. The primal Slater condition holds, however, since all variables are bounded.

Note that several MISDP approaches for minimum partitioning problems have appeared in the literature. For instance, Ghaddar et al. [34] and Anjos et al. [6] investigate the problem without restrictions on the size of the parts. Rendl et al. [62] deal with max cut problems, i.e., $k = 2$. Moreover, Armbruster et al. [7] investigate SDP- and LP-approaches for the minimum bisection problem, which also corresponds to $k = 2$. They bound the weights of the subsets, but minimize the cost of edges bridging the cut. This is equivalent to minimizing the cost of edges within the cuts by switching the sign of the weights (and changing the objective value by a constant). The formulation in Ferreira et al. [27] is closest to ours, but again minimizing the total weight of edges between the parts. They apply an LP-based branch-and-cut algorithm, including a separation routine for constraints involving edge-variables and additional cutting planes.

In our experiments, we use an MISDP approach and $k \geq 2$. The goal is to illustrate that one can obtain solutions for such problems using a general purpose code. Moreover, it is easy to add constraints like individual bounds for the weights of parts. Of course, our code cannot be competitive with the above mentioned problem-specific approaches.

11. NUMERICAL RESULTS

In this section, we will investigate the influence of different solver components like dual fixing, heuristics, and branching rules on the solution process. Furthermore, we will also consider the behavior of SDP-relaxations not fulfilling the dual Slater condition during the solution process of mixed-integer semidefinite programs.

11.1. Testset. For testing our implementation, we created a testset of 194 instances arising from the three applications described in the preceding section. An overview over all instances can be found in Table 1. The first two columns list the names of the problem set and the number of instances. Moreover, it presents the range of the following numbers: the number of continuous and binary variables, the number of SDP blocks, the size of each SDP block, and the number of linear constraints.

11.1.1. *Truss Topology Design.* For the truss topology design problem we created a total of 60 instances. The ground structures consist of 6 to 27 nodes and 12 to 196 bars. For each bar we have a choice between 1 (i.e., a yes/no decision whether

to use the bar) and 16 different sizes. We applied 1 to 4 different primary forces, and for 52 of the 60 instances we further used an ellipsoidal uncertainty set.

11.1.2. *Cardinality Constrained Least Squares.* For testing the performance of the solver on the cardinality constrained least squares problem, we created a testset of 65 instances of varying difficulty. The testset consists of 20 instances based on real-world data and 45 randomly generated instances.

The real-world instances are based on the colon cancer data set in [4], consisting of 62 samples and 2000 features. These data were also used in [59]. Since solving MISDPs exactly uses considerably more time than solving only a single SDP, we had to use instances of smaller size in comparison to [59]. After normalizing the entries, we split the data into 20 instances using 100 features each, resulting in data $A \in \mathbb{R}^{62\times 100}$ and $b \in \{\pm 1\}^{62}$, distinguishing between tumor tissues (-1) and normal tissues $(+1)$, like in [59]. The sparsity level k was chosen between 5 and 24.

For creating the randomly generated instances, we used the same technique as in [59], which was originally proposed by Wainwright [73]. The design matrix $A \in \mathbb{R}^{m\times d}$ was randomly generated with independent and identically distributed Gaussian $N(0,1)$ entries. For creating the measurement data $b \in \mathbb{R}^m$, we generated k-sparse regression vectors $x^\star$ with uniformly distributed support and entries $\pm 1/\sqrt{k}$, where the sign was again chosen uniformly. Then we set $b = Ax^\star + \varepsilon$, where $\varepsilon \in \mathbb{R}^m$ with $N(0, 0.25)$ entries as in [73].

We generated three instances each for 15 different choices of the triple (m,d,k). As in [59], we chose $k = \lceil \sqrt{d} \rceil$ and $m = \lceil \alpha k \ln d \rceil$ with $\alpha \in \{2,4,6,8\}$. Again, we had to use smaller instances compared to [59], i.e., we use $d \in \{32,64,96,128\}$ instead of $d \in \{64,128,256\}$, also omitting the combination of $d = 128$ and $\alpha = 8$.

The resulting instances are completely dense, i.e., all matrices A_i are dense, unless some observations are zero.

11.1.3. *Minimum k-Partitioning.* The instances for the minimum k-partitioning problem were generated based on the graphs from Ghaddar et al. [34], which were in turn generated using the rudy graph-generator [63]. We only used the medium-sized instances, since our general purpose code is not as fast as the specialized branch-and-cut algorithm from [34]. We added size constraints with $\ell = \lfloor n/k \rfloor$, $u = \lceil n/k \rceil$ as explained in Section 10.3. Furthermore, while the instances were only solved for $k = 2$ and $k = 3$ in [34], we used larger values of k for some of them to create a more heterogeneous testset.

We generated a total of 69 instances. These consist of 42 instances based on randomized spin glass graphs for two- and three-dimensional grids. They have 16 to 49 nodes, 32 to 98 edges with Gaussian distributed or ± 1 weights, and k between 2 and $n/2$; these instances correspond to a physics application of the max-cut problem described in more detail in Liers et al. [48]. Further 17 instances are based on clique graphs of size 20 to 70, with edge-weight $|i-j|$ for the edge between the i-th and j-th node and k again chosen between 2 and $n/2$.

In addition to the randomly generated instances, we also use ten of the smaller instances, which first appeared in Ferreira et al. [27] and are now part of a benchmarking set [28]. These real-world instances treat the clustering of cells on a chip as a preprocessing step for the layout-problem in very-large-scale integration (VLSI). We select all VLSI-instances solved in [27] within a time limit of 300 minutes. Since we are minimizing the cost of edges within the parts instead of the

TABLE 2. Data columns for presenting the solving times

column name	description
`solved`	number of instances solved to optimality within the time limit of 3600 seconds
`aborts`	number of instances aborted by the solver
`nodes`	shifted geometric mean of the number of nodes in the tree, where we only use those instances that were solved by all settings (shift $s = 100$)
`time`	shifted geometric mean of the solving time in seconds (shift $s = 10$, unsolved instances use 3600 seconds)
`iters`	shifted geometric mean of the number of SDP-iterations for all instances solved by all settings (shift $s = 1000$)
`penalty`	arithmetic mean of the percentage of nodes that were solved to optimality or infeasibility using the penalty formulations introduced in Section 6
`unsucc`	arithmetic mean of the percentage of nodes that could not be solved successfully by any method
`fixings`	arithmetic mean of the number of fixings performed by the dual fixing technique of Section 7

cut, we switched the sign of all edge-weights. The instances consist of 15 to 48 nodes of different weight, four clusters with identical maximum capacities, which we use as upper bounds on the size of the parts, and 29 to 132 edges.

Note that when written in the dual form (D-MISDP), the minimum k-partitioning problem yields very sparse SDP-blocks in which every matrix A_i has only a single nonzero entry.

