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

A Combinatorial Cut-and-Lift Procedure with an Application to 0-1 Second-Order Conic Programming

Margarita P. Castro · Andre A. Cire · J. Christopher Beck

Received: date / Accepted: date

Abstract Cut generation and lifting are key components for the performance of state-of-the-art mathematical programming solvers. This work proposes a new general cut-and-lift procedure that exploits the combinatorial structure of 0-1 problems via a binary decision diagram (BDD) encoding of their constraints. We present a general framework that can be applied to a wide range of binary optimization problems and show its applicability for second-order conic inequalities. We identify conditions for which our lifted inequalities are facet-defining and derive a new BDD-based cut generation linear program. Such a model serves as a basis for a max-flow combinatorial algorithm over the BDD that can be applied to derive valid cuts more efficiently. Our numerical results show encouraging performance when incorporated into a state-of-the-art mathematical programming solver, significantly reducing the root node gap, increasing the number of problems solved, and reducing the run-time by a factor of three on average.

Keywords Lifting · Cutting Planes · Decision Diagrams · Binary Optimization · Second-order Cones

Margarita P. Castro
Department of Industrial and Systems Engineering, Pontificia Universidad Católica de Chile
E-mail: margarita.castro@ing.puc.cl

Andre A. Cire
Department of Management, University of Toronto Scarborough and
Rotman School of Management
E-mail: andre.cire@rotman.utoronto.ca

J. Christopher Beck
Department of Mechanical and Industrial Engineering, University of Toronto,
E-mail: jcb@mie.utoronto.ca

1 Introduction

Cutting plane methodologies have played a key role in the theoretical and computational development of mathematical programming [19,46]. Extensive literature has focused on cuts that exploit special problem substructure, leading to an array of techniques that are now integral into state-of-the-art solvers [41]. For general problems, cuts are obtained either by leveraging disjunctive reformulations [9,10] or by *lifting*, i.e., relaxing an initial inequality so that it is valid for a higher-dimensional polyhedron [28,43,58].

In this paper, we study both a cut generation procedure and a lifting approach for general binary optimization problems of the form

$$\max_{\boldsymbol{x} \in X \subseteq \{0,1\}^n} \boldsymbol{c}^\top \boldsymbol{x}, \tag{BP}$$

where the feasible set X is arbitrary, e.g., possibly represented by a conjunction of linear and/or non-linear constraints. Our methodologies consist of exploiting *network structure* via a binary decision diagram (BDD) embedding of X. A BDD is a graphical model that represents solutions as paths in a directed acyclic graph, which can be viewed as a network-flow reformulation of X. Such a model is potentially orders of magnitude smaller than an explicit representation of X as it identifies and merges equivalent partial solutions. Several BDD encodings have already been investigated for linear and non-linear problems [12,13,44] and are used to exploit submodularity [14] or more general combinatorial structure [16].

We propose a sequential lifting procedure that can be applied to any initial inequality (e.g., given by another cutting-plane technique). The lifting algorithm uses 0-1 disjunctions derived from a BDD representation of X to rotate inequalities while maintaining their validity. We show that each step of our sequential lifting can be performed efficiently in the size of the BDD and, when applicable, increases the dimension of the face by at least one. We also establish conditions for which the inequality becomes facet-defining and draw connections between our procedure and existing lifting techniques from disjunctive programming [8], showing that our approach generalizes well-known lifting procedures for 0-1 inequalities [7,32,50].

For our cut generation approach, we propose a reformulation of the BDD polytope based on capacitated flows, which leads to an alternative cut generation linear program (CGLP) for separating infeasible points. We show that the set of cuts derived from this model defines the convex hull of the solutions encoded by the BDD, i.e., X, and that this methodology circumvents common issues of existing BDD cut techniques. Finally, we build on this model to develop a weaker but computationally faster alternative that solves a combinatorial max-flow/min-cut problem over the BDD to generate valid inequalities.

For optimization problems where a BDD for X may be exponentially large in n, our lifting and cut procedures remain valid when considering instead a limited-size *relaxed* BDD for BP, i.e., where the BDD encodes a superset of X. Several efficient methods exist to build relaxed BDDs, such as only considering

a subset of the problem constraints [16]. This approach is similar in spirit, e.g., to when a linear relaxation is used to lift cover inequalities of a single knapsack constraint [7]. We exploit the discrete relaxation given by the BDD as opposed to a continuous relaxation, which captures some of the combinatorial structure of the problem. Both our lifting procedure and combinatorial cuts are also of low complexity in the size of the BDD and, when a relaxed BDD is employed, its size can be controlled through a parameter limiting its maximum width.

To assess our lifting and cut generation procedure, we apply our methodology to a class of second-order conic programming problems (SOCPs) with multiple second-order conic (SOC) inequalities, each reformulated as a BDD. We experiment on the SOC knapsack benchmark [5,36] and 270 randomly generated instances with more general and challenging SOC inequalities, incorporating our cut-and-lift approach into `CPLEX`. We also compare with existing BDD cutting techniques [55,27] and a SOC cut-and-lift method [5], also noting that `CPLEX` includes many state-of-the-art SOC techniques [35].

Our numerical results indicated that (a) our lifting procedure reduced the average root gap up to 29% in our benchmark when applied to cuts generated by all procedures; (b) when cuts are added only at the root node, a hybrid technique combining our method with BDD target cuts [55] is the most effective for SOC knapsacks, while for general SOC inequalities our cuts perform similarly to target cuts; and (c) when adding cuts during the tree search, the hybrid is also the best performing across all methods, particularly achieving the best final gaps, solution times, and number of solved instances for our general SOCP benchmark. Furthermore, the hybrid BDD technique improves upon default `CPLEX` and reduced up to 52.2% of the root node gap, closed at last 17 instances in each benchmark, and decreased solution times by threefold.

