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

A branch-and-cut algorithm for solving mixed-integer semidefinite optimization problems

Ken Kobayashi · Yuich Takano

Received: date / Accepted: date

Abstract We consider a cutting-plane algorithm for solving mixed-integer semidefinite optimization (MISDO) problems. In this algorithm, the positive semidefinite (psd) constraint is relaxed, and the resultant mixed-integer linear optimization problem is solved repeatedly, imposing at each iteration a valid inequality for the psd constraint. We prove the convergence properties of the algorithm. Moreover, to speed up the computation, we devise a branch-and-cut algorithm, in which valid inequalities are dynamically added during a branch-and-bound procedure. We test the computational performance of our cutting-plane and branch-and-cut algorithms for three types of MISDO problem: random instances, computing restricted isometry constants, and robust truss topology design. Our experimental results demonstrate that, for many problem instances, our branch-and-cut algorithm delivered superior performance compared with general-purpose MISDO solvers in terms of computational efficiency and stability.

Keywords mixed-integer optimization · semidefinite optimization · cutting-plane algorithm · branch-and-cut algorithm

1 Introduction

Mixed-integer semidefinite optimization (MISDO) problems involve minimizing a linear objective function subject to the constraints that a given matrix formed from the decision variables is positive semidefinite (psd) and some of the variables are integer-valued. Various nonlinear constraints can be expressed as psd

Ken Kobayashi (corresponding author)
Artificial Intelligence Laboratory, Fujitsu Laboratories LTD., 4-1-1 Kamikodanaka, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8588, Japan
Tel.: +81-44-754-2328
Fax: +81-44-754-2664
E-mail: ken-kobayashi@jp.fujitsu.com

Yuichi Takano
Faculty of Engineering, Information and Systems, University of Tsukuba, 1-1-1 Tennodai, Tsukuba-shi, Ibaraki 305-8577, Japan

constraints [39,42], and some important combinatorial optimization problems can be reformulated as (or approximated by) semidefinite optimization (SDO) problems [33,36]. The discrete nature of practical decision-making (e.g., a yes-or-no decision) is naturally described by integrality constraints [43]. All these advantages are provided by MISDO problems and so they have been used effectively in diverse fields, such as architecture [11,47], control systems [20,27,38,40], graph theory [3,4,34], signal processing [16,31], surgery planning [48], and statistical data analysis [2,30,37].

A natural method for solving MISDO problems is to use a branch-and-bound (B&B) algorithm. This algorithm has been employed extensively for various applications [2–4,11,31,34,35,47]. Recently, Gally et al. [17] developed a general-purpose MISDO solver, SCIP-SDP, by combining the B&B framework of SCIP [1] with SDO solvers that use interior-point methods. These B&B algorithms involve solving a continuous relaxation (i.e., SDO) problem at each node of the enumeration tree. In this case, however, we cannot implement a warm-starting strategy for efficiently solving a series of SDO problems because the interior-point methods do not receive a given initial solution. Moreover, the progress of the B&B algorithm is sometimes disrupted and returns an incorrect solution due to numerical instability, especially when Slater's condition does not hold for some of the SDO problems.

In this paper we focus on another method: the cutting-plane algorithm [21, 44]. Specifically, we shall extend the cutting-plane algorithms [22,24] that were originally developed for solving SDO problems. These algorithms can readily be extended to solving MISDO problems by removing the psd constraint and repeatedly solving the resultant mixed-integer linear optimization (MILO) problem in which a valid inequality is imposed at each iteration to exclude infeasible solutions for the removed psd constraint. This approach has the advantage of being able to use state-of-the-art optimization software (e.g., Gurobi or CPLEX) when solving MILO problems. However, to solve an MISDO problem with a sufficient degree of accuracy, we must deal with an exponentially large number of MILO problems [10]. A similar cutting-plane algorithm for solving MISDO problems has been implemented in the solver CUTSDP in YALMIP [25] and also extended to handle mixed-integer conic optimization problems [26]. To the best of our knowledge, however, the theoretical properties of the algorithm have not been fully described, and no detailed computational results have been reported.

The purpose of this paper is to describe an effective computational framework for solving MISDO problems. First, we formulate a cutting-plane algorithm for solving MISDO problems and prove its convergence properties. We then devise a high-performance branch-and-cut (B&C) algorithm on the basis of this cutting-plane algorithm. Specifically, we start by solving an MILO problem in which the psd constraint has been relaxed, and then we include valid inequalities for the constraint dynamically during the B&B procedure. We implement this B&C algorithm through the use of a callback function in the Gurobi Optimizer. We compare the computational performance of our algorithms with that of SCIP-SDP [17] and CUTSDP for three types of MISDO problem: random instances, computing restricted isometry constants [16], and robust truss topology design [47].

The main advantages of our B&C algorithm can be summarized as follows:

– Our B&C algorithm solves an MILO problem subject to a series of valid inequalities. In contrast to the existing B&B algorithms that handle a series of

SDO problems, a warm-starting strategy using the dual simplex method makes the B&C computation much faster.
– Our B&C algorithm can be implemented using sophisticated MILO software, such as Gurobi or CPLEX, and it deals with no SDO problems that may cause numerical instability. Hence, computation using our algorithm is very stable.
– Our B&C algorithm adds valid inequalities dynamically during the MILO computation. As a result, our algorithm needs to execute the B&B procedure only once, in contrast with the cutting-plane algorithm, which repeats the B&B procedure.

The remainder of this paper is organized as follows. In Section 2, we give a definition of the MISDO problems to be addressed. In Section 3, we present our cutting-plane algorithm and analyze its convergence properties. In Section 4, we describe our B&C algorithm for solving MISDO problems. In Section 5, we report the computational results of tests of our algorithms, and in Section 6 we conclude with a brief summary.

2 Mixed-integer Semidefinite Optimization Problem

In this section we introduce the MISDO problem that we consider in this paper.

2.1 Notation

We denote by $\boldsymbol{x} := (x_i) \in \mathbb{R}^n$ an n-dimensional column vector of real numbers. The zero vector of appropriate size is written as $\mathbf{0}$. The ℓ_p-norm of vector $\boldsymbol{x}$ is denoted by $\|\boldsymbol{x}\|_p := \left(\sum_{i=1}^n |x_i|^p\right)^{1/p}$.

We denote by $\boldsymbol{X} := (x_{ij}) \in \mathbb{R}^{m\times n}$ an $m \times n$ real matrix. The zero and identity matrices of appropriate sizes are written as $\boldsymbol{O}$ and $\boldsymbol{I}$, respectively. The trace $\mathrm{Tr}(\,\cdot\,)$ is the sum of the diagonal elements of a square matrix. The $\mathrm{diag}(\,\cdot\,)$ operator extracts the main diagonal from the matrix as a vector, and the $\mathrm{Diag}(\,\cdot\,)$ operator maps the vector to the diagonal matrix.

The set of all $n \times n$ real symmetric matrices is denoted by $\mathcal{S}^n$. The minimum eigenvalue of matrix $\boldsymbol{X} \in \mathcal{S}^n$ is denoted by $\lambda_{\min}(\boldsymbol{X})$. We write $\boldsymbol{X} \succeq \boldsymbol{O}$ if the matrix $\boldsymbol{X}$ is symmetric and positive semidefinite. The standard inner product of matrices $\boldsymbol{A} := (a_{ij}) \in \mathcal{S}^n$ and $\boldsymbol{B} := (b_{ij}) \in \mathcal{S}^n$ is defined as $\boldsymbol{A} \bullet \boldsymbol{B} := \sum_{i=1}^n \sum_{j=1}^n a_{ij} b_{ij}$.