11.2. Performance of Different Components. In this section, we report on the performance achieved with our implementation on the described testset using the different components and user-specifiable settings for these components, e.g., the frequency to apply heuristics. For a list of all possible settings and how to change them, we refer to the online documentation of SCIP-SDP. The computations were performed on a Linux cluster with Intel i3-550 CPUs with 3.2GHz, 4MB cache and 8GB memory running Linux. The code was compiled with gcc 4.4.5 with `-O3` optimization. Each computation was performed single-threaded with a single process running on each computer and with a time limit of 3600 seconds. The experiments were carried out using preliminary versions of SCIP-SDP 2.1.0 and SCIP 3.2.1 with either DSDP 5.8 or SDPA 7.3.8. Note that for SDPA it is also possible to run the linear algebra subroutines multi-threaded, which, however, does not seem to be worthwhile for instances of the size used in our testset. Detailed results for all computations can be found in the online supplement [32].

For reporting some of the aggregated results, we will use the *shifted geometric mean* to decrease the influence of easy instances, see Achterberg [2] for more details. The shifted geometric mean of values $x_1, \ldots, x_n$ is computed as

$$\left(\prod_{i=1}^{n}(x_i + s)\right)^{1/n} - s$$

for a given shift s. The tables below contain data columns as described in Table 2. We also present performance plots, in which the number of solved instances is plotted against the solving time factor relative to the fastest solver for each instance, see Dolan and Moré [23].

Note that in some cases our solver failed to finish and aborted prematurely. The main reason is that the root node relaxation could not be solved, in which case we have to terminate the solution process. SDPA can also abort the solving process in case of errors in the linear algebra subroutines, which sometimes happens for infeasible subproblems. Many of these problems do not appear for these instances

TABLE 3. Results for DSDP/SDPA and different branching rules for the *truss topology* testset of 60 instances

settings	solved	aborts	nodes	time	iters	penalty	unsucc
`DSDP-infer`	43	0	4871.1	716.7	145,272.4	0.16 %	0.08 %
`DSDP-infobj`	48	0	1871.5	455.5	62,209.3	0.11 %	0.22 %
`DSDP-obj`	41	0	5495.3	815.3	162,678.1	0.76 %	1.81 %
`DSDP-inf`	42	0	2853.9	620.4	94,125.0	0.69 %	1.50 %
`SDPA-infer`	40	7	3739.9	480.7	78,858.1	4.29 %	4.75 %
`SDPA-infobj`	43	9	1926.4	319.9	42,317.8	2.74 %	1.66 %
`SDPA-obj`	45	2	4220.0	444.8	88,068.3	4.52 %	5.12 %
`SDPA-inf`	36	9	3229.1	501.8	70,068.3	2.49 %	1.96 %

when using different parameters for the SDP-solvers, e.g., λ^* in SDPA. But we were unable to generate a general rule that works for all instances. Furthermore, there are some cases where the SDP-solvers exceeded the time or memory limit, which caused the cluster to abort these processes.

11.2.1. *Influence of Solvers and Branching Rules.* We begin our presentation of the results by an investigation of the influence of branching rules and the two different SDP-solvers. We use eight different settings. In each setting, the fractional diving heuristic is only called in the root node and randomized rounding and dual fixing are disabled. Either DSDP or SDPA is used to solve the SDP-relaxations together with one of the following four branching rules, for details see Section 8.2:

- `infer`: inference branching: branch on a variable that historically led to the highest number of implied fixings;
- `infobj`: objective & infeasibility branching: choose a variable with largest product of integer infeasibility and absolute value of objective coefficient;
- `obj`: objective branching: branch on a variable with largest absolute value of its objective coefficient;
- `inf`: infeasibility branching: choose a variable with largest integer infeasibility.

Table 3 gives an overview of the results for these eight settings for the truss topology instances. For DSDP, the `infobj` branching rule clearly outperforms the rest, while for SDPA it is still the fastest, but cannot solve as many instances as `obj` branching, which is mainly caused by the higher number of instances aborted by SDPA.

When comparing the solvers, SDPA is 20 to 50 % faster than DSDP, but solves less instances, again because of the relatively high number of aborted instances. The number of branch-and-bound nodes is similar for both solvers, while the number of SDP-iterations is significantly higher for DSDP, which is due to the different choice of algorithms applied by DSDP and SDPA. In contrast to the primal-dual algorithm used in SDPA, the dual-scaling method in DSDP does not possess superlinear convergence, but the computational cost of a single iteration is significantly smaller, see, e.g., [14], therefore a higher number of SDP-iterations per problem is to be expected. When comparing their numerical stability, SDPA runs into numerical troubles for up to 10 % of the SDP-relaxations, while DSDP can solve at least 97 % of the relaxations using the default formulation.

TABLE 4. Results for DSDP/SDPA and different branching rules for the *cardinality constrained least squares* testset of 65 instances

settings	solved	aborts	nodes	time	iters	penalty	unsucc
`DSDP-infer`	23	0	245.7	1494.7	9997.3	0.06 %	1.52 %
`DSDP-infobj`	34	0	40.1	1162.5	3932.3	0.00 %	2.11 %
`DSDP-obj`	34	0	40.1	1163.1	3932.3	0.00 %	2.11 %
`DSDP-inf`	34	0	40.1	1166.6	3967.5	0.00 %	2.11 %
`SDPA-infer`	37	0	200.2	835.9	5405.3	15.05 %	0.03 %
`SDPA-infobj`	47	0	40.2	490.7	1937.7	8.46 %	1.60 %
`SDPA-obj`	47	0	40.2	490.6	1937.7	8.47 %	1.60 %
`SDPA-inf`	47	0	40.2	490.7	1937.7	8.46 %	1.60 %

TABLE 5. Results for DSDP/SDPA and different branching rules for the *min-k-partitioning* testset of 69 instances

settings	solved	aborts	nodes	time	iters	penalty	unsucc
`DSDP-infer`	54	0	308.6	464.5	13,010.7	0.04 %	1.06 %
`DSDP-infobj`	56	0	157.0	328.7	7811.8	0.01 %	0.27 %
`DSDP-obj`	54	0	191.1	377.9	9075.6	0.04 %	1.17 %
`DSDP-inf`	51	0	303.6	580.3	12,787.2	0.03 %	0.08 %
`SDPA-infer`	45	4	442.9	494.6	12,676.1	14.72 %	26.29 %
`SDPA-infobj`	46	3	255.4	384.7	8463.2	15.60 %	27.49 %
`SDPA-obj`	44	4	288.5	433.8	9308.5	15.08 %	27.82 %
`SDPA-inf`	37	1	456.1	632.4	14,548.0	14.35 %	33.41 %

The results for the cardinality constrained least squares instances are given in Table 4. For this testset, we get identical results for `infobj`, `obj`, and `inf` branching, since in the cardinality constrained least squares instances all binary variables have objective zero and the fallback is infeasibility branching. The `infer` branching rule does not perform well on these instances, since the branch-and-bound trees are much smaller than for truss topology design and less branching history information is available.