The paper is structured as follows. §2 introduces notation and background material. §3 describes related works in the BDD and lifting literature. §4 describes our combinatorial lifting procedure while §5 details our BDD-based cutting-plane algorithms. §6 introduces the case study problem and describes the BDD encoding for SOC inequalities. Lastly, §7 and §8 present the empirical evaluation and final remarks, respectively.

2 Background

This section introduces the notation used throughout this work and the background material on BDDs. For convenience, we assume $n \geq 1$ and let $I := \{1, ..., n\}$ represent the component indices of any n-dimensional point $\boldsymbol{x}$.

We denote by $\dim(P)$ the dimension of a polytope $P \subseteq [0,1]^n$. An inequality $\boldsymbol{\pi}^\top \boldsymbol{x} \leq \pi_0$ with $\boldsymbol{\pi} \in \mathbb{R}^n$ and $\pi_0 \in \mathbb{R}$ is valid for P if $\boldsymbol{\pi}^\top \boldsymbol{x} \leq \pi_0$ holds for all $\boldsymbol{x} \in P$. The inequality defines a face of P if $F(\boldsymbol{\pi}) := \{\boldsymbol{x} \in P : \boldsymbol{\pi}^\top \boldsymbol{x} = \pi_0\}$ is not empty, i.e., the inequality *supports* P. A face $F(\boldsymbol{\pi})$ is a *facet* if $\dim(F(\boldsymbol{\pi})) = \dim(P) - 1$; in such a case, $\boldsymbol{\pi}^\top \boldsymbol{x} \leq \pi_0$ is *facet-defining*. Finally, we denote the convex hull of P by $\text{conv}(P)$.

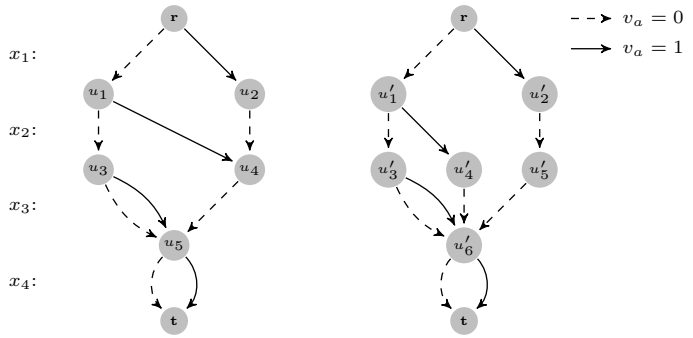

Fig. 1 Two BDDs $\mathcal{B}_1$ (left-hand side) and $\mathcal{B}_2$ (right-hand side) with $X_{\mathcal{B}_1} = X_{\mathcal{B}_2} = \{\boldsymbol{x} \in \{0,1\}^4 : 7x_1 + 5x_2 + 4x_3 + x_4 \leq 8\}$. $\mathcal{B}_1$ is reduced.

Binary Decision Diagrams. A BDD $\mathcal{B}$ is an extended representation of a set $X_{\mathcal{B}} \subseteq \{0,1\}^n$ as a network. Specifically, $\mathcal{B} = (\mathcal{N}, \mathcal{A})$ is a layered directed acyclic graph with node set $\mathcal{N}$ and arc set $\mathcal{A}$. The node set $\mathcal{N}$ is partitioned into $n+1$ *layers* $\mathcal{N} = (\mathcal{N}_1, ..., \mathcal{N}_{n+1})$. The first and last layers are the singletons $\mathcal{N}_1 = \{\mathbf{r}\}$ and $\mathcal{N}_{n+1} = \{\mathbf{t}\}$, respectively, where $\mathbf{r}$ is the root node and $\mathbf{t}$ is the terminal node. An arc $a = (u, u') \in \mathcal{A}$ has a source node $s(a) = u$ and a target node $t(a) = u'$ in consecutive layers, i.e., $u' \in \mathcal{N}_{i+1}$ whenever $u \in \mathcal{N}_i$ for $i \in I$.

The points of $X_{\mathcal{B}}$ are mapped to paths in the network, as follows. With each arc $a \in \mathcal{A}$ we associate a value $v_a \in \{0,1\}$, where a node $u \in \mathcal{N}$ has at most one arc of each value emanating from it. Given an arc-specified $\mathbf{r}-\mathbf{t}$ path $p = (a_1, ..., a_n)$ with $s(a_1) = \mathbf{r}$ and $t(a_n) = \mathbf{t}$, we let $\boldsymbol{x}^p := (v_{a_1}, v_{a_2}, \ldots, v_{a_n}) \in \{0,1\}^n$ be the n-dimensional point encoded by path p. Thus, if $\mathcal{P}$ is the set of all $\mathbf{r}-\mathbf{t}$ paths in $\mathcal{B}$, the set of points represented by the BDD is $X_{\mathcal{B}} = \bigcup_{p \in \mathcal{P}} \{\boldsymbol{x}^p\}$.

A BDD $\mathcal{B}$ is *exact* for set $X \subseteq \{0,1\}^n$ when $X = X_{\mathcal{B}}$, i.e., there is a one-to-one relationship between the points in X and the $\mathbf{r} - \mathbf{t}$ paths in $\mathcal{B}$. Alternatively, $\mathcal{B}$ is *relaxed* when $X \subseteq X_{\mathcal{B}}$, i.e., every point in X maps to a path in $\mathcal{B}$ but the converse is not necessarily true.