2.2 Formulations

We focus on solving MISDO problems in the dual form:

$$\text{maximize} \quad \boldsymbol{b}^\top \boldsymbol{y} \tag{1}$$

$$\text{subject to} \quad \boldsymbol{A}(\boldsymbol{y}) := \boldsymbol{A}_0 - \sum_{i=1}^{m} \boldsymbol{A}_i y_i \succeq \boldsymbol{O}, \tag{2}$$

$$\boldsymbol{y} \in \{0,1\}^{m_{\mathrm{b}}} \times \mathbb{R}^{m_{\mathrm{c}}}, \tag{3}$$

where $\boldsymbol{b} := (b_i) \in \mathbb{R}^m$ and $\boldsymbol{A}_i \in \mathcal{S}^n$ $(i = 0, 1, \ldots, m)$ are given, and $\boldsymbol{y}$ is a vector of binary and continuous decision variables. Note that $m = m_{\mathrm{b}} + m_{\mathrm{c}}$ holds, where m_{b} and m_{c} are, respectively, the numbers of binary and continuous decision variables.

Alternatively, we may also consider MISDO problems in the primal form:

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{A}_0 \bullet \boldsymbol{X} && (4)\\ \text{subject to} \quad & \boldsymbol{A}_i \bullet \boldsymbol{X} = b_i, \quad \forall i = 1, 2, \ldots, m, && (5)\\ & \boldsymbol{X} \succeq \boldsymbol{O}, && (6)\\ & x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in \mathcal{Q}, && (7) \end{aligned}$$

where $\boldsymbol{X} := (x_{ij}) \in \mathcal{S}^n$ is a matrix of decision variables. Note that the matrix $\boldsymbol{X}$ has some binary-valued elements specified by the index set $\mathcal{Q}$. In the following sections, we shall deal with the problem (1)–(3), but our algorithms can readily be applied to the problem (4)–(7).

We assume that the feasible region of the problem (1)–(3) is bounded even if the psd constraint (2) is removed. Since

$$\boldsymbol{A}(\boldsymbol{y}) \succeq \boldsymbol{O} \;\Rightarrow\; \mathrm{diag}(\boldsymbol{A}(\boldsymbol{y})) \geq \boldsymbol{0},$$

we define the relaxed feasible region

$$\mathcal{Y} := \{\boldsymbol{y} \mid \mathrm{diag}(\boldsymbol{A}(\boldsymbol{y})) \geq \boldsymbol{0},\ \boldsymbol{y} \in \{0, 1\}^{m_{\mathrm{b}}} \times \mathbb{R}^{m_{\mathrm{c}}}\}.$$

Assumption 1 *The relaxed feasible region $\mathcal{Y}$ is bounded.*

Let $\boldsymbol{y}^*$ be an optimal solution to the problem (1)–(3). Note that Assumption 1 is fulfilled if the following constraint is incorporated into the definition of the feasible region:

$$-M \leq y_i \leq M, \quad m_{\mathrm{b}} + 1 \leq \forall i \leq m,$$

where M is a sufficiently large number such that this constraint is satisfied by $\boldsymbol{y}^*$.

3 Cutting-Plane Algorithm

Our cutting-plane algorithm for solving MISDO problems is an extension of the cutting-plane algorithms [22, 24] for solving SDO problems. It can also be regarded as an improved version of the extended cutting-plane algorithm [44] in the sense that our algorithm makes it possible to handle psd constraints.

Our algorithm starts with the relaxed MILO problem from which the psd constraint (2) has been removed:

$$\max\{\boldsymbol{b}^\top \boldsymbol{y} \mid \boldsymbol{y} \in \mathcal{Y}\}. \tag{8}$$

If $\mathcal{Y} = \emptyset$, the problem (1)–(3) is infeasible because its feasible region is contained in $\mathcal{Y}$. Otherwise, since the feasible region $\mathcal{Y}$ is bounded from Assumption 1, there exists an optimal solution $\hat{\boldsymbol{y}}$ to the problem (8).

We next check whether the psd constraint (2) is satisfied by the relaxed solution $\hat{\boldsymbol{y}}$. If $\lambda_{\min}(\boldsymbol{A}(\hat{\boldsymbol{y}})) \geq 0$, then $\boldsymbol{A}(\hat{\boldsymbol{y}})$ is psd, so we terminate the algorithm. Otherwise, let $\hat{\boldsymbol{d}} \in \mathbb{R}^n$ be the normalized eigenvector of $\boldsymbol{A}(\hat{\boldsymbol{y}})$ corresponding to the minimum eigenvalue. It follows that

$$\hat{\boldsymbol{d}}^\top \boldsymbol{A}(\hat{\boldsymbol{y}}) \hat{\boldsymbol{d}} = \lambda_{\min}(\boldsymbol{A}(\hat{\boldsymbol{y}})) < 0.$$

Accordingly, we can exclude the solution $\hat{\boldsymbol{y}}$ from the feasible region by incorporating the linear constraint of $\boldsymbol{y}$,

$$\hat{\boldsymbol{d}}^\top \boldsymbol{A}(\boldsymbol{y})\hat{\boldsymbol{d}} \geq 0. \tag{9}$$

We call this constraint a *valid inequality* for the psd constraint (2) because

$$\boldsymbol{A}(\boldsymbol{y}) \succeq \boldsymbol{O} \iff \left\langle \boldsymbol{d}^\top \boldsymbol{A}(\boldsymbol{y})\boldsymbol{d} \geq 0,\ \forall \boldsymbol{d} \in \mathbb{R}^n \right\rangle.$$

After constraint (9) is included, we solve the MILO problem (8) again. We repeat this process (i.e., solving the MILO problem and adding a valid inequality) until $\boldsymbol{A}(\hat{\boldsymbol{y}})$ becomes nearly psd:

$$\lambda_{\min}(\boldsymbol{A}(\hat{\boldsymbol{y}})) \geq -\varepsilon, \tag{10}$$

where ε is a sufficiently small non-negative number representing a tolerance for feasibility. In this case, constraints (3) and (10) are satisfied by $\hat{\boldsymbol{y}}$, and the value of the objective function is not less than the maximum (i.e., $\boldsymbol{b}^\top \hat{\boldsymbol{y}} \geq \boldsymbol{b}^\top \boldsymbol{y}^*$) because $\hat{\boldsymbol{y}}$ solves the relaxed problem (8). Such a solution is called an *ε-optimal solution*. Our cutting-plane algorithm is summarized as Algorithm 1; if the algorithm terminates, it proves the infeasibility of the problem (1)–(3) or yields an ε-optimal solution.

Algorithm 1 Cutting-Plane Algorithm for Solving MISDO Problems

Step 0 (Initialization) Let $\varepsilon \geq 0$ be a tolerance for feasibility. Define the initial feasible region as $\mathcal{F}_1 := \mathcal{Y}$. Set $k \leftarrow 1$.

Step 1 (MILO Solution) Solve the MILO problem,

$$\boldsymbol{y}^k \in \arg\max\{\boldsymbol{b}^\top \boldsymbol{y} \mid \boldsymbol{y} \in \mathcal{F}_k\}.$$

Step 2 (Termination Condition)

(a) If $\mathcal{F}_k = \emptyset$, then the problem (1)–(3) is infeasible.
(b) If $\lambda_{\min}(\boldsymbol{A}(\boldsymbol{y}^k)) \geq -\varepsilon$, then $\boldsymbol{y}^k$ is an ε-optimal solution.

Step 3 (Cut Generation) Update the feasible region,

$$\mathcal{F}_{k+1} \leftarrow \mathcal{F}_k \cap \{\boldsymbol{y} \mid (\boldsymbol{d}^k)^\top \boldsymbol{A}(\boldsymbol{y})\boldsymbol{d}^k \geq 0\},$$

where $\boldsymbol{d}^k$ is the normalized eigenvector of $\boldsymbol{A}(\boldsymbol{y}^k)$ corresponding to the minimum eigenvalue.