When looking at the SDP-solvers, SDPA is faster than DSDP by a factor of two and can solve many more instances, which might be due to the fact that these instances are completely dense. In such cases, one of the main advantages of the dual-scaling method implemented in DSDP, namely that it can exploit sparsity in the dual problem, cannot be exploited. For these instances, there are also no aborts for SDPA, even though the number of problems where the penalty formulation has to be used is still much higher than for DSDP.

Results for the minimum k-partitioning instances are given in Table 5. Here, the branching rules using the objective values perform much better than `infer` and `inf` branching, since the information about the particular instances is solely contained in the objective coefficients of the variables. The best results, however, are obtained by the combined `infobj` rule.

For the minimum k-partitioning instances, DSDP outperforms SDPA, which might be due to the fact that these instances are very sparse. This fact can be exploited better by a dual-scaling than a primal-dual algorithm, which also has to take care of the potentially dense primal matrix. Another important factor is that these problems are numerically the most troubling, since the Slater condition fails after fixing a single variable to one, as we explained in Section 10.3. This can also be seen in the results, with SDPA failing in more than 40 % of the nodes, while they do not seem to pose too many problems for DSDP.

TABLE 6. Results for DSDP/SDPA and different rules for the *complete testset* of 194 instances

settings	solved	aborts	nodes	time	iters	penalty	unsucc
`DSDP-infer`	120	0	522.5	786.9	19,302.1	0.08 %	0.91 %
`DSDP-infobj`	138	0	173.4	556.5	8565.4	0.04 %	0.87 %
`DSDP-obj`	129	0	233.2	699.8	10,498.9	0.25 %	1.68 %
`DSDP-inf`	127	0	234.1	749.1	10,623.8	0.22 %	1.20 %
`SDPA-infer`	122	11	488.8	584.8	12,170.7	11.82 %	10.73 %
`SDPA-infobj`	136	12	202.5	394.4	5805.1	9.45 %	11.01 %
`SDPA-obj`	136	6	248.1	455.6	6795.4	9.54 %	11.75 %
`SDPA-inf`	120	10	274.3	540.8	7490.0	8.98 %	13.45 %

Results for the whole testset can be found in Table 6 and Figure 1. The best overall results are obtained for the `infobj` branching rule. Inference branching does not seem to be a good choice for either solver, possibly because the branch-and-bound trees are too small compared to mixed-integer linear programs, but for SDPA it at least solves more instances than `inf` branching.

Between the solvers, SDPA is about 30 % faster than DSDP for each branching rule. When looking at the number of solved instances, however, the results for DSDP and SDPA are similar, which is again caused by a number of instances that get aborted when using SDPA, sometimes because the root node relaxation could not be solved, others because SDPA aborts inside the linear algebra subroutines. Recall that these problems can be solved by a certain choice of parameters, but we could not find a choice that works for all instances simultaneously. The number of branch-and-bound nodes is in general smaller for DSDP than for SDPA, because of the smaller number of unsuccessfully solved nodes, which need further branching, while they might be cut off if solved successfully. The number of SDP-iterations, however, is bigger for DSDP, because of the slower convergence of the dual-scaling method.

Over the whole testset, SDPA fails to solve about 20 % of the relaxations with the default formulation, while for DSDP it is less than 2 %. The penalty formulation does not play a big role for DSDP, since it is also used internally until the penalty variable can be fixed to 0. For SDPA, it allows to solve about 50 % of the problems that failed at first. In the remaining cases, we might still be able to generate lower bounds by finding optimal solutions for the penalty problem that are infeasible for the original formulation. This may be enough to cut these nodes off instead of generating a whole subtree that needs to be enumerated.

11.2.2. *Influence of Dual Fixing and the Heuristics.* We next investigate the influence of dual fixing as well as the fractional diving and randomized rounding heuristics on the performance of the solution process. These tests are performed for the fastest combination of solver and branching rule according to the tests in the last section, which is SDPA with `infobj` branching. In case of randomized rounding, we performed five rounds for different random seeds in every node in which the heuristic was called. We compare the following ten settings, with the first one being the default setting from the last section:

- `rootdive`: fractional diving in the root node; dual fixing disabled;
- `rootdive-dualfix`: fractional diving in the root node; dual fixing enabled;
- `dive10`: fractional diving with depth frequency 10; dual fixing disabled;

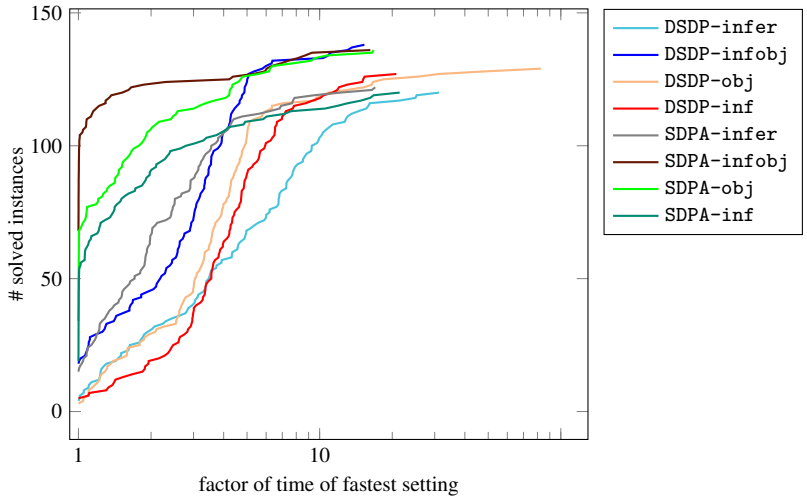

FIGURE 1. Performanceplot for DSDP/SDPA and different branching rules for the *complete testset* of 194 instances

- `dive10-dualfix`: fractional diving, depth frequency 10; dual fixing enabled;
- `nodive`: fractional diving disabled; dual fixing disabled;
- `nodive-dualfix`: fractional diving disabled; dual fixing enabled.
- `randroot`: randomized rounding in the root node; dual fixing disabled;
- `randroot-dualfix`: randomized rounding in root node; dual fixing enabled;
- `rand10`: randomized rounding with depth frequency 10; dual fixing disabled;
- `rand10-dualfix`: randomized rounding with depth frequency 10; dual fixing enabled.

Results for the whole testset can be found in Table 7 and Figure 2. Dual fixing shows to be a worthwhile technique, reducing solving times by 7 to 28 %, depending on the quality of the primal solutions, and in all cases increases the number of solved instances significantly.

Fractional diving is less successful, since it is too time consuming when used frequently. But it helps dual fixing by supplying good primal bounds. Furthermore, there are three instances that can only be solved using fractional diving and four more that need either fractional diving or randomized rounding. Randomized rounding performs very well by producing good primal solutions for propagation without taking too much time. With enabled dual fixing, a frequent use of the randomized rounding heuristic can reduce solving times by almost 30 %.

The results do not vary much with respect to the different testsets, but dual fixing seems to perform best for cardinality constrained least squares, while randomized rounding has largest impact for minimum k-partitioning.

The results are very similar when using DSDP instead of SDPA. The only notable difference is that for DSDP, the frequent use of the diving heuristic increases the number of unsolved relaxations significantly, but this does not seem to have a big influence on solving times.