Example 1 Consider $X = \{\boldsymbol{x} \in \{0,1\}^4 : 7x_1 + 5x_2 + 4x_3 + x_4 \leq 8\}$. Figure 1 illustrates two exact BDDs for X: $\mathcal{B}_1$ on the left-hand side and $\mathcal{B}_2$ on the right-hand side. Dashed and solid arcs have a value of 0 and 1, respectively. Each point $\boldsymbol{x} \in X$ is represented by a path in $\mathcal{B}_1$ and $\mathcal{B}_2$. For example, $\boldsymbol{x} = (1,0,0,1) \in X$ is encoded by the path $((\mathbf{r}, u_2), (u_2, u_4), (u_4, u_5), (u_5, \mathbf{t}))$ in $\mathcal{B}_1$, and by the path $((\mathbf{r}, u'_2), (u'_2, u'_5), (u'_5, u'_6), (u'_6, \mathbf{t}))$ in $\mathcal{B}_2$. □

A BDD $\mathcal{B}$ is *reduced* if it is the smallest network (with respect to number of nodes) that represents the set $X_{\mathcal{B}}$. There exists a unique reduced BDD for a given ordering of the indices I. Furthermore, given any $\mathcal{B}$ and an ordering, we can obtain its associated reduced BDD in polynomial time in the size of $\mathcal{B}$ [22]. For instance, $\mathcal{B}_1$ in Figure 1 is reduced and can be obtained by merging nodes u'_4 and u'_5 from $\mathcal{B}_2$ and adjusting their emanating arcs appropriately.

Several exact and relaxed BDD construction mechanisms are available for general and specialized discrete optimization problems [16,14,55]. These tech-

niques either reformulate the problem as a dynamic program, where $\mathcal{B}$ represents an underlying state-transition graph, or separate infeasible paths of a relaxed BDD. It is often the case that BDDs can be exponentially smaller than enumerating $X_{\mathcal{B}}$ explicitly [16]. If exact BDDs are too large, relaxed BDDs can be built for X by either imposing a limit on the number of nodes or by considering a subset of the constraints. We discuss the construction and relaxation techniques used for our case study in §6.

3 Related Work

Recent research has shown the versatility of BDDs for modeling linear and non-linear inequalities [2,33,17,14] and there is a growing literature on BDD encodings for vehicle routing [23,51,39], scheduling [25,20,21,34], and other combinatorial optimization problems [13,15,24,16]. Within the context of this work, Becker et al. (2005) [11] presented the first BDD cut generation procedure based on an iterative subgradient algorithm that relies on a longest-path problem over the BDD. Behle (2007) [12] formalized this procedure and proposed a branch-and-cut algorithm that employs BDDs to generate exclusion and implication cuts. The author also introduced the network flow model employed by most BDD cutting-plane procedures [27,44,55].

More recently, two BDD-based cutting plane techniques have been proposed with theoretical and computational considerations. Tjandraatmadja and van Hoeve (2019) [55] generate target cuts from polar sets using relaxed BDDs to derive a more tractable methodology that in Becker et al. (2005) [11]. Davarnia and van Hoeve (2020) [27] develop an iterative method to generate outer-approximations for non-linear inequalities, relying on a subgradient algorithm to avoid solving linear programs. We compare these two models conceptually to our approach in §5.4 and evaluate them numerically in §7.

Our cutting plane methods are related to the method by Lozano and Smith (2018) [44] for a class of two-stage stochastic programming problems. The authors propose a Benders decomposition approach using BDDs to encode second-stage decisions, where arcs are can be activated or deactivated based on first-stage decisions. Benders cuts are obtained by solving a network-flow model over the BDD where arcs are bounded by the first-stage variables. Similarly, our CGLP model proposed in §5 is also based on a capacitated network-flow model. However, our model is more structured in that it incorporates one variable per layer for all arc types, leading to more specialized inequalities and structural results for separation purposes (e.g., Lemma 3 and Theorems 3 and 4). We also leverage this model to develop our combinatorial max-flow cuts that do not depend on linear programming (LP) solutions.

Our numerical case study is focused on SOCPs. Recent work in the field relies on ideas from split cuts [45], disjunctions [38,42], or are more specialized [37,52]. Techniques based on mixed-integer rounding [6] and lift-and-project [54] have also shown to be suitable in practice, and are currently implemented

in commercial solvers [35]. Several recent works also exploit SOCPs with special structure, such as binary SOC knapsack inequalities [5,18,4,36].

While the literature on BDD cutting-plane procedures has grown recently, to the best of our knowledge, this is the first work that leverages BDDs to lift general form linear inequalities. Behle (2007) [12] proposes lifting cover inequalities using classic techniques that compute new coefficients one at a time [59] and where each sub-problem is solved using a BDD. Becker et al. (2005) [11] also present a mechanism that uses 0-1 disjunctions over a BDD to obtain new inequalities. Their technique differs from ours with respect to both the procedure to obtain the new inequality and its theoretical guarantees. In particular, their lifted inequality might not separate fractional points that the original inequality does nor induce a face with higher dimension.

Our combinatorial lifting relates to sequential lifting algorithms based on 0-1 disjunctions [8], including specialized procedures for the knapsack polytope [7,48,47] and submodular inequalities [32,5]. These procedures successfully address special cases of the general lifting problem we investigate (see §4), focusing on given problem structures (e.g., monotone sets). In particular, lifting cover inequalities is a well-studied area [29,30], often using the classical knapsack dynamic program to efficiently lift coefficients [60] when such constraints are present. In contrast, our approach is general in that it can be applied to any type of valid linear inequality (i.e., not restricted to cover inequalities) or feasibility set, including those defined by non-linear constraints. We also note that BDD sizes can be parameterized for large-scale problems.

Lastly, our methodology is closely related to the n-step lifting procedure by Perregaard and Balas (2001) [50], which generalizes the special cases mentioned above (e.g., lifting cover inequalities). We briefly introduce this procedure below and relate it to our combinatorial lifting algorithm in §4.4.