Step 4 Set $k \leftarrow k+1$ and return to Step 1.

Following the approach used in the case of solving SDO problems [22,23], we prove convergence of the algorithm.

Theorem 2 *For any $\varepsilon > 0$, Algorithm 1 terminates in a finite number of iterations.*

Proof Suppose that the algorithm does not terminate. The infinite sequence $\{(\boldsymbol{d}^k, \boldsymbol{y}^k)\}$ generated by the algorithm is bounded because $\|\boldsymbol{d}^k\| = 1$ and $\boldsymbol{y}^k \in \mathcal{Y}$ for all k. Therefore, we can choose a subsequence $\{(\boldsymbol{d}^k, \boldsymbol{y}^k) \mid k \in \mathcal{K}\}$ that converges to an accumulation point $(\bar{\boldsymbol{d}}, \bar{\boldsymbol{y}})$.

Since the termination condition is never satisfied, we have that for all $k \in \mathcal{K}$,

$$(\boldsymbol{d}^k)^\top \boldsymbol{A}(\boldsymbol{y}^k)\boldsymbol{d}^k = \lambda_{\min}(\boldsymbol{A}(\boldsymbol{y}^k)) < -\varepsilon.$$

It follows that

$$\lim_{k\to\infty\,(k\in\mathcal{K})} (\boldsymbol{d}^k)^\top \boldsymbol{A}(\boldsymbol{y}^k)\boldsymbol{d}^k = \bar{\boldsymbol{d}}^\top \boldsymbol{A}(\bar{\boldsymbol{y}})\bar{\boldsymbol{d}} \leq -\varepsilon. \tag{11}$$

For each $k < \ell$, $\boldsymbol{y}^\ell$ satisfies the kth valid inequality

$$(\boldsymbol{d}^k)^\top \boldsymbol{A}(\boldsymbol{y}^\ell)\boldsymbol{d}^k \geq 0.$$

It follows that

$$\lim_{k\to\infty\,(k\in\mathcal{K})} \lim_{\ell\to\infty\,(\ell\in\mathcal{K})} (\boldsymbol{d}^k)^\top \boldsymbol{A}(\boldsymbol{y}^\ell)\boldsymbol{d}^k = \bar{\boldsymbol{d}}^\top \boldsymbol{A}(\bar{\boldsymbol{y}})\bar{\boldsymbol{d}} \geq 0. \tag{12}$$

Eqs. (11) and (12) imply that $\varepsilon = 0$, which contradicts the hypothesis.

Theorem 3 *Suppose that $\varepsilon = 0$. Even if Algorithm 1 does not terminate, every accumulation point of the sequence of solutions $\{\boldsymbol{y}^k\}$ is an optimal solution to the problem* (1)–(3).

Proof Suppose that $\{(\boldsymbol{d}^k, \boldsymbol{y}^k) \mid k \in \mathcal{K}\}$ is a sequence that converges to an accumulation point $(\bar{\boldsymbol{d}}, \bar{\boldsymbol{y}})$. Note that $\lambda_{\min}(\boldsymbol{A}(\boldsymbol{y}^k))$ converges to $\lambda_{\min}(\boldsymbol{A}(\bar{\boldsymbol{y}}))$ as $k \to \infty$ $(k \in \mathcal{K})$ (see, e.g., Theorem 2.4.9.2 [19]). Due to Eq. (12), we have

$$\begin{aligned}\lambda_{\min}(\boldsymbol{A}(\bar{\boldsymbol{y}})) &= \lim_{k\to\infty\,(k\in\mathcal{K})} \lambda_{\min}(\boldsymbol{A}(\boldsymbol{y}^k)) \\ &= \lim_{k\to\infty\,(k\in\mathcal{K})} (\boldsymbol{d}^k)^\top \boldsymbol{A}(\boldsymbol{y}^k)\boldsymbol{d}^k = \bar{\boldsymbol{d}}^\top \boldsymbol{A}(\bar{\boldsymbol{y}})\bar{\boldsymbol{d}} \geq 0,\end{aligned} \tag{13}$$

which means that the matrix $\boldsymbol{A}(\bar{\boldsymbol{y}})$ is psd. Since $\boldsymbol{b}^\top \boldsymbol{y}^k \geq \boldsymbol{b}^\top \boldsymbol{y}^*$ for all k, we have that $\boldsymbol{b}^\top \bar{\boldsymbol{y}} \geq \boldsymbol{b}^\top \boldsymbol{y}^*$. This implies that the accumulation point $\bar{\boldsymbol{y}}$ is an optimal solution to the problem (1)–(3).

When all of the decision variables are binary, we can prove finite convergence of the algorithm even in the case $\varepsilon = 0$.

Theorem 4 *If $m_{\mathrm{c}} = 0$, Algorithm 1 terminates in a finite number of iterations.*

Proof Recall that Step 3 excludes the solution $\boldsymbol{y}^k$ from the feasible region $\mathcal{F}_k$ in the kth iteration. Since the number of possible solutions $\boldsymbol{y} \in \{0,1\}^m$ is 2^m, the algorithm terminates with $\mathcal{F}_k = \emptyset$ after at most 2^m iterations.

4 Branch-and-Cut Algorithm

Note that Step 1 of the cutting-plane algorithm solves an MILO problem at every iteration, and therefore we must execute the B&B algorithm many times from scratch. To resolve this issue, we develop a B&C algorithm that generates valid inequalities dynamically during the B&B procedure. Such dynamic constraint generation is known as *lazy constraint callback* in the optimization literature. This functionality is offered by modern optimization software (e.g., CPLEX or Gurobi) as a state-of-the-art computational feature. Lazy constraints have been employed to speed up the computation for various applications, including harvest scheduling [41], energy supply systems [46], distribution network design [29], robust optimization [8], and statistical model selection [9]. To the best of our knowledge, however, we are the first to use this approach to dealing with psd constraints.

Our method starts with the B&B algorithm for solving the relaxed MILO problem (8). Once a feasible solution $\hat{\boldsymbol{y}} \in \mathcal{Y}$ is found, the callback procedure is initiated; if the psd constraint (2) is violated by $\hat{\boldsymbol{y}}$, a valid inequality is appended to exclude $\hat{\boldsymbol{y}}$ from the feasible region. After that, we proceed with the B&B algorithm from the middle of the MILO computation. We continue this process until the optimality of the obtained solution is proved thorough the B&B procedure. This B&C algorithm is summarized as Algorithm 2. Although our algorithms are designed for solving the problem (1)–(3) of the dual form, they can readily be modified to solve the problem (4)–(7) of the primal form.

Algorithm 2 B&C Algorithm for Solving MISDO Problems

Step 0 (Initialization) Let $\varepsilon \geq 0$ be a tolerance for feasibility. Set the feasible region as $\mathcal{F} \leftarrow \mathcal{Y}$.

Step 1 (B&B Procedure) Start (or continue) the B&B algorithm for solving the MILO problem,

$$\max\{\boldsymbol{b}^\top \boldsymbol{y} \mid \boldsymbol{y} \in \mathcal{F}\}.$$

Step 2 (Callback Procedure) Once a feasible solution $\hat{\boldsymbol{y}} \in \mathcal{F}$ is found, go to the following steps:

Step 2.1 (Cut Generation) If $\lambda_{\min}(\boldsymbol{A}(\hat{\boldsymbol{y}})) < -\varepsilon$, update the feasible region,

$$\mathcal{F} \leftarrow \mathcal{F} \cap \{\boldsymbol{y} \mid \hat{\boldsymbol{d}}^\top \boldsymbol{A}(\boldsymbol{y})\hat{\boldsymbol{d}} \geq 0\},$$

where $\hat{\boldsymbol{d}}$ is the normalized eigenvector of $\boldsymbol{A}(\hat{\boldsymbol{y}})$ corresponding to the minimum eigenvalue.