11.3. Randomized Rounding in the Root Node. Pilanci et al. [59] used repeated randomized rounding to generate solutions for the cardinality constrained least

TABLE 7. Results for SDPA with combined infeasibility/objective branching with and without fractional diving, randomized rounding, and dual fixing for the *complete testset* of 194 instances

settings	solved	aborts	nodes	time	iters	penalty	unsucc	fixings
`rootdive`	136	12	243.7	394.4	7095.6	9.45 %	11.01 %	0.0
`rootdive-dualfix`	144	11	232.9	339.8	7237.2	8.98 %	11.32 %	6293.4
`dive10`	125	17	240.7	537.6	10,515.4	10.62 %	11.79 %	0.0
`dive10-dualfix`	139	19	197.6	395.3	9096.9	11.22 %	12.19 %	5629.4
`nodive`	141	13	261.6	350.3	5401.8	7.09 %	10.89 %	0.0
`nodive-dualfix`	148	12	261.0	319.9	5463.6	7.55 %	10.54 %	3346.5
`rootrand`	142	13	254.8	339.5	5284.2	9.99 %	11.10 %	0.0
`rootrand-dualfix`	149	12	247.4	261.9	5219.3	9.67 %	10.58 %	3998.5
`rand10`	146	15	252.1	316.2	5077.1	12.00 %	11.37 %	0.0
`rand10-dualfix`	156	15	218.3	228.5	4755.0	12.23 %	11.20 %	10,083.0

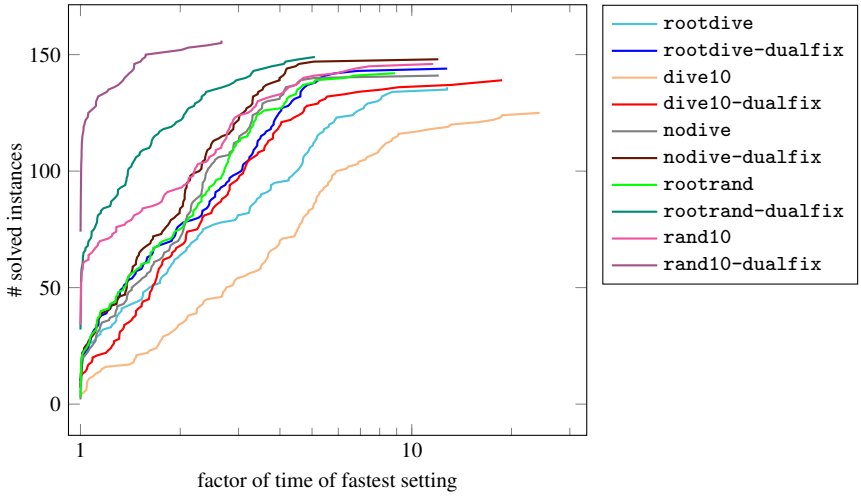

FIGURE 2. Performanceplot for SDPA with combined infeasibility/objective branching with and without fractional diving, randomized rounding, and dual fixing for the *complete testset* of 194 instances

TABLE 8. Data columns for presenting the results of the randomized rounding heuristic

column name	description
`success`	number of instances with at least one generated feasible solution
`optimal`	number of instances where the optimal solution could be found
`aborts`	number of instances aborted by the solver
`found`	arithmetic mean of number of improving solutions found by randomized rounding
`root-gap`	arithmetic mean of duality gap |primalbound − dualbound| / |dualbound| after the root node over all instances where at least one solution was found
`optimal-gap`	arithmetic mean of gap between best solution found by heuristic and optimal solution, given by |bestsolval − optsolval| / |optsolval|, over all instances with at least one solution found
`time`	shifted geometric mean of the solving time in seconds (shift $s = 10$)
`iters`	shifted geometric mean of the number of SDP-iterations (shift $s = 1000$)

squares problem. Using our MISDP-solver, we can now provide an empirical estimation of the quality of the produced solutions. We will also investigate the influence of the number of rounds of the randomized rounding heuristic, using 1, 10, 100, and 1000 rounds ([59] used 1000).

The results for the cardinality constrained least squares testset are given in Table 9, using column data as described in Table 8. When applying randomized

TABLE 9. Results after the root node using only randomized rounding for the *cardinality constrained least squares* testset of 65 instances

settings	success	optimal	aborts	found	root-gap	optimal-gap	time	iters
DSDP-1rand	59	17	0	0.9	48.58 %	44.61 %	50.7	144.6
DSDP-10rand	65	42	0	5.1	5.34 %	3.03 %	55.6	300.5
DSDP-100rand	65	46	0	31.2	2.84 %	0.85 %	69.5	1600.6
DSDP-1000rand	65	47	0	134.3	2.46 %	0.54 %	194.9	14,859.3
SDPA-1rand	49	9	0	0.8	50.90 %	46.63 %	15.2	79.1
SDPA-10rand	65	43	0	5.2	4.93 %	2.75 %	17.8	180.2
SDPA-100rand	65	46	0	32.8	2.80 %	0.83 %	26.3	978.4
SDPA-1000rand	65	48	0	133.6	2.54 %	0.63 %	101.5	8952.5

rounding at least ten times, very good results can be achieved. The generated solutions are feasible most of the time (they are only infeasible if more than k variables are rounded up), and we could even find an optimal solution for two thirds of the instances. With 100 or 1000 rounds, the best solution value found was less than one percent worse than the optimal solution value on average. In this case, however, the time needed is already approaching the levels of our branch-and-bound solver. Note that for this specific application, the solving times could be reduced by computing τ directly instead of solving an SDP with a single variable in every round of the randomized rounding heuristic, which is not possible for general MISDPs. Between DSDP and SDPA, the quality of the solutions does not differ, while the solving times are slower for DSDP by the same factor of two as for solving the problems to optimality.

We also tested the same approach for the other two applications. The results, however, are much worse. In case of minimum k-partitioning, feasible solutions were generated for up to four instances, which were, however, always optimal. For the truss topology instances, solutions for up to eight instances with an average optimality gap of around 15 % were found. Since we do not have to solve a final SDP in these cases, the running time of the randomized rounding heuristic is very small for these instances.

11.4. Slater Condition. In Section 5, we discussed under which theoretical conditions strong duality and the Slater constraint qualification are preserved in the subproblems of the tree. The Slater condition is of particular practical importance, since it is often required for convergence analyses of SDP-solvers. This motivates the experiments in the following, in which we estimate the proportion of subproblems in the tree for which the Slater condition holds.