An Iterative Lifting Procedure based on Disjunctive Programming. Given a mixed-integer linear programming (MILP) problem of the form $\max_{\boldsymbol{x}}\{\boldsymbol{c}^\top \boldsymbol{x}: A\boldsymbol{x} \leq \boldsymbol{b}, x_i \in \mathbb{Z}, \ \ \forall i \in I' \subseteq I\}$, the authors [50] propose the relaxation

$$\max_{\boldsymbol{x}} \left\{ \boldsymbol{c}^\top \boldsymbol{x} : \ A\boldsymbol{x} \leq \boldsymbol{b}, \ \bigvee_{k \in K} D^k \boldsymbol{x} \leq \boldsymbol{d}^k, x_i \in \mathbb{Z} \quad \forall i \in I'' \subset I' \right\}, \tag{DP}$$

where fewer variables are constrained to be integral. The set K that defines the disjunctive constraints is typically derived by considering the 0-1 integrality constraints of individual variables (e.g., $x_i \leq 0 \vee x_i \geq 1$).

Let P_{DP} be the set of solutions of DP. The n-step procedure considers two inputs: (a) an inequality $\boldsymbol{\pi}^\top \boldsymbol{x} \leq \pi_0$ that supports $\text{conv}(P_{DP})$; and (b) an arbitrary target inequality $\widetilde{\boldsymbol{\pi}}^\top \boldsymbol{x} \leq \widetilde{\pi}_0$ that is tight for all integer points in $F(\boldsymbol{\pi})$. The procedure uses a parameter γ to rotate the supporting inequality towards the target inequality, generating a new lifted inequality $(\boldsymbol{\pi}+\gamma\widetilde{\boldsymbol{\pi}})^\top \boldsymbol{x} \leq \pi_0+\gamma\widetilde{\pi}_0$ that is valid for $\text{conv}(P_{DP})$. In particular, if $\widetilde{\boldsymbol{\pi}}^\top \boldsymbol{x} \leq \widetilde{\pi}_0$ is *not* valid for P_{DP}, it

can be shown that there is a finite maximal γ given by the disjunctive program

$$\gamma^* = \min_{\boldsymbol{x}, x_0} \Big\{ \pi_0 x_0 - \boldsymbol{\pi}^\top \boldsymbol{x} : \;\; A\boldsymbol{x} - \boldsymbol{b}x_0 \leq 0, \;\; \bigvee_{k \in K} D^k \boldsymbol{x} - \boldsymbol{d}^k x_0 \leq 0, \\ \widetilde{\pi}_0 x_0 - \widetilde{\boldsymbol{\pi}}^\top \boldsymbol{x} = -1, \;\; x_0 \geq 0, \;\; x_i \in \mathbb{Z} \;\; \forall i \in I'' \subset I' \Big\}.$$

Under the same assumptions, the lifted inequality becomes a facet of $\text{conv}(P_{DP})$ if the procedure is repeated n times, using the rotated inequality and an appropriate target inequality.

Similarly, our approach is a sequential procedure that relies on disjunctions. It differs from the above method in that we exploit the combinatorial structure encoded by a BDD as opposed to a disjunctive program relaxation. Such a BDD may encode, e.g., complex non-linear constraints that are not necessarily convex [14]. Furthermore, we also exploit the network to derive a tractable and efficient way to compute several disjunctions simultaneously, while previous algorithms are typically restricted to a small number of disjunctions [50].

4 Combinatorial Lifting

We now present our combinatorial lifting procedure and develop its structural properties. We begin by introducing our basic methodology in §4.1, which is defined in general terms and does not depend on a BDD $\mathcal{B}$ encoding. Next, in §4.2 we present a methodology that exploits network structure to perform the proposed lifting in polynomial time in the size of $\mathcal{B}$ (i.e., in the number of nodes and arcs). Next, §4.3 incorporates the technique in a sequential procedure and investigate the dimension of the resulting face. Finally, we depict the relationship with previous disjunctive methodologies in §4.4.

Throughout this section, we assume that, for a given $X \subseteq \{0,1\}^n$, (a) inequality $\boldsymbol{\pi}^\top \boldsymbol{x} \leq \pi_0$ is valid and supports $\text{conv}(X)$; (b) $\mathcal{B}$ is an exact BDD for X, i.e., $X_{\mathcal{B}} = X$; and (c) for any $i \in I$, there exists $\boldsymbol{x}, \boldsymbol{x}' \in X$ such that $x_i = 0$ and $x'_i = 1$. Assumption (a) is a common lifting condition that is satisfied by setting $\pi_0 := \max_{\boldsymbol{x} \in X} \{\boldsymbol{\pi}^\top \boldsymbol{x}\}$. This, in turn, can be enforced in linear time in the size of $\mathcal{B}$ (see §4.2). Assumption (b) is needed for our theoretical results but it can be relaxed in practice (see §7). For (c), we can soundly remove any i-th component not satisfying the assumption, adjusting n accordingly.

Our goal is to lift $\boldsymbol{\pi}^\top \boldsymbol{x} \leq \pi_0$ and better represent $\text{conv}(X)$ by exploiting the network structure of $\mathcal{B}$. The resulting cuts are valid for any subset $X' \subseteq X$; e.g., when $\mathcal{B}$ (and hence X) is a relaxation of some feasible set.

4.1 Disjunctive Slack Lifting

The core element of our lifting procedure is what we denote by *disjunctive slack vector* (or *d-slack* in short). The i-th component of the d-slack indicates the change in the maximum values of the left-hand side of $\boldsymbol{\pi}^\top \boldsymbol{x} \leq \pi_0$ when varying x_i. This is formalized in Definition 1.

Definition 1 The disjunctive slack vector $\boldsymbol{\lambda}(\boldsymbol{\pi})$ with respect to $\boldsymbol{\pi}$ is given by

$$\lambda_i(\boldsymbol{\pi}) := \lambda_i^0(\boldsymbol{\pi}) - \lambda_i^1(\boldsymbol{\pi}), \quad \forall i \in I,$$

with $\lambda_i^0(\boldsymbol{\pi}) := \max_{\boldsymbol{x} \in X}\{\boldsymbol{\pi}^\top \boldsymbol{x} : x_i = 0\}$ and $\lambda_i^1(\boldsymbol{\pi}) := \max_{\boldsymbol{x} \in X}\{\boldsymbol{\pi}^\top \boldsymbol{x} : x_i = 1\}$.