Step 2.2 Return to Step 1.

5 Computational Experiments

We performed several computational experiments to evaluate the effectiveness of our algorithms for solving MISDO problems.

5.1 Experimental Design

We compare the computational performance of the following methods:

SCIP: MISDO solver SCIP-SDP[1] [17],
CUTSDP: MISDO solver CUTSDP[2] in YALMIP [25],
CPA: cutting-plane algorithm (Algorithm 1),
B&C: B&C algorithm (Algorithm 2).

We used SCIP-SDP 3.1.0 on the NEOS Server[3] [12], where the B&B framework of SCIP[4] 5.0.0 [18] and the SDO solver DSDP[5] 5.8 [6] were combined in the default configuration. CUTSDP is based on a two-phase cutting-plane algorithm; the first phase deals with continuous relaxation (i.e., linear optimization) problems to generate valid inequalities for the psd constraint (2), and the second phase starts solving a sequence of MILO problems (similarly to Algorithm 1) with the valid inequalities generated in the first phase. Note also that in this algorithm, more than one valid inequality can be added at one iteration. CUTSDP in MATLAB R2015b was used in the default configuration, and CPA and B&C were implemented in the Python language; MILO problems were solved using the Gurobi Optimizer[6] 8.0, and its callback function was employed for Step 2 of Algorithm 2. The tolerance for feasibility in CPA and B&C was set as $\varepsilon := 10^{-5} \cdot (1 + \|\boldsymbol{b}\|_1)$ on the basis of the DIMACS error [28], where $\boldsymbol{b}$ is the coefficient vector of the objective function. These computations (i.e., CUTSDP, CPA, and B&C) were performed on a Windows 7 PC with an Intel Core i7-4790 CPU (3.60 GHz) and 16 GB of memory. The computation of each method was terminated if it did not finish by itself within 7200 s. In these cases, the results obtained at 7200 s were taken as the final outcome.

The column labels used in the tables of experimental results are defined as follows.

n: size of a psd matrix in constraints (2), (20), and (22),
m_{b}: number of binary decision variables,
m_{c}: number of continuous decision variables,
Time: computation time in seconds,
#Abort: number of times that the computation was aborted by a numerical issue or a memory shortage,
#Limit: number of times that the computation reached the time limit of 7200 s,
#Cuts: number of valid inequalities generated by CUTSDP, CPA, and B&C,
#Nodes: number of nodes in the enumeration tree formed by SCIP and B&C.

Following Gally et al. [17], we use the shifted geometric mean to aggregate results of random instances. The shifted geometric mean of values $x_1, x_2, \ldots, x_n$ is defined as

$$\left(\prod_{j=1}^{n} (x_j + s) \right)^{1/n} - s,$$

where the shift s was set to 10, 1000, and 100 for "Time," "#Cuts," and "#Nodes," respectively, as in Gally et al. [17].

[1] http://www.opt.tu-darmstadt.de/scipsdp/
[2] https://yalmip.github.io/solver/cutsdp/
[3] https://neos-server.org/neos/
[4] http://scip.zib.de/
[5] http://www.mcs.anl.gov/hs/software/DSDP/
[6] http://www.gurobi.com/

5.2 Random Instances

In this subsection, we report experimental results for solving random instances of the problem (1)–(3).

5.2.1 Instance Generation

Following Yamashita et al. [45], we generated random instances of the problem (1)–(3). First, we constructed $\tilde{\boldsymbol{y}}$ as a feasible solution: $\tilde{y}_i$ was randomly chosen from $\{0, 1\}$ for $i \leq m_{\rm b}$ and was drawn from a uniform distribution on the interval $[0, 1]$ for $i \geq m_{\rm b} + 1$. The elements of each of the matrices $\boldsymbol{A}_i$ $(i = 1, 2, \ldots, m)$ were drawn from a uniform distribution on the interval $[-1, 1]$. We next set $\boldsymbol{A}_0 := \alpha \boldsymbol{I} + \sum_{i=1}^{m} \boldsymbol{A}_i \tilde{y}_i$, where α is a positive-valued parameter ensuring that matrix $\boldsymbol{A}(\tilde{\boldsymbol{y}})$ is psd. We then set $b_i := \boldsymbol{A}_i \bullet \boldsymbol{I}$ for $i = 1, 2, \ldots, m$.

5.2.2 Experimental Results

Table 1 gives the computational results for the random instances case. Five instances of the problem (1)–(3) were generated for each tuple $(n, m_{\rm b}, m_{\rm c}, \alpha)$, and the shifted geometric means of "Time," "#Cuts," and "#Nodes" were computed.

We can see from Table 1 that our algorithm B&C was always the fastest among the four methods. The number of generated nodes in an enumeration tree was much larger for B&C than for SCIP; however, the series of continuous relaxation problems at such nodes can be solved efficiently in the case of B&C computation.

CPA was often slower than SCIP, but it is noteworthy that the SCIP computations were aborted in some instances by numerical issues. In contrast to SCIP, our algorithms do not handle SDO problems, so they can solve MISDO problems without encountering numerical instability. A typical example is the case of $(n, m_{\rm b}, m_{\rm c}, \alpha) = (30, 30, 60, 1.0)$: the computations of CPA and B&C finished on average in 2069.8 s and 18.8 s, respectively, but the SCIP computations were aborted in all five instances.

CPA was faster than CUTSDP except for the case of $(n, m_{\rm b}, m_{\rm c}, \alpha) = (15, 30, 30, 1.0)$. The number of valid inequalities generated by CPA was also smaller than that generated by CUTSDP. This is presumably because CUTSDP produced many unnecessary constraints in the first phase, and these constraints slowed down each MILO computation in the second phase.

5.3 Computing Restricted Isometry Constants

Here, we compare the performance of the algorithms for the problem of computing restricted isometry constants [16].

5.3.1 Problem Formulation

Let us consider a system of linear equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where the coefficient matrix $\boldsymbol{A} \in \mathbb{R}^{\mu \times n}$ and the right-hand side vector $\boldsymbol{b} \in \mathbb{R}^{\mu}$ are given. Compressed sensing [5]

Table 1 Results for random instances of the problem (1)–(3)

n	m_{b}	m_{c}	α	Method	Time	#Abort	#Limit	#Cuts	#Nodes
30	30	30	1.0	SCIP	61.5	0	0	—	25.4
				CUTSDP	378.4	0	0	1541.2	—
				CPA	131.0	0	0	543.6	—
				B&C	4.2	0	0	556.2	1071.7
15	30	30	1.0	SCIP	35.9	1	0	—	63.4
				CUTSDP	253.3	0	0	1279.7	—
				CPA	315.8	0	0	524.8	—
				B&C	4.4	0	0	770.8	5270.1
60	30	30	1.0	SCIP	142.2	1	0	—	23.0
				CUTSDP	1086.7	0	0	2235.7	—
				CPA	186.3	0	0	633.1	—
				B&C	4.7	0	0	628.8	1010.8
30	15	30	1.0	SCIP	34.6	2	0	—	15.7
				CUTSDP	158.1	0	0	1478.7	—
				CPA	78.0	0	0	540.3	—
				B&C	2.1	0	0	546.6	629.2
30	60	30	1.0	SCIP	128.7	0	0	—	51.6
				CUTSDP	978.8	0	0	1624.6	—
				CPA	439.8	0	0	553.8	—
				B&C	5.3	0	0	584.2	5183.2
30	30	15	1.0	SCIP	30.6	0	0	—	24.1
				CUTSDP	83.5	0	0	690.7	—
				CPA	14.4	0	0	185.7	—
				B&C	0.7	0	0	199.1	446.6
30	30	60	1.0	SCIP	—	5	—	—	—
				CUTSDP	2637.0	0	0	3858.8	—
				CPA	2069.8	0	0	1559.3	—
				B&C	18.8	0	0	1617.3	2977.7
30	30	30	0.1	SCIP	22.8	3	0	—	1.0
				CUTSDP	282.5	0	0	1273.0	—
				CPA	145.1	0	0	445.7	—
				B&C	2.0	0	0	431.1	474.2
30	30	30	10.0	SCIP	75.3	2	0	—	47.2
				CUTSDP	144.0	0	0	995.1	—
				CPA	97.8	0	0	401.8	—
				B&C	55.1	0	0	3947.6	13111.2
45	45	45	1.0	SCIP	164.8	0	0	—	38.8
				CUTSDP	3046.6	0	0	3107.7	—
				CPA	1123.8	0	0	1066.5	—
				B&C	11.8	0	0	1112.2	2403.8
60	60	60	1.0	SCIP	558.0	0	0	—	50.0
				CUTSDP	7169.6	0	4	4468.7	—
				CPA	5069.8	0	0	1644.0	—
				B&C	31.0	0	0	1663.5	4125.5