For the results in this section, we solved all instances in the three testsets again, but tested every SDP-relaxation for the primal and dual Slater condition using the auxiliary problems explained in Section 9.4. These tests were carried out on a server with Intel Xeon E5-4650 CPUs with 2.7GHz, 20MB cache, and 8GB memory running Linux, with each computation performed single-threaded. The code was compiled with gcc 4.4.5 with `-O3` optimization. A time limit of one hour was used for each instance. Since the last section showed that dual fixing and the different heuristics can have significant influence on the amount of numerical problems encountered during solving the relaxations, we tested both DSDP and SDPA with the following three settings:

- `nodive`: objective & infeasibility branching with randomized rounding, diving heuristic, and dual fixing disabled;

TABLE 10. Statistics of Slater condition for the *truss topology* testset of 60 instances

	Dual Slater				Primal Slater		
settings	✓	✗	inf	?	✓	✗	?
`DSDP-nodive`	97.24 %	1.24 %	1.52 %	0.00 %	99.93 %	0.00 %	0.07 %
`DSDP-frac10-fix`	86.31 %	5.87 %	7.78 %	0.03 %	98.57 %	0.00 %	1.43 %
`DSDP-nodive-rand10-fix`	91.04 %	4.57 %	4.38 %	0.01 %	98.91 %	0.00 %	1.09 %
`SDPA-nodive`	95.69 %	0.86 %	1.83 %	1.62 %	98.32 %	0.00 %	1.68 %
`SDPA-frac10-fix`	85.64 %	4.53 %	6.86 %	2.98 %	94.50 %	0.00 %	5.50 %
`SDPA-nodive-rand10-fix`	89.04 %	3.58 %	4.39 %	2.99 %	96.28 %	0.00 %	3.72 %

- `frac10-fix`: objective & infeasibility branching with a diving heuristic frequency of 10; randomized rounding is disabled; dual fixing is enabled;
- `nodive-rand10-fix`: objective & infeasibility branching with a randomized rounding frequency of 10; fractional diving is disabled; dual fixing is enabled.

11.4.1. *Fulfillment of the Slater Condition.* In this section, we investigate how often the primal and dual Slater condition hold in practice when applying a branch-and-bound approach to the MISDP applications explained in Section 10. The tables give percentages of how often we could show that the primal and dual Slater condition holds (✓) for each subproblem, respectively, how often we could show that it does not hold (✗), and the percentage of SDP-relaxations where we either failed to solve the auxiliary problems (P-Slater) or (FC), or already found the subproblem to be infeasible in presolving, given in the column labeled "?". For the dual Slater condition we also indicate the amount of infeasible subproblems. All numbers are presented as arithmetic means.

The results for the truss topology testset are given in Table 10. The first thing to observe is that, as expected by our theoretical results, the primal Slater condition holds in all branch-and-bound nodes (not cut off in presolving and with the SDP-solver successfully solving the auxiliary problem (P-Slater)). The dual Slater condition holds in the root node, but can get lost after specific branchings for up to 15 % of the SDP-relaxations. When enabling the heuristics and dual fixing, the number of infeasible subproblems is increased: On the one hand, subproblems are pruned earlier, since the primal solutions are better and dual fixing generates tighter bounds. On the other hand, diving heuristics tend to produce a higher number of infeasible subproblems, by construction. However, also the number of feasible subproblems failing the dual Slater condition is increased to up to 5 %.

Results for the cardinality constrained least squares testset are given in Table 11. As we have seen in Section 10.2, the dual Slater condition theoretically holds for all SDP-relaxations. In our implementation, however, an upper bound on τ is inferred automatically, using the best known solution. But this might introduce subproblems that do not satisfy the Slater condition. This regularly happened in our numerical results, most often when applying the randomized rounding heuristic to generate tight primal bounds. However, the above mentioned upper bound has the benefit of early cutting off subtrees without optimal solutions.

The results for the minimum k-partitioning testset are given in Table 12. In this case, as explained in Section 10.3, the number of subproblems for which the dual Slater condition holds is very small, since fixing a single binary variable to 1 already causes it to fail.

TABLE 11. Statistics of Slater condition for the *cardinality constrained least squares* testset of 65 instances

	Dual Slater				Primal Slater		
settings	✓	✗	inf	?	✓	✗	?
DSDP-nodive	95.57 %	0.01 %	3.49 %	0.93 %	100.00 %	0.00 %	0.00 %
DSDP-frac10-fix	92.19 %	0.60 %	6.77 %	0.44 %	100.00 %	0.00 %	0.00 %
DSDP-nodive-rand10-fix	80.32 %	1.24 %	17.83 %	0.61 %	100.00 %	0.00 %	0.00 %
SDPA-nodive	94.01 %	0.03 %	5.68 %	0.29 %	96.99 %	0.00 %	3.01 %
SDPA-frac10-fix	85.80 %	1.43 %	11.86 %	0.91 %	93.59 %	0.00 %	6.41 %
SDPA-nodive-rand10-fix	79.88 %	1.93 %	18.19 %	0.00 %	90.32 %	0.00 %	9.68 %

TABLE 12. Statistics of Slater condition for the *min-k-partitioning* testset of 69 instances

	Dual Slater				Primal Slater		
settings	✓	✗	inf	?	✓	✗	?
DSDP-nodive	2.32 %	94.84 %	1.65 %	1.18 %	100.00 %	0.00 %	0.00 %
DSDP-frac10-fix	2.37 %	96.02 %	0.85 %	0.76 %	100.00 %	0.00 %	0.00 %
DSDP-nodive-rand10-fix	2.56 %	94.39 %	1.09 %	1.96 %	100.00 %	0.00 %	0.00 %
SDPA-nodive	1.91 %	91.43 %	4.51 %	2.15 %	98.35 %	0.00 %	1.65 %
SDPA-frac10-fix	2.15 %	91.30 %	1.59 %	4.96 %	99.46 %	0.00 %	0.54 %
SDPA-nodive-rand10-fix	2.02 %	92.56 %	3.62 %	1.80 %	98.30 %	0.00 %	1.70 %

TABLE 13. Statistics of Slater condition for the *complete testset* of 194 instances

	Dual Slater				Primal Slater		
settings	✓	✗	inf	?	✓	✗	?
DSDP-nodive	62.92 %	34.12 %	2.23 %	0.73 %	99.98 %	0.00 %	0.02 %
DSDP-frac10-fix	58.43 %	36.17 %	4.98 %	0.43 %	99.56 %	0.00 %	0.44 %
DSDP-nodive-rand10-fix	55.98 %	35.40 %	7.71 %	0.91 %	99.66 %	0.00 %	0.34 %
SDPA-nodive	60.95 %	33.54 %	4.16 %	1.34 %	97.86 %	0.00 %	2.14 %
SDPA-frac10-fix	54.16 %	36.21 %	6.65 %	2.99 %	96.05 %	0.00 %	3.95 %
SDPA-nodive-rand10-fix	53.14 %	36.29 %	9.11 %	1.46 %	94.87 %	0.00 %	5.13 %

Results for the whole testset are given in Table 13. The primal Slater condition holds for all of our problems, as expected from the theoretical results, while the dual Slater condition holds for slightly more than half the nodes overall, with one third failing it and the rest being infeasible.