For notational convenience, we let $S^-(\boldsymbol{\pi}) := \{i \in I : \lambda_i(\boldsymbol{\pi}) < 0\}$, $S^0(\boldsymbol{\pi}) := \{i \in I : \lambda_i(\boldsymbol{\pi}) = 0\}$, and $S^+(\boldsymbol{\pi}) := \{i \in I : \lambda_i(\boldsymbol{\pi}) > 0\}$ be a partition of I with respect to negative, zero, and positive d-slacks, respectively. Lemma 1 presents key properties of d-slacks used for our main results.

Lemma 1 *For any $\boldsymbol{\lambda}(\boldsymbol{\pi})$ and index $i \in I$,*

(1) $i \in S^-(\boldsymbol{\pi})$ if and only if $x_i = 1$ for all $\boldsymbol{x} \in F(\boldsymbol{\pi})$.
(2) $i \in S^+(\boldsymbol{\pi})$ if and only if $x_i = 0$ for all $\boldsymbol{x} \in F(\boldsymbol{\pi})$.
(3) $i \in S^0(\boldsymbol{\pi})$ if and only if there exists $\boldsymbol{x}, \boldsymbol{x}' \in F(\boldsymbol{\pi})$ with $x_i = 0$ and $x'_i = 1$.

Proof For the necessary conditions, consider first $x_i = 1$ for all $\boldsymbol{x} \in F(\boldsymbol{\pi})$. Since the solutions when optimizing over $\boldsymbol{\pi}^\top \boldsymbol{x}$ must belong to the face $F(\boldsymbol{\pi})$, we must necessarily have $\lambda_i^1(\boldsymbol{\pi}) > \lambda_i^0(\boldsymbol{\pi})$ and so the d-slack $\lambda_i(\boldsymbol{\pi})$ is negative. An analogous reasoning holds for the other two cases.

For the sufficient conditions, consider first $i \in S^-(\boldsymbol{\pi})$. Then $\lambda_i^1(\boldsymbol{\pi}) = \pi_0$ and $\lambda_i^0(\boldsymbol{\pi}) < \pi_0$, i.e., all $\boldsymbol{x} \in F(\boldsymbol{\pi})$ are such that $x_i = 1$. The same argument can be applied to the case $i \in S^+(\boldsymbol{\pi})$. Lastly, if $i \in S^0(\boldsymbol{\pi})$, $\lambda_i^0(\boldsymbol{\pi}) = \lambda_i^1(\boldsymbol{\pi}) = \pi_0$. Thus, there exists $\boldsymbol{x} \in F(\boldsymbol{\pi})$ that maximizes $\lambda_i^1(\boldsymbol{\pi})$ (i.e., $x_i = 1$) and $\boldsymbol{x}' \in F(\boldsymbol{\pi})$ that maximizes $\lambda_i^0(\boldsymbol{\pi})$ (i.e., $x'_i = 0$). ■

We now show in Theorem 1 how to apply the d-slacks to lift $\boldsymbol{\pi}^\top \boldsymbol{x} \leq \pi_0$. In particular, the resulting inequality is valid for X (and thereby $\text{conv}(X)$), the dimension of the face necessarily increases, and points separated by the original inequality are still separated after lifting. This last characteristic is important, e.g., if the input inequality $\boldsymbol{\pi}^\top \boldsymbol{x} \leq \pi_0$ was derived to separate a fractional point. Note that we require a d-slack with a non-zero component to rotate the inequality, as we later illustrate in Example 2.

Theorem 1 *Suppose $\lambda_i(\boldsymbol{\pi}) \neq 0$ for some $i \in I$. Let $\langle \boldsymbol{\pi}', \pi'_0 \rangle$ be such that*

$$\pi'_j := \begin{cases} \pi_j & \text{if } j \neq i, \\ \pi_j + \lambda_j(\boldsymbol{\pi}) & \text{otherwise,} \end{cases} \quad \forall j \in I, \quad \pi'_0 := \begin{cases} \pi_0 & \text{if } i \in S^+(\boldsymbol{\pi}), \\ \pi_0 + \lambda_i(\boldsymbol{\pi}) & \text{otherwise.} \end{cases}$$

The following properties hold:

(1) $\boldsymbol{\pi}'^\top \boldsymbol{x} \leq \pi'_0$ is valid for X.
(2) $F(\boldsymbol{\pi}) \subset F(\boldsymbol{\pi}')$ and $\dim(F(\boldsymbol{\pi}')) \geq \dim(F(\boldsymbol{\pi})) + 1$.
(3) For any $\bar{\boldsymbol{x}} \in [0,1]^n$ with $\boldsymbol{\pi}^\top \bar{\boldsymbol{x}} > \pi_0$, we have that $\boldsymbol{\pi}'^\top \bar{\boldsymbol{x}} > \pi'_0$.

Proof Let $\boldsymbol{x} \in X$. We begin by showing (1) and (2). Assume first that $i \in S^+(\boldsymbol{\pi})$. By construction, $\boldsymbol{\pi}'^\top \boldsymbol{x} \leq \pi'_0 \iff \boldsymbol{\pi}^\top \boldsymbol{x} + \lambda_i(\boldsymbol{\pi}) x_i \leq \pi_0$.