focuses on finding the sparsest solution $\hat{\boldsymbol{x}}^0$ to an undetermined linear system with $\mu \leq n$:

$$\hat{\boldsymbol{x}}^0 := \arg\min\{\|\boldsymbol{x}\|_0 \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\},$$

where $\|\boldsymbol{x}\|_0$ denotes the number of nonzero elements of vector $\boldsymbol{x} \in \mathbb{R}^n$. This problem is known to be NP-hard [5].

The lower and upper restricted isometry constants (RICs) [15] of order κ are defined as

$$\alpha_\kappa := \min\{\|\boldsymbol{A}\boldsymbol{x}\|_2^2 \mid \|\boldsymbol{x}\|_2^2 = 1,\ \|\boldsymbol{x}\|_0 \le \kappa\}, \tag{14}$$

$$\beta_\kappa := \max\{\|\boldsymbol{A}\boldsymbol{x}\|_2^2 \mid \|\boldsymbol{x}\|_2^2 = 1,\ \|\boldsymbol{x}\|_0 \le \kappa\}. \tag{15}$$

When certain conditions on RICs are fulfilled, the sparsest solution $\hat{\boldsymbol{x}}^0$ coincides with the following ℓ_1-norm minimizer [15]:

$$\hat{\boldsymbol{x}}^1 := \arg\min\{\|\boldsymbol{x}\|_1 \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}.$$

Gally and Pfetsch [16] proved that exact values of α_κ and β_κ can be computed by solving the following MISDO problem:

$$\begin{aligned}
\text{minimize/maximize} \quad & \operatorname{Tr}(\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{X}) && (16)\\
\text{subject to} \quad & \operatorname{Tr}(\boldsymbol{X}) = 1, && (17)\\
& \sum_{j=1}^{\nu} z_j \le \kappa, && (18)\\
& -\frac{1}{2} z_j \le x_{ij} \le \frac{1}{2} z_j, \quad \forall i, j = 1, 2, \ldots, n, && (19)\\
& \boldsymbol{X} \succeq \boldsymbol{O}, && (20)\\
& z_j \in \{0, 1\}, \quad \forall j = 1, 2, \ldots, n, && (21)
\end{aligned}$$

where $\boldsymbol{X} := (x_{ij}) \in \mathcal{S}^n$ and $\boldsymbol{z} := (z_j) \in \{0, 1\}^n$ are decision variables.

5.3.2 Experimental Results

Tables 2 and 3 give the results for computing lower and upper RICs, respectively. The size of the problem (16)–(21) is represented by $n \in \{10, 15, 20, 25, 30, 35\}$, and the order of RICs is $\kappa \in \{3, 5\}$. Note that the number of linear equations in the system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ was fixed to $\mu = 10$ because it is independent of the size of the problem (16)–(21). As in the case of Gaussian-distributed matrices [16], each element of matrix $\boldsymbol{A}$ was drawn from the standard normal distribution. Five instances of matrix $\boldsymbol{A}$ were generated for each pair (n, κ), and the shifted geometric means of "Time," "#Cuts," and "#Nodes" were computed.

We can see from Tables 2 and 3 that, as was the case for random instances, B&C was the fastest among the four methods. CPA was also faster than SCIP and CUTSDP in the case of computing RICs. We notice that when the problem size (i.e., n) was large, the SCIP computation was frequently aborted due to a numerical issue or a memory shortage. Indeed, when $n \ge 20$, SCIP failed to complete the computation for 39 out of 40 instances of computing lower RICs (Table 2) and for 31 out of 40 instances of computing upper RICs (Table 3). CUTSDP always reached the time limit of 7200 s when computing lower RICs with $n \ge 20$ (Table 2) and upper RICs with $n \ge 25$ (Table 3). Additionally, CUTSDP generated a much greater number of valid inequalities than CPA did.

Table 2 Results for computing lower RICs ($\mu = 10$)

n	κ	m_{b}	m_{c}	Method	Time	#Abort	#Limit	#Cuts	#Nodes
10	3	10	45	SCIP	6.1	0	0	—	27.6
				CUTSDP	21.9	0	0	730.5	—
				CPA	1.7	0	0	55.4	—
				B&C	1.0	0	0	479.2	751.6
10	5	10	45	SCIP	16.6	2	0	—	91.4
				CUTSDP	50.7	0	0	945.9	—
				CPA	14.0	0	0	183.1	—
				B&C	2.3	0	0	1087.0	2827.2
15	3	15	105	SCIP	140.3	0	0	—	187.6
				CUTSDP	2468.1	0	1	4369.6	—
				CPA	11.5	0	0	107.6	—
				B&C	3.8	0	0	943.5	1864.8
15	5	15	105	SCIP	693.4	0	0	—	891.4
				CUTSDP	7200.0	0	5	4296.1	—
				CPA	237.9	0	0	342.4	—
				B&C	14.1	0	0	1718.1	6494.2
20	3	20	190	SCIP	1037.6	4	0	—	373.0
				CUTSDP	7200.0	0	5	5478.4	—
				CPA	58.7	0	0	191.7	—
				B&C	8.9	0	0	1155.7	3031.0
20	5	20	190	SCIP	—	5	—	—	—
				CUTSDP	7200.0	0	5	5153.0	—
				CPA	1938.7	0	0	669.5	—
				B&C	112.5	0	0	533.3	21239.5
25	3	25	300	SCIP	—	5	—	—	—
				CUTSDP	7200.0	0	5	5149.2	—
				CPA	188.4	0	0	294.4	—
				B&C	16.9	0	0	1407.6	5435.7
25	5	25	300	SCIP	—	5	—	—	—
				CUTSDP	7200.0	0	5	5332.3	—
				CPA	7200.0	0	5	1178.9	—
				B&C	373.0	0	0	1008.0	60996.7
30	3	30	435	SCIP	—	5	—	—	—
				CUTSDP	7200.0	0	5	5116.3	—
				CPA	616.48	0	0	449.1	—
				B&C	45.33	0	0	1123.2	9828.7
30	5	30	435	SCIP	—	5	—	—	—
				CUTSDP	7200.0	0	5	5619.5	—
				CPA	7200.0	0	5	2104.3	—
				B&C	1343.6	0	0	2272.5	136486.1
35	3	35	595	SCIP	—	5	—	—	—
				CUTSDP	7200.0	0	5	6635.7	—
				CPA	2257.4	0	0	652.3	—
				B&C	102.1	0	0	609.7	16269.0
35	5	35	595	SCIP	—	5	—	—	—
				CUTSDP	7200.0	0	5	6696.3	—
				CPA	7200.0	0	5	2776.7	—
				B&C	3802.6	0	0	3336.7	225854.9