11.4.2. *Influence of the Slater Condition on the Solver Behavior.* In the following, we investigate the influence of the Slater condition on the behavior of the solvers. We will use the same settings as in the last section and report the arithmetic means of the percentages of SDP-relaxations solved to the different outcomes given in Table 14. The results are split into subproblems in which both the primal and dual Slater condition holds, subproblems in which one fails, and subproblems which are infeasible. Note that we take averages over all SDP-relaxations occurring in all instances. Thus, the statistics will be dominated by the truss topology instances, since they usually require a significantly larger number of branch-and-bound nodes than the other types of problems.

Table 15 shows that for the whole testset, both solvers are very stable for subproblems in which the Slater condition holds, with over 99 % of the problems solved to optimality.

TABLE 14. Data columns for presenting the solver behavior

column name	description
`number`	the number of SDP relaxations with Slater condition holding in both subproblems / holding in at most one subproblem / showing infeasibility
`default`	the subproblem could be solved by the original formulation
`penalty`	the subproblem could be solved to optimality or infeasibility using the penalty formulations introduced in Section 6
`bound`	the penalty subproblem could be solved to generate a lower bound for the original, but the solution was not feasible in the original formulation
`unsucc`	even with the penalty formulation the problem could not be solved

TABLE 15. Statistics of solver fails when the primal and dual Slater condition holds for the *complete testset* of 194 instances

settings	number	default	penalty	bound	unsucc
`DSDP-nodive`	939,227	99.86 %	0.10 %	0.00 %	0.04 %
`DSDP-frac10-fix`	1,648,399	99.55 %	0.32 %	0.00 %	0.13 %
`DSDP-nodive-rand10-fix`	976,547	99.16 %	0.39 %	0.00 %	0.46 %
`SDPA-nodive`	872,872	99.97 %	0.01 %	0.00 %	0.02 %
`SDPA-frac10-fix`	1,397,142	99.88 %	0.04 %	0.00 %	0.08 %
`SDPA-nodive-rand10-fix`	749,151	99.96 %	0.01 %	0.00 %	0.03 %

TABLE 16. Statistics of solver fails when either the primal or dual Slater condition does not hold for the *complete testset* of 194 instances

settings	number	default	penalty	bound	unsucc
`DSDP-nodive`	27,176	87.51 %	0.10 %	0.00 %	12.36 %
`DSDP-frac10-fix`	119,671	75.65 %	0.03 %	0.01 %	24.31 %
`DSDP-nodive-rand10-fix`	55,791	67.72 %	0.11 %	0.01 %	32.15 %
`SDPA-nodive`	38,197	59.74 %	2.67 %	23.30 %	14.30 %
`SDPA-frac10-fix`	69,965	74.18 %	3.06 %	15.46 %	7.29 %
`SDPA-nodive-rand10-fix`	32,616	60.19 %	1.66 %	29.05 %	9.10 %

TABLE 17. Statistics of solver fails for infeasible subproblems for the *complete testset* of 194 instances

settings	number	default	penalty	bound	unsucc
`DSDP-nodive`	17,047	88.94 %	11.05 %	0.00 %	0.00 %
`DSDP-frac10-fix`	162,536	84.16 %	15.84 %	0.00 %	0.00 %
`DSDP-nodive-rand10-fix`	51,508	58.14 %	41.85 %	0.00 %	0.00 %
`SDPA-nodive`	20,928	12.15 %	87.85 %	0.00 %	0.00 %
`SDPA-frac10-fix`	124,991	18.97 %	81.03 %	0.00 %	0.00 %
`SDPA-nodive-rand10-fix`	53,088	30.75 %	69.25 %	0.00 %	0.00 %

In Table 16 the same statistics are given for those problems that do not satisfy the primal or dual Slater condition. Here, the number of problems solved with the original formulation drops below 90 % for all settings, even as low as 60 % for SDPA with some settings. DSDP usually cannot solve these problems even with the penalty formulation, since it is already based on a penalty formulation. For SDPA on the other hand, the penalty formulation is helpful, even though it can only generate a feasible solution for our original formulation about 10 % of the time. But in two out of three cases it at least enables us to compute an upper bound on the optimal objective value for the subtree, which in some cases will be enough to cut the subtree off. The penalty formulation leads to a lower number of

unsolved problems for SDPA than DSDP in this case, even though with the original formulation DSDP manages to solve more problems than SDPA.

For the infeasible subproblems given in Table 17, the difference between DSDP and SDPA is much larger. For DSDP, the performance is similar to the case of the Slater condition not holding, with a success rate between 60 and 90 % depending on the used settings. This is due to the fact that the penalty formulation handles infeasibility in the same way as the Slater condition and also allows to detect infeasibility using the value of the penalty variable. Note that after failing to solve the problems at first, our usage of the penalty formulation varies from that of DSDP in this case, since we will first solve the auxiliary Problem (FC) to detect infeasibility, which allows us to cut these problems off early before proceeding with the penalty formulation used in DSDP. For SDPA, however, the success rate drops from 60–75 % in the case of problems without the Slater condition to 10–30 % for infeasible subproblems. However, using the penalty formulation resolves all infeasibilities. This shows the importance of a dedicated handling of infeasibility detection within a branch-and-bound approach for mixed-integer semidefinite programming, such as our usage of Problem (FC).

12. Conclusions

The goal of this paper was to investigate generic solution approaches for mixed-integer semidefinite programs both in theory and practice. On the theoretical side, strong duality and the Slater condition are inherited to the subproblems, under certain conditions. These conditions are often naturally fulfilled in applications. On the negative side, there is a significant number of examples in which strict feasibility may get lost after branching.

A generic mixed-integer semidefinite solver should take these theoretical results into account by implementing several safe-guards against failures to solve the SDP-relaxation, including removing fixed variables, using a penalty formulation, and varying solver settings. Moreover, among the solver components, we discussed dual fixing, branching rules, and primal heuristics. We introduced basic methods for each of these areas, yielding a very positive impact on the solution process.

The computational results show that these techniques lead to a relatively stable behavior, allowing to solve small to medium sized instances of various applications within reasonable time. Nevertheless, more research and software development is necessary to further improve the performance, stability, and sizes of successfully solved instances. In fact, the sizes of the instances are still quite a bit away from what would be needed to use these techniques in most practical applications.

Future research should continue with improvements of the discussed solver components and implement interfaces to further SDP-solvers, e.g., MOSEK. Moreover, cutting planes should be exploited, see Section 8.4. Furthermore, we plan to investigate warm-starting techniques for the interior-point solvers in the implementation, for example, along proposals by Gondzio [37] and Benson and Shanno [13].

A viable alternative to using the penalty formulation might be facial reduction, see Borwein and Wolkowicz [18]. When using a homogeneous embedding algorithm for solving the SDP-relaxations as proposed by Ye et al. [77] and implemented in MOSEK, the needed facial reduction certificates could even be extracted from the results of the SDP-solver, as mentioned by Permenter et al. [57].

ACKNOWLEDGMENTS

The authors would like to thank the German Research Foundation (DFG) for funding, since parts of this research have been carried out within the Collaborative Research Center 805. Moreover, they thank Sonja Mars and Lars Schewe for initiating and preparing the first version of SCIP-SDP. Finally, the authors would like to thank the two anonymous referees for their constructive comments which significantly improved the presentation of the paper.