If $x_i = 0$, the lifted inequality is equivalent to the original and therefore valid. Otherwise, if $x_i = 1$, $i \in S^+(\boldsymbol{\pi})$ implies that $\lambda_i(\boldsymbol{\pi}) = \pi_0 - \lambda_i^1(\boldsymbol{\pi})$. Thus,

$$\boldsymbol{\pi}'^\top \boldsymbol{x} \leq \pi_0' \iff \boldsymbol{\pi}^\top \boldsymbol{x} + \pi_0 - \lambda_i^1(\boldsymbol{\pi}) \leq \pi_0 \iff \boldsymbol{\pi}^\top \boldsymbol{x} \leq \lambda_i^1(\boldsymbol{\pi}).$$

The last inequality above holds because we are restricting to the case $x_i = 1$ and, by definition, $\lambda_i^1(\boldsymbol{\pi}) = \max_{\boldsymbol{x}' \in X}\{\boldsymbol{\pi}^\top \boldsymbol{x}' : x_i' = 1\}$. Since $x_i' = 0$ for all $\boldsymbol{x}' \in F(\boldsymbol{\pi})$ (Lemma 1), the lifted inequality is tight for all $\boldsymbol{x}' \in F(\boldsymbol{\pi})$, i.e., $F(\boldsymbol{\pi}) \subset F(\boldsymbol{\pi}')$. Notice also that this inequality is tight for $\boldsymbol{x}^* = \arg\max_{\boldsymbol{x}' \in X}\{\boldsymbol{\pi}^\top \boldsymbol{x}' : x_i' = 1\}$, i.e., $\boldsymbol{x}^* \in F(\boldsymbol{\pi}')$. Then, $\boldsymbol{x}^*$ is affinely independent to all points of $F(\boldsymbol{\pi})$ and therefore $\dim(F(\boldsymbol{\pi}')) \geq \dim(F(\boldsymbol{\pi})) + 1$.

Assume now that $i \in S^-(\boldsymbol{\pi})$. Once again by construction, $\boldsymbol{\pi}'^\top \boldsymbol{x} \leq \pi_0'$ if and only if $\boldsymbol{\pi}^\top \boldsymbol{x} + \lambda_i(\boldsymbol{\pi}) x_i \leq \pi_0 + \lambda_i(\boldsymbol{\pi})$. If $x_i = 1$, the lifted inequality is equivalent to the original and therefore valid. Otherwise, if $x_i = 0$, $i \in S^-(\boldsymbol{\pi})$ implies that $\lambda_i(\boldsymbol{\pi}) = \lambda_i^0(\boldsymbol{\pi}) - \pi_0$. Thus,

$$\boldsymbol{\pi}'^\top \boldsymbol{x} \leq \pi_0' \iff \boldsymbol{\pi}^\top \boldsymbol{x} \leq \pi_0 + \lambda_i^0(\boldsymbol{\pi}) - \pi_0 \iff \boldsymbol{\pi}^\top \boldsymbol{x} \leq \lambda_i^0(\boldsymbol{\pi}).$$

The last inequality above holds because $x_i = 0$ and, by definition, $\lambda_i^0(\boldsymbol{\pi}) = \max_{\boldsymbol{x}' \in X}\{\boldsymbol{\pi}^\top \boldsymbol{x}' : x_i' = 0\}$. As before, notice that this inequality is tight for $\boldsymbol{x}^* = \arg\max_{\boldsymbol{x}' \in X}\{\boldsymbol{\pi}^\top \boldsymbol{x}' : x_i' = 0\}$, i.e., $\boldsymbol{x}^* \in F(\boldsymbol{\pi}')$. Since $x_i' = 1$ for all $\boldsymbol{x}' \in F(\boldsymbol{\pi})$ (Lemma 1), the inequality is tight for all $\boldsymbol{x}' \in F(\boldsymbol{\pi})$, $\boldsymbol{x}^*$ is affinely independent to all points of $F(\boldsymbol{\pi})$, and therefore $\dim(F(\boldsymbol{\pi}')) \geq \dim(F(\boldsymbol{\pi})) + 1$.

Lastly, we demonstrate (3). We restrict to the case $i \in S^+(\boldsymbol{\pi})$; the other case is analogous. Given a fractional point $\bar{\boldsymbol{x}} \in [0,1]^n$ as defined above, we have $\boldsymbol{\pi}'^\top \bar{\boldsymbol{x}} = \boldsymbol{\pi}^\top \bar{\boldsymbol{x}} + \lambda_i(\boldsymbol{\pi}) \bar{x}_i > \pi_0 + \lambda_i(\boldsymbol{\pi}) \bar{x}_i \geq \pi_0 = \pi_0'$. ■

Example 2 Let $X = \{\boldsymbol{x} \in \{0,1\}^4 : 7x_1 + 5x_2 + 4x_3 + x_4 \leq 8\}$ and consider an inequality $x_1 + x_2 \leq 1$ supporting $\text{conv}(X)$. The d-slack is $\boldsymbol{\lambda}(\boldsymbol{\pi}) = (0, 0, 1, 0)^\top$ and the lifted inequality with respect to $\lambda_3(\boldsymbol{\pi}) = 1$ is $\boldsymbol{\pi}'^\top \boldsymbol{x} = x_1 + x_2 + x_3 \leq 1$. Note that $\boldsymbol{\pi}'^\top \boldsymbol{x} \leq 1$ is facet-defining for $\text{conv}(X)$ and $\boldsymbol{\lambda}(\boldsymbol{\pi}') = \mathbf{0}$. □

4.2 Extracting Disjunctive Slacks from a BDD

Identifying d-slacks $\boldsymbol{\lambda}(\boldsymbol{\pi})$ is a non-trivial task since we are required to solve $2n$ binary optimization problems, i.e., one for each component $i \in I$ and values 0 and 1. In this section, we leverage the network representation of a BDD $\mathcal{B} = (\mathcal{N}, \mathcal{A})$ for X to generate all d-slacks simultaneously, which is key to the computational complexity of the approach. We also show that the procedure complexity is linear in the number of arcs $|\mathcal{A}|$ of $\mathcal{B}$.