Table 3 Results for computing upper RICs ($\mu = 10$)

n	κ	m_{b}	m_{c}	Method	Time	#Abort	#Limit	#Cuts	#Nodes
10	3	10	45	SCIP	5.4	0	0	—	26.3
				CUTSDP	13.5	0	0	416.7	—
				CPA	1.5	0	0	30.2	—
				B&C	0.6	0	0	308.4	429.1
10	5	10	45	SCIP	9.7	0	0	—	74.3
				CUTSDP	53.9	0	0	826.5	—
				CPA	4.8	0	0	77.8	—
				B&C	2.0	0	0	880.4	2233.6
15	3	15	105	SCIP	41.3	0	0	—	51.7
				CUTSDP	231.3	0	0	1091.7	—
				CPA	3.8	0	0	34.2	—
				B&C	2.7	0	0	683.0	1465.0
15	5	15	105	SCIP	54.5	0	0	—	76.6
				CUTSDP	2657.9	0	3	2028.3	—
				CPA	22.8	0	0	93.7	—
				B&C	6.2	0	0	1084.2	2608.2
20	3	20	190	SCIP	322.0	0	0	—	93.1
				CUTSDP	1479.7	0	1	1177.4	—
				CPA	8.1	0	0	34.8	—
				B&C	3.8	0	0	511.9	1662.8
20	5	20	190	SCIP	605.9	2	0	—	130.8
				CUTSDP	3651.5	0	3	922.2	—
				CPA	129.2	0	0	121.9	—
				B&C	14.6	0	0	997.3	6100.2
25	3	25	300	SCIP	1224.0	4	0	—	137.0
				CUTSDP	7200.0	0	5	1072.6	—
				CPA	23.9	0	0	46.4	—
				B&C	7.9	0	0	681.2	3211.8
25	5	25	300	SCIP	—	5	—	—	—
				CUTSDP	7200.0	0	5	1106.3	—
				CPA	1285.2	0	0	165.4	—
				B&C	68.9	0	0	922.4	13854.4
30	3	30	435	SCIP	—	5	—	—	—
				CUTSDP	7200.0	0	5	1185.4	—
				CPA	40.8	0	0	44.6	—
				B&C	14.7	0	0	654.5	4552.8
30	5	30	435	SCIP	—	5	—	—	—
				CUTSDP	7200.0	0	5	1424.9	—
				CPA	2380.4	0	0	168.9	—
				B&C	177.9	0	0	1199.2	24939.4
35	3	35	595	SCIP	—	5	—	—	—
				CUTSDP	7200.0	0	5	1519.3	—
				CPA	92.4	0	0	46.8	—
				B&C	26.5	0	0	301.3	5424.4
35	5	35	595	SCIP	—	5	—	—	—
				CUTSDP	7200.0	0	5	1887.3	—
				CPA	6426.1	0	4	158.4	—
				B&C	318.0	0	0	938.1	56816.4

5.4 Robust Truss Topology Design

In this subsection, we report the experimental results for our tests of the four algorithms on the problem of robust truss topology design [47].

5.4.1 Problem Formulation

Let us consider an undirected graph called the *ground structure* in d-dimensional space. We suppose that some nodes are fixed, and d-dimensional forces act on the q unfixed nodes; so the external load vector is defined as $\boldsymbol{f} \in \mathbb{R}^{\nu}$ with $\nu := d \cdot q$. The index set in the external load vector is denoted by $\mathcal{J} := \{1, 2, \ldots, \nu\}$. The index set of edges in the ground structure is denoted by $\mathcal{I} := \{1, 2, \ldots, \mu\}$, which represents candidate members (or bars) of a truss. Truss topology design involves choosing a subset of members and determining their cross-sectional areas so that the truss structure will be lightweight but sufficiently rigid.

We begin by introducing the decision variables. We denote by $\boldsymbol{a} := (a_i)_{i \in \mathcal{I}} \in \mathbb{R}^{\mu}$ the vector of member cross-sectional areas. Additionally, the vector $\boldsymbol{p} := (p_j)_{j \in \mathcal{J}} \in \{0,1\}^{\nu}$ corresponds to the existence of each node, and the vector $\boldsymbol{t} := (t_i)_{i \in \mathcal{I}} \in \{0,1\}^{\mu}$ represents the selection of members in the truss structure.

Given matrices $\boldsymbol{K}_i \in \mathcal{S}^{\nu}$ $(i \in \mathcal{I})$, the stiffness matrix is defined as $\boldsymbol{K}(\boldsymbol{a}) := \sum_{i \in \mathcal{I}} a_i \boldsymbol{K}_i$. The following matrix is used to express uncertainty about the external loads applied at each node in the truss structure:

$$\boldsymbol{Q} := (\tilde{\boldsymbol{f}}, r\boldsymbol{v}_1, r\boldsymbol{v}_2, \ldots, r\boldsymbol{v}_{\nu-1}) \in \mathbb{R}^{\nu \times \nu},$$

where $\tilde{\boldsymbol{f}} \in \mathbb{R}^{\nu}$ is the nominal external load, r is the level of uncertainty, and $\boldsymbol{v}_j \in \mathbb{R}^{\nu}$ $(j = 1, 2, \ldots, \nu-1)$ are orthonormal basis vectors of the orthogonal complement of $\tilde{\boldsymbol{f}}$. When an ellipsoidal uncertainty set of external loads [7] is employed, the worst-case compliance constraint is posed as

$$\begin{pmatrix} \bar{\tau}\boldsymbol{I} & (\mathrm{Diag}(\boldsymbol{p})\boldsymbol{Q})^{\top} \\ \mathrm{Diag}(\boldsymbol{p})\boldsymbol{Q} & \boldsymbol{K}(\boldsymbol{a}) \end{pmatrix} \succeq \boldsymbol{O}, \tag{22}$$

where $\bar{\tau}$ is the upper-bound parameter for the compliance; see Yonekura and Kanno [47] for the details.

We define $\mathcal{J}_{\mathrm{f}} \subseteq \mathcal{J}$ to represent the nodes that should remain in the truss structure. Other nodes can be removed, but we must maintain the nodes that are connected to at least one member. Such constraints can be expressed as

$$0 \leq p_j \leq 1, \quad \forall j \in \mathcal{J} \setminus \mathcal{J}_{\mathrm{f}}, \tag{23}$$

$$t_i \leq p_j, \quad \forall i \in \mathcal{I}_{\hat{k}(j)}, \forall j \in \mathcal{J} \setminus \mathcal{J}_{\mathrm{f}}, \tag{24}$$

$$p_j = 1, \quad \forall j \in \mathcal{J}_{\mathrm{f}}, \tag{25}$$

where $\mathcal{I}_{\hat{k}(j)} \subseteq \mathcal{I}$ is the set of members that are connected to the node associated with the jth element of $\tilde{\boldsymbol{f}}$.

The following constraints are imposed on the member cross-sectional areas:

$$a_{\min} t_i \leq a_i \leq a_{\max} t_i, \quad \forall i \in \mathcal{I}, \tag{26}$$

$$t_i \in \{0,1\}, \quad \forall i \in \mathcal{I}, \tag{27}$$

where $a_{\min}$ and $a_{\max}$ are, respectively, lower and upper bounds on cross-sectional areas.

The robust truss topology design minimizing the structural volume is formulated as the following MISDO problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{i \in \mathcal{I}} \ell_i a_i \\ \text{subject to} \quad & \text{Eqs. (22)–(27)}, \end{aligned} \tag{28}$$

where ℓ_i is the length of the ith member. Although $\boldsymbol{p}$ is treated as a vector of continuous variables, the binary constraint $\boldsymbol{p} \in \{0,1\}^{\nu}$ can be satisfied.