REFERENCES

[1] T. Achterberg. *Constraint Integer Programming*. PhD thesis, TU Berlin, 2007.

[2] T. Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[3] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33:42–54, 2004.

[4] U. Alon, N. Barkai, D. A. Notterman, K. Gish, S. Ybarra, D. Mack, and A. J. Levine. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proceedings of the National Academy of Sciences of the United States of America*, 96:6745–6750, 1999.

[5] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, Philadelphia, PA, third edition, 1999.

[6] M. F. Anjos, B. Ghaddar, L. Hupp, F. Liers, and A. Wiegele. Solving k-way graph partitioning problems to optimality: The impact of semidefinite relaxations and the bundle method. In M. Jünger and G. Reinelt, editors, *Facets of Combinatorial Optimization – Festschrift for Martin Grötschel*, pages 355–386. Springer, Berlin Heidelberg, 2013.

[7] M. Armbruster, M. Fügenschuh, C. Helmberg, and A. Martin. LP and SDP branch-and-cut algorithms for the minimum graph bisection problem: a computational comparison. *Mathematical Programming Computation*, 4(3):275–306, 2012.

[8] A. Atamtürk and V. Narayanan. Conic mixed-integer rounding cuts. *Mathematical Programming*, 122(1):1–20, 2010.

[9] R. Bellman and K. Fan. On systems of linear inequalities in Hermitian matrix variables. In *Convexity*, volume 7 of *Proceedings of Symposia in Pure Mathematics*, pages 1–11. American Mathematical Society, Providence, RI, 1963.

[10] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust Optimization*. Princeton University Press, 2009.

[11] A. Ben-Tal and A. Nemirovski. Robust truss topology design via semidefinite programming. *SIAM Journal on Optimization*, 7(4):991–1016, 1997.

[12] M. P. Bendsøe and O. Sigmund. *Topology Optimization: Theory, Methods and Applications*. Springer, Berlin and Heidelberg, 2003.

[13] H. Y. Benson and D. F. Shanno. An exact primal-dual penalty method approach to warmstarting interior-point methods for linear programming. *Computational Optimization and Applications*, 38(8):371–399, 2007.

[14] S. J. Benson and Y. Ye. Algorithm 875: DSDP5–software for semidefinite programming. *ACM Transactions on Mathematical Software*, 34(4):Article 16, 20 pages, 2008.

[15] T. Berthold. *Heuristic algorithms in global MINLP solvers*. PhD thesis, TU Berlin, 2014.

[16] T. Berthold. RENS – the optimal rounding. *Mathematical Programming Computation*, 6(1):33–54, 2014.

[17] T. Berthold, S. Heinz, M. E. Pfetsch, and S. Vigerske. Large neighborhood search beyond MIP. In L. D. Gaspero, A. Schaerf, and T. Stützle, editors, *Proceedings of the 9th Metaheuristics International Conference (MIC 2011)*, pages 51–60, 2011.

[18] J. Borwein and H. Wolkowicz. Regularizing the abstract convex program. *Journal of Mathematical Analysis and Applications*, 83(2):495–530, 1981.

[19] G. Braun, S. Fiorini, S. Pokutta, and D. Steurer. Approximation limits of linear programs (beyond hierarchies). *Mathematics of Operations Research*, 40(3):756–772, 2015.

[20] M. T. Çezik and G. Iyengar. Cuts for mixed $0-1$ conic programming. *Mathematical Programming*, 104:179–202, 2005.
[21] R. J. Dakin. A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, 8(3):250–255, 1965.
[22] E. Danna, E. Rothberg, and C. L. Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102(1):71–90, 2004.
[23] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.
[24] S. Drewes and S. Pokutta. Symmetry-exploiting cuts for a class of mixed-0/1 second-order cone programs. *Discrete Optimization*, 13:23–35, 2014.
[25] A. Eisenblätter. *Frequency Assignment in GSM Networks: Models, Heuristics, and Lower Bounds*. PhD thesis, TU Berlin, 2001.
[26] A. Eisenblätter. The semidefinite relaxation of the k-partition polytope is strong. In W. J. Cook and A. S. Schulz, editors, *Proceedings of the 9th International IPCO Conference on Integer Programming and Combinatorial Optimization*, volume 2337 of *Lecture Notes in Computer Science*, pages 273–290. Springer, Berlin Heidelberg, 2002.
[27] C. E. Ferreira, A. Martin, C. C. de Souza, R. Weismantel, and L. A. Wolsey. The node capacitated graph partitioning problem: A computational study. *Mathematical Programming*, 81:229–256, 1998.
[28] C. E. Ferreira, A. Martin, C. C. de Souza, R. Weismantel, and L. A. Wolsey. The node capacitated graph partitioning problem – benchmark instances. Available at www.ic.unicamp.br/~cid/Problem-instances/Graph-Partition, visited 03/2016.
[29] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming Series B*, 98(1-3):23–47, 2003.
[30] M. Fischetti and A. Lodi. Heuristics in mixed integer programming. In J. J. Cochran, L. A. Cox, P. Keskinocak, J. P. Kharoufeh, and J. C. Smith, editors, *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc., 2010.
[31] H. A. Friberg. Facial reduction heuristics and the motivational example of mixed-integer conic optimization. Technical report, Optimization-Online, 2016.
[32] T. Gally, M. E. Pfetsch, and S. Ulbrich. Online supplement to: A framework for solving mixed-integer semidefinite programs. available at http://www.opt.tu-darmstadt.de/scipsdp/downloads/OnlineSupplement-GallyPfetschUlbrich-FrameworkMISDP.pdf, visited 11/2016.
[33] G. Gamrath, T. Fischer, T. Gally, A. M. Gleixner, G. Hendel, T. Koch, S. J. Maher, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, S. Schenker, R. Schwarz, F. Serrano, Y. Shinano, S. Vigerske, D. Weninger, M. Winkler, J. T. Witt, and J. Witzig. The SCIP optimization suite 3.2. Technical report, ZIB-Report (15-60), 2016.
[34] B. Ghaddar, M. F. Anjos, and F. Liers. A semidefinite programming branch-and-cut algorithm for the minimum k-partition problem. *Annals of Operations Research*, 188(1):155–174, 2011.
[35] S. Ghosh. DINS, a MIP improvement heuristic. In M. Fischetti and D. P. Williamson, editors, *Integer Programming and Combinatorial Optimization (IPCO 2007)*, volume 4513 of *Lecture Notes in Computer Science*, pages 310–323. Springer, 2007.
[36] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the Association for Computing Machinery*, 42:1115–1145, 1995.
[37] J. Gondzio. Warm start of the primal-dual method applied in the cutting plane scheme. *Mathematical Programming*, 83(1):125–143, 1998.
[38] C. Helmberg. Fixing variables in semidefinite relaxations. *SIAM Journal on Matrix Analysis and Applications*, 21(3):952–969, 2000.
[39] C. Helmberg. Semidefinite programming for combinatorial optimization. Habilitationsschrift, TU Berlin, ZIB-Report ZR-00-34, Zuse-Institue Berlin, January 2000.
[40] C. Helmberg and F. Rendl. Solving quadratic (0,1)-problems by semidefinite programs and cutting planes. *Mathematical Programming*, 82:291–315, 1998.
[41] C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization*, 10(3):673–696, 2000.
[42] R. R. Hocking. The analysis and selection of variables in linear regression. *Biometrics*, 32(1):1–49, 1976.