We associate a *length* of $\pi_i \cdot v_a$ to each arc $a \in \mathcal{A}$ with value $v_a \in \{0,1\}$ and source $s(a) \in \mathcal{N}_i$ for some $i \in I$. The longest $\mathbf{r} - \mathbf{t}$ path of $\mathcal{B}$ with respect to such lengths maximizes $\boldsymbol{\pi}^\top \boldsymbol{x}$ over X. Given the $\mathbf{r} - \mathbf{t}$ paths $\mathcal{P}$ of $\mathcal{B}$, let

$$\ell_a := \max\left\{\sum_{k=1}^{n} \pi_k \cdot v_{a_k} : p = (a_1, \ldots, a_n) \in \mathcal{P}, a_i = a\right\}$$

be the longest-path value conditioned on all paths that include arc a. Because each variable is uniquely associated with a layer, it follows that

$$\lambda_i^j(\boldsymbol{\pi}) = \max_{a \in \mathcal{A}} \{\ell_a \colon s(a) \in \mathcal{N}_i,\ v_a = j\}, \quad \forall i \in I,\, \forall j \in \{0,1\},$$

and the final d-slacks are obtained by the differences $\lambda_i^0(\boldsymbol{\pi}) - \lambda_i^1(\boldsymbol{\pi})$ for all i.

The lengths ℓ_a are derived by performing two longest-path computations over $\mathcal{B}$. Specifically, let $\mathcal{A}^{\mathsf{in}}(u)$ and $\mathcal{A}^{\mathsf{out}}(u)$ be the set of incoming and outgoing arcs of a node $u \in \mathcal{N}$, respectively. The solution of the recursion $L^{\downarrow}(\boldsymbol{\pi}, \mathbf{r}) = 0$,

$$L^{\downarrow}(\boldsymbol{\pi}, u) = \max_{a \in \mathcal{A}^{\mathsf{in}}(u)} \{L^{\downarrow}(\boldsymbol{\pi}, s(a)) + \pi_{i-1} \cdot v_a\}, \quad \forall u \in \mathcal{N}_i, \forall i \in \{2, \dots, n+1\}$$

provides the longest-path value $L^{\downarrow}(\boldsymbol{\pi}, u)$ from $\mathbf{r}$ to u, while $L^{\uparrow}(\boldsymbol{\pi}, \mathbf{t}) = 0$,

$$L^{\uparrow}(\boldsymbol{\pi}, u) = \max_{a \in \mathcal{A}^{\mathsf{out}}(u)} \{L^{\uparrow}(\boldsymbol{\pi}, t(a)) + \pi_i \cdot v_a\}, \quad \forall u \in \mathcal{N}_i,\ \forall i \in \{1, \dots, n\},$$

provides the longest-path value $L^{\uparrow}(\boldsymbol{\pi}, u)$ from u to $\mathbf{t}$. The values $L^{\downarrow}(\boldsymbol{\pi}, u)$ can be calculated via a top-down pass on $\mathcal{B}$, i.e., starting from $\mathbf{r}$ and considering one layer $\mathcal{N}_2, \dots, \mathcal{N}_{n+1}$ at a time. Analogously, the values $L^{\uparrow}(\boldsymbol{\pi}, u)$ are obtained via a bottom-up pass on $\mathcal{B}$, i.e., starting from $\mathbf{t}$ and considering one layer $\mathcal{N}_n, \mathcal{N}_{n-1}, \dots, \mathcal{N}_1$ at a time. For any arc $a = (s(a), t(a))$ such that $s(a) \in \mathcal{N}_i$, its length is given by $\ell_a = L^{\downarrow}(\boldsymbol{\pi}, s(a)) + L^{\uparrow}(\boldsymbol{\pi}, t(a)) + \pi_i \cdot v_a$. Since each arc is traversed twice via the top-down and bottom-up passes, the complexity of the procedure is $\mathcal{O}(|A|)$.

We remark that the algorithm above is similar in spirit to the lifting procedures for cover inequalities based on dynamic programming [60,29,59], in particular also solving a recursive model to lift coefficients. The existing techniques, however, are applicable only for the knapsack polytope, solve a new dynamic program for each inequality to be lifted, and specialize on cover inequalities. In contrast, our approach is suitable for any set X and valid linear inequality, and utilizes the same BDD to lift any given inequality (i.e., the BDD needs to be constructed only once).

4.3 Sequential Lifting and Dimension Implications

The lifting procedure detailed in Theorem 1 can be applied sequentially to strengthen an inequality. Specifically, we start with $\langle \boldsymbol{\pi}, \pi_0 \rangle$ satisfying our main assumptions (a)-(c). Next, we calculate the d-slacks, choose $i \in I$ such that $\lambda_i(\boldsymbol{\pi}) \neq 0$, and apply Theorem 1 to obtain the tuple $\langle \boldsymbol{\pi}', \pi_0' \rangle$ defining the lifted inequality. We re-apply this operation with the new $\langle \boldsymbol{\pi}', \pi_0' \rangle$, and repeat until all d-slacks are equal to zero. The procedure stops in a finite number of iterations since the face dimension increases after each rotation; see property (2) of Theorem 1. We summarize the procedure in Algorithm 1.

The choice of i in step 4 of Algorithm 1 is critical to the dimension of the resulting face, as illustrated in Example 3.

Algorithm 1 Sequential Combinatorial Lifting Procedure

procedure CombinatorialLifting($\langle \boldsymbol{\pi}, \pi_0 \rangle$, $\mathcal{B}$)
  Calculate the disjunctive slacks $\boldsymbol{\lambda}(\boldsymbol{\pi})$ using $\mathcal{B}$ as explained in §4.2
  while $\boldsymbol{\lambda}(\boldsymbol{\pi}) \neq \mathbf{0}$ **do**
    Choose $i \in I$ such that $\lambda_i(\boldsymbol{\pi}) \neq 0$
    Apply Theorem 1 to calculate $\langle \boldsymbol{\pi}', \pi_0' \rangle$
    Set $\langle \boldsymbol{\pi}, \pi_0 \rangle = \langle \boldsymbol{\pi}', \pi_0' \rangle$
    Recalculate $\boldsymbol{\lambda}(\boldsymbol{\pi})$
  return $\langle \boldsymbol{\pi}, \pi_0 \rangle$