5.4.2 Experimental Results

We considered a two-dimensional lattice as the ground structure; nodes were placed on lattice points, and every pair of distinct nodes was connected by a unique edge. Note that the longer edge was removed if two edges overlapped along the same straight line, and the remaining edges were treated as candidate members.

As explained below, the settings of parameters were the same as those used by Yonekura and Kanno [47]. Specifically, the lengths of horizontal and vertical members were, respectively, 100 cm and 50 cm, the lower and upper bounds on cross-sectional areas were, respectively, $a_{\min} := 10\,\mathrm{cm}^2$ and $a_{\max} := 1000\,\mathrm{cm}^2$, and the elastic modulus was 200 GPa. The leftmost nodes of trusses were fixed, and the bottom-right node was loaded with a vertical force of 1 kN as a nominal external load $\tilde{\boldsymbol{f}}$. The level of uncertainty was $r := 0.1$, and the upper-bound parameter for compliance was $\bar{\tau} := 10\,\mathrm{kN}$. Note that all lengths were measured in centimeters, and that the value of each member length, ℓ_i, was divided by 1000 to reduce numerical error.

Table 4 gives the computational results for the robust truss topology design problem. For instance, Lat(3,3) is a problem instance corresponding to 3×3 lattice points. Also, 22mem and 51mem stand for the 22- and 51-member truss instances defined in Yonekura and Kanno [47]. The column labeled "Obj" shows the value of the objective function (28). If the computation reached the time limit of 7200 s, the obtained lower and upper bounds on the objective function are shown.

We can see from Table 4 that there is no clear inferior-to-superior relationship between B&C and SCIP for small-sized instances. For example, B&C was three times faster than SCIP for the Lat(3,4) instance, whereas B&C was four times slower than SCIP for the 22mem instances. For large-sized instances, the computation time of B&C was relatively long. Indeed, B&C took much longer to solve the Lat(4,4) and 51mem instances than SCIP did. However, it is noteworthy that B&C successfully found solutions of good quality to such large-sized instances. For the Lat(4,4) instance, although the B&C computation was terminated due to the time limit, its computed upper-bound value (i.e., the objective value of the obtained feasible solution) was 71.01, which is very close to the optimal value (i.e., 69.34) obtained by SCIP. For the Lat(5,3) instance, B&C computed an upper-bound value of 294.49, which is very close to the optimal value (i.e., 292.68) provided by CPA and CUTSDP, whereas SCIP failed to solve this instance due to a memory shortage. CPA and CUTSDP were comparable in computational performance for

Table 4 Results for robust truss topology design

Instance	ν	μ	n	m_{b}	m_{c}	Method	Time	Obj	#Cuts	#Nodes
Lat(3,3)	12	26	24	26	38	SCIP	18.6	32.40	—	71
						CUTSDP	35.2	32.40	497	—
						CPA	55.1	32.40	347	—
						B&C	24.4	32.40	2612	17115
Lat(3,4)	16	46	32	46	65	SCIP	45.2	21.24	—	101
						CUTSDP	1590.3	21.24	764	—
						CPA	943.1	21.24	479	—
						B&C	12.9	21.24	1111	14594
Lat(4,3)	18	47	36	47	96	SCIP	326.1	114.07	—	755
						CUTSDP	629.8	114.07	1384	—
						CPA	741.3	114.07	982	—
						B&C	324.8	114.07	8855	85147
Lat(3,5)	20	70	40	70	90	SCIP	593.0	17.32	—	354
						CUTSDP	>7200	[15.92, —]	>962	—
						CPA	>7200	[16.14, —]	>433	—
						B&C	424.8	17.32	3902	149298
Lat(4,4)	24	83	48	83	107	SCIP	850.8	69.34	—	525
						CUTSDP	>7200	[64.82, —]	>1690	—
						CPA	>7200	[64.61, —]	>996	—
						B&C	>7200	[69.31, 71.01]	>10059	>423279
Lat(5,3)	24	72	48	72	96	SCIP	Abort	—	—	—
						CUTSDP	6852.7	292.68	2963	—
						CPA	6324.3	292.68	2127	—
						B&C	>7200	[292.67, 294.49]	>22398	>336055
22mem	12	22	24	22	34	SCIP	7.7	33.27	—	23
						CUTSDP	24.1	33.27	467	—
						CPA	39.2	33.27	321	—
						B&C	36.3	33.27	7507	15343
51mem	18	51	36	51	69	SCIP	336.5	75.05	—	236
						CUTSDP	>7200	[74.95, —]	>1999	—
						CPA	>7200	[74.86, —]	>1375	—
						B&C	4308.7	75.05	17680	267973

robust truss topology design, but they required a longer time of computation than both SCIP and B&C did for many problem instances.

We checked the log files of Gurobi to explain the reason why B&C did not provide superior performance for robust truss topology design. In the case of robust truss topology design, it took a relatively long time to find an initial feasible solution to the original MISDO problem, and at the same time, the gap between the lower and upper bounds on the objective function was very large. Consequently, we can see that it was very difficult in an early stage of our B&C algorithm to find a feasible solution of good quality to the problems of robust truss topology design; this is part of the reason why the performance of our B&C algorithm was decreased.

6 Conclusion

It is well known that various computational and decision-making problems can be posed as MISDO problems. In this paper we have described a high-performance

computational framework to solve such problems. Firstly, we formulated a cutting-plane algorithm to solve MISDO problems and proved its convergence properties. We then developed a B&C algorithm, where valid inequalities are added dynamically to the relaxed MILO problem during a B&B procedure. This approach reduced computation times. The experimental results confirmed that our B&C algorithm was effective in solving three types of MISDO problem: random instances, computing restricted isometry constants, and robust truss topology design. Our algorithm is based on the B&B procedure for solving MILO problems, so it can use a warm-starting strategy to solve a series of continuous relaxation problems efficiently. Moreover, no SDO problems are handled in our algorithm, which makes the MISDO computation very stable.

A future direction of study is to extend our algorithm so that it can handle a nonlinear objective function [23] and nonlinear constraints [21]. Additionally, the strength of the valid inequalities will be increased if we use the outer approximation algorithm [13, 14, 32] or the supporting hyperplane method [23]. We will also look for new applications that can be formulated as MISDO problems and try to solve them using our algorithm.

Acknowledgments

The authors would like to thank Mirai Tanaka for valuable comments on MISDO formulations.

References

1. Achterberg, T.: SCIP: solving constraint integer programs. Mathematical Programming Computation, 1(1), 1–41 (2009).
2. Aloise, D., Hansen, P.: A branch-and-cut SDP-based algorithm for minimum sum-of-squares clustering. Pesquisa Operacional, 29(3), 503–516 (2009).
3. Anjos, M. F., Ghaddar, B., Hupp, L., Liers, F., Wiegele, A.: Solving k-way graph partitioning problems to optimality: The impact of semidefinite relaxations and the bundle method. In: Jünger M., Reinelt G. (eds) Facets of Combinatorial Optimization, pp. 355–386. Springer, Berlin, Heidelberg (2013).
4. Armbruster, M., Fügenschuh, M., Helmberg, C., Martin, A.: LP and SDP branch-and-cut algorithms for the minimum graph bisection problem: a computational comparison. Mathematical Programming Computation, 4(3), 275–306 (2012).
5. Baraniuk, R. G.: Compressive sensing [lecture notes]. IEEE Signal Processing Magazine, 24(4), 118–121 (2007).
6. Benson, S. J., Ye, Y., Zhang, X.: Solving large-scale sparse semidefinite programs for combinatorial optimization. SIAM Journal on Optimization, 10(2), 443–461 (2000).
7. Ben-Tal, A., Nemirovski, A.: Robust truss topology design via semidefinite programming. SIAM Journal on Optimization, 7(4), 991–1016 (1997).
8. Bertsimas, D., Dunning, I., Lubin, M.: Reformulation versus cutting-planes for robust optimization. Computational Management Science, 13(2), 195–217 (2016).
9. Bertsimas, D., King, A.: Logistic regression: From art to science. Statistical Science, 32(3), 367–384 (2017).
10. Braun, G., Fiorini, S., Pokutta, S., Steurer, D.: Approximation limits of linear programs (beyond hierarchies). Mathematics of Operations Research, 40(3), 756–772 (2015).
11. Cerveira, A., Agra, A., Bastos, F., Gromicho, J.: A new Branch and Bound method for a discrete truss topology design problem. Computational Optimization and Applications, 54(1), 163–187 (2013).
12. Czyzyk, J., Mesnier, M. P., Moré, J. J.: The NEOS server. IEEE Computational Science and Engineering, 5(3), 68–75 (1998).