[43] V. Kann, S. Khanna, J. Lagergren, and A. Panconesi. On the hardness of approximating MAX k-CUT and its dual. *Chicago Journal of Theoretical Computer Science*, 1997(2):1–18, 1997.
[44] K. Krishnan and J. E. Mitchell. A linear programming approach to semidefinite programming problems. Technical report, Dept. of Mathematical Sciences, RPI, Troy, 2001.
[45] K. Krishnan and J. E. Mitchell. A unifying framework for several cutting plane methods for semidefinite programming. *Optimization Methods and Software*, 21(1):57–74, 2006.
[46] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.
[47] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. John Wiley & Sons, Chichester, 1990.
[48] F. Liers, M. Jünger, G. Reinelt, and G. Rinaldi. Computing exact ground states of hard Ising spin glass problems by branch-and-cut. In A. Hartmann and H. Rieger, editors, *New Optimization Algorithms in Physics*, pages 47–68. Wiley, 2004.
[49] J. Löfberg. YALMIP: a toolbox for modeling and optimization in MATLAB. In *IEEE International Symposium on Computer Aided Control Systems Design*, pages 284–289, 2004.
[50] J. Löfberg. YALMIP. Available at http://users.isy.liu.se/johanl/yalmip/, visited 02/2016.
[51] S. Mars. *Mixed-Integer Semidefinite Programming with an Application to Truss Topology Design*. PhD thesis, FAU Erlangen-Nürnberg, 2013.
[52] S. Mars and L. Schewe. An SDP-package for SCIP. Technical report, TU Darmstadt and FAU Erlangen-Nürnberg, August 2012.
[53] J. E. Mitchell. Fixing variables and generating classical cutting planes when using an interior point branch and cut method to solve integer programming problems. *European Journal of Operational Research*, 97:139–148, 1997.
[54] S. Modaresi, M. R. Kılınç, and J. P. Vielma. Split cuts and extended formulations for mixed integer conic quadratic programming. *Operations Research Letters*, 43(1):10–15, 2015.
[55] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, New York, 1988.
[56] Y. Nesterov and A. Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Studies in Applied and Numerical Mathematics. SIAM, Philadelphia, 1994.
[57] F. Permenter, H. A. Friberg, and E. D. Andersen. Solving conic optimization problems via self-dual embedding and facial reduction: a unified approach. Technical report, Optimization-Online, 2015.
[58] A. Philipp, S. Ulbrich, Y. Cheng, and M. Pesavento. Multiuser downlink beamforming with interference cancellation using a SDP-based branch-and-bound algorithm. In *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Process (ICASSP)*, pages 7724–7728, 2014.
[59] M. Pilanci, M. J. Wainwright, and L. El Ghaoui. Sparse learning via Boolean relaxations. *Mathematical Programming Series B*, 151(1):62–87, 2015.
[60] L. Porkolab and L. Khachiyan. On the complexity of semidefinite programs. *Journal of Global Optimization*, 10:351–365, 1997.
[61] F. Rendl. Semidefinite relaxations for integer programming. In M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors, *50 Years of Integer Programming 1958–2008*, pages 687–726. Springer, 2009.
[62] 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.
[63] G. Rinaldi. Rudy. Available at http://www-user.tu-chemnitz.de/~helmberg/rudy.tar.gz, visited 03/2016.
[64] E. Rothberg. An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing*, 19(4):534–541, 2007.
[65] M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal of Computing*, 6(4):445–454, 1994.
[66] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, Chichester, 1986.
[67] SCIP–Solving Constraint Integer Programs. Available at http://scip.zib.de, visited 03/2016.
[68] SCIP-SDP. Available at http://www.opt.tu-darmstadt.de/scipsdp/, visited 03/2016.

[69] A. Shapiro and K. Scheinberg. Duality and optimality conditions. In H. Wolkowicz, R. Saigal, and L. Vandenberghe, editors, *Handbook of Semidefinite Programming*, pages 67–110. Kluwer Academic Publishers, 2000.

[70] R. Sotirov. SDP relaxations for some combinatorial optimization problems. In M. Anjos and J. Lasserre, editors, *Handbook of Semidefinite, Conic and Polynomial Optimization: Theory, Algorithms, Software and Applications*, volume 166 of *International Series in Operational Research and Management Science*, pages 795–820. Springer, 2012.

[71] M. J. Todd. Semidefinite optimization. *Acta Numerica*, 10:515–560, 2001.

[72] S. Vigerske. *Decomposition in Multistage Stochastic Programming and a Constraint Integer Programming Approach to Mixed-Integer Nonlinear Programming*. PhD thesis, HU Berlin, 2012.

[73] M. J. Wainwright. Sharp thresholds for high-dimensional and noisy sparsity recovery using ℓ_1-constrained quadratic programming (Lasso). *IEEE Transactions on Information Theory*, 55(5):2183–2202, 2009.

[74] M. Yamashita, K. Fujisawa, and M. Kojima. Implementation and evaluation of SDPA 6.0 (Semi-Definite Programming Algorithm 6.0). *Optimization Methods and Software*, 18:491–505, 2003.

[75] M. Yamashita, K. Fujisawa, K. Nakata, M. Nakata, M. Fukuda, K. Kobayashi, and K. Goto. A high-performance software package for semidefinite programs: SDPA 7. Research Report B-460, Dept. of Mathematical and Computing Science, Tokyo Institute of Technology, September 2010.

[76] Y. Ye. *Interior Point Algorithms: Theory and Analysis*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, New York, 1997.

[77] Y. Ye, M. J. Todd, and S. Mizuno. An $O(\sqrt{nl})$-iteration homogeneous and self-dual linear programming algorithm. *Mathematics of Operations Research*, 19:53–67, 1994.

TRISTAN GALLY, TECHNISCHE UNIVERSITÄT DARMSTADT, DEPARTMENT OF MATHEMATICS, DOLIVOSTRASSE 15, 64293 DARMSTADT, GERMANY

E-mail address: gally@mathematik.tu-darmstadt.de

MARC E. PFETSCH, TECHNISCHE UNIVERSITÄT DARMSTADT, DEPARTMENT OF MATHEMATICS, DOLIVOSTRASSE 15, 64293 DARMSTADT, GERMANY

E-mail address: pfetsch@mathematik.tu-darmstadt.de

STEFAN ULBRICH, TECHNISCHE UNIVERSITÄT DARMSTADT, DEPARTMENT OF MATHEMATICS, DOLIVOSTRASSE 15, 64293 DARMSTADT, GERMANY

E-mail address: ulbrich@mathematik.tu-darmstadt.de