Example 3 Consider the set $X = \{x \in \{0,1\}^3 : 5x_1 + 2x_2 + 3x_3 \leq 6\}$ and inequality $\boldsymbol{\pi}^\top \boldsymbol{x} = x_1 + x_2 + x_3 \leq 2$ that supports $\text{conv}(X)$. We have $\boldsymbol{\lambda}(\boldsymbol{\pi}) = (1, -1, -1)^\top$ and the lifted inequality with respect to $\lambda_1(\boldsymbol{\pi}) = 1$ is $\boldsymbol{\pi'}^\top \boldsymbol{x} = 2x_1 + x_2 + x_3 \leq 2$ and has $\boldsymbol{\lambda}(\boldsymbol{\pi}') = \mathbf{0}$. The lifted inequality is not facet-defining since $\dim(\text{conv}(X)) = 3$ and $\dim(F(\boldsymbol{\pi}')) = 1$.

If we instead lift $x_1 + x_2 + x_3 \leq 2$ with respect to $\lambda_2(\boldsymbol{\pi}) = -1$ the lifted inequality is $\boldsymbol{\pi'}^\top \boldsymbol{x} = x_1 + x_3 \leq 1$ and $\boldsymbol{\lambda}(\boldsymbol{\pi}') = \mathbf{0}$. In this case, the lifted inequality is facet-defining since $\dim(F(\boldsymbol{\pi}')) = 2$. □

In order to understand the impact of the index choice, we first show in Lemma 2 a relationship between d-slacks and the dimension of the face. Specifically, the cardinality of $S^0(\boldsymbol{\pi})$ bounds $\dim(F(\boldsymbol{\pi}))$. We later use this result to gauge when the sequential procedure leads to a facet-defining inequality.

Lemma 2 *The dimension of a face $F(\boldsymbol{\pi})$ satisfies $\dim(F(\boldsymbol{\pi})) \leq |S^0(\boldsymbol{\pi})|$. Moreover, $|S^0(\boldsymbol{\pi})| = 0$ if $\dim(F(\boldsymbol{\pi})) = 0$.*

Proof For any $i \in S^-(\boldsymbol{\pi}) \cup S^+(\boldsymbol{\pi})$, the value of x_i is fixed at either 0 or 1 for all $\boldsymbol{x} \in F(\boldsymbol{\pi})$ according to Lemma 1. Thus, the dimension of $\dim(F(\boldsymbol{\pi}))$ is bounded by $|S^0(\boldsymbol{\pi})|$, since at most $|S^0(\boldsymbol{\pi})| + 1$ affinely independent points can be obtained from $F(\boldsymbol{\pi})$. Now, assume $S^0(\boldsymbol{\pi}) \neq \emptyset$ and $\dim(F(\boldsymbol{\pi})) \geq 0$. There exist $\boldsymbol{x}, \boldsymbol{x}' \in F(\boldsymbol{\pi})$ such that $x_i \neq x_i'$ for $i \in S^0(\boldsymbol{\pi})$. These two points are affinely independent and therefore $\dim(F(\boldsymbol{\pi})) \geq 1$. Thus, $S^0(\boldsymbol{\pi}) = \emptyset$ if $\dim(F(\boldsymbol{\pi})) = 0$. ■

Example 3 depicts a case where $|S^0(\boldsymbol{\pi})|$ increases faster than the number of affinely independent points in $F(\boldsymbol{\pi})$. In view of Lemma 2, we would like to choose i so that $|S^0(\boldsymbol{\pi})|$ increases at a slower rate, since each lifting operation increases $\dim(F(\boldsymbol{\pi}))$ by at least one according to Theorem 1-(2). We show in Theorem 2 that the slow increase of $|S^0(\boldsymbol{\pi})|$ occurs when there exists a unique slack with minimum non-zero absolute value.

Theorem 2 *Suppose there exists $i \notin S^0(\boldsymbol{\pi})$ such that $|\lambda_i(\boldsymbol{\pi})| < |\lambda_{i'}(\boldsymbol{\pi})|$ for all $i' \notin S^0(\boldsymbol{\pi})$ $(i' \neq i)$. Then, for $\langle \boldsymbol{\pi}', \pi_0' \rangle$ obtained when lifting $\langle \boldsymbol{\pi}, \pi_0 \rangle$ with respect to $\lambda_i(\boldsymbol{\pi})$, $\dim(F(\boldsymbol{\pi}')) = \dim(F(\boldsymbol{\pi})) + 1$ and $|S^0(\boldsymbol{\pi}')| = |S^0(\boldsymbol{\pi})| + 1$.*

Proof From Lemma 1, it suffices to show that, for any $\boldsymbol{x} \in F(\boldsymbol{\pi}')$ and $i' \notin S^0(\boldsymbol{\pi})$ such that $i' \neq i$, we have: 1) $i' \in S^+(\boldsymbol{\pi})$ implies that $x_{i'} = 0$; and

2) $i' \in S^-(\boldsymbol{\pi})$ implies that $x_{i'} = 1$. In such cases, an index i' that was originally in $S^-(\boldsymbol{\pi})$ or $S^+(\boldsymbol{\pi})$ will remain in its original partition $S^-(\boldsymbol{\pi}')$ or $S^+(\boldsymbol{\pi}')$ for the lifted $\boldsymbol{\pi}'$. The statement then follows due to Theorem 1-(2) and Lemma 2.

We will focus our attention to the case $\lambda_i(\boldsymbol{\pi}) > 0$ (the others are analogous). For any $\boldsymbol{x} \in F(\boldsymbol{\pi}')$, we have by construction that $\boldsymbol{\pi}^\top \boldsymbol{x} = \pi'_0 - \lambda_i(\boldsymbol{\pi}) x_i \geq \pi_0 - \lambda_i(\boldsymbol{