13. Duran, M. A., Grossmann, I. E.: An outer-approximation algorithm for a class of mixed-integer nonlinear programs. Mathematical programming, 36(3), 307–339 (1986).
14. Fletcher, R., Leyffer, S.: Solving mixed integer nonlinear programs by outer approximation. Mathematical programming, 66(1–3), 327–349 (1994).
15. Foucart, S., Lai, M. J.: Sparsest solutions of underdetermined linear systems via ℓ_q-minimization for $0 < q \leq 1$. Applied and Computational Harmonic Analysis, 26(3), 395–407 (2009).
16. Gally, T., Pfetsch, M. E.: Computing restricted isometry constants via mixed-integer semidefinite programming. Optimization Online, `http://www.optimization-online.org/DB_HTML/2016/04/5395.html` (2016).
17. Gally, T., Pfetsch, M. E., Ulbrich, S.: A framework for solving mixed-integer semidefinite programs. Optimization Methods and Software, 33(3), 594–632 (2018).
18. Gleixner, A., Eifler, L., Gally, T., Gamrath, G., Gemander, P., Gottwald, R. L., ... Müller, B.: The SCIP optimization suite 5.0. Optimization Online, `http://www.optimization-online.org/DB_HTML/2017/12/6385.html` (2017).
19. Horn, R. A., Johnson, C. R.: Matrix Analysis (2nd Edition). Cambridge university press (2013).
20. Joshi, S., Boyd, S.: Sensor selection via convex optimization. IEEE Transactions on Signal Processing, 57(2), 451–462 (2009).
21. Kelley, Jr, J. E.: The cutting-plane method for solving convex programs. Journal of the society for Industrial and Applied Mathematics, 8(4), 703–712 (1960).
22. Konno, H., Gotoh, J., Uno, T., Yuki, A.: A cutting plane algorithm for semi-definite programming problems with applications to failure discriminant analysis. Journal of Computational and Applied Mathematics, 146(1), 141–154 (2002).
23. Konno, H., Kawadai, N., Tuy, H.: Cutting plane algorithms for nonlinear semi-definite programming problems with applications. Journal of Global Optimization, 25(2), 141–155 (2003).
24. Krishnan, K., Mitchell, J. E.: A unifying framework for several cutting plane methods for semidefinite programming. Optimization methods and software, 21(1), 57–74 (2006).
25. Löfberg, J.: YALMIP: A toolbox for modeling and optimization in MATLAB. In: Proceedings of the 2004 IEEE International Symposium on Computer Aided Control Systems Design, pp. 284–289 (2004).
26. Lubin, M., Yamangil, E., Bent, R., Vielma, J. P.: Polyhedral approximation in mixed-integer convex optimization. Mathematical Programming, 1–30 (2016).
27. Manousakis, N. M., Korres, G. N.: Semidefinite programming for optimal placement of PMUs with channel limits considering pre-existing SCADA and PMU measurements. In: Proceedings of the 2016 Power Systems Computation Conference, pp. 1–7 (2016).
28. Mittelmann, H. D.: An independent benchmarking of SDP and SOCP solvers. Mathematical Programming, 95(2), 407–430 (2003).
29. Noyan, N., Balcik, B., Atakan, S.: A stochastic optimization model for designing last mile relief networks. Transportation Science, 50(3), 1092–1113 (2015).
30. Peng, J., Xia, Y.: A new theoretical framework for k-means-type clustering. In: Chu W., Young Lin T. (eds) Foundations and advances in data mining, pp. 79–96. Springer, Berlin, Heidelberg (2005).
31. Philipp, A., Ulbrich, S., Cheng, Y., Pesavento, M.: Multiuser downlink beamforming with interference cancellation using a SDP-based branch-and-bound algorithm. In: Proceedings of the 2014 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 7724–7728 (2014).
32. Quesada, I., Grossmann, I. E.: An LP/NLP based branch and bound algorithm for convex MINLP optimization problems. Computers & Chemical Engineering, 16(10–11), 937–947 (1992).
33. Rendl, F.: Semidefinite relaxations for integer programming. In: Jünger M. et al. (eds) 50 Years of Integer Programming 1958–2008, pp. 687–726. Springer, Berlin, Heidelberg (2010).
34. Rendl, F., Rinaldi, G., Wiegele, A.: Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations. Mathematical Programming, 121(2), 307–335 (2010).
35. Rowe, C., Maciejowski, J.: An efficient algorithm for mixed integer semidefinite optimisation. In: Proceedings of the 2003 American Control Conference, Vol. 6, pp. 4730–4735 (2003).
36. Sotirov, R.: SDP relaxations for some combinatorial optimization problems. In: Anjos M., Lasserre J. (eds) Handbook on Semidefinite, Conic and Polynomial Optimization, pp. 795–819. Springer, Boston, MA (2012).

37. Tamura, R., Kobayashi, K., Takano, Y., Miyashiro, R., Nakata, K., Matsui, T.: Best subset selection for eliminating multicollinearity. Journal of the Operations Research Society of Japan, 60(3), 321–336 (2017).
38. Taylor, J. A., Luangsomboon, N., Fooladivanda, D.: Allocating sensors and actuators via optimal estimation and control. IEEE Transactions on Control Systems Technology, 25(3), 1060–1067 (2017).
39. Todd, M. J.: Semidefinite optimization. Acta Numerica, 10, 515–560 (2001).
40. Torchio, M., Magni, L., Raimondo, D. M.: A mixed integer SDP approach for the optimal placement of energy storage devices in power grids with renewable penetration. In: Proceedings of the American Control Conference, pp. 3892–3897 (2015).
41. Tóth, S. F., McDill, M. E., Könnyü, N., George, S.: Testing the use of lazy constraints in solving area-based adjacency formulations of harvest scheduling models. Forest Science, 59(2), 157–176 (2013).
42. Vandenberghe, L., Boyd, S.: Semidefinite programming. SIAM Review, 38(1), 49–95 (1996).
43. Williams, H. P.: Model building in mathematical programming. John Wiley Sons (2013).
44. Westerlund, T., Pettersson, F.: An extended cutting plane method for solving convex MINLP problems. Computers & Chemical Engineering, 19, 131–136 (1995).
45. Yamashita, M., Fujisawa, K., Kojima, M.: Implementation and evaluation of SDPA 6.0 (semidefinite programming algorithm 6.0). Optimization Methods and Software, 18(4), 491–505 (2003).
46. Yokoyama, R., Shinano, Y., Taniguchi, S., Ohkura, M., Wakui, T.: Optimization of energy supply systems by MILP branch and bound method in consideration of hierarchical relationship between design and operation. Energy Conversion and Management, 92, 92–104 (2015).
47. Yonekura, K., Kanno, Y.: Global optimization of robust truss topology via mixed integer semidefinite programming. Optimization and Engineering, 11(3), 355–379 (2010).
48. Zhang, Y., Shen, S., Erdogan, S. A.: Solving 0-1 Semidefinite Programs for Distributionally Robust Allocation of Surgery Blocks. Optimization Letter (in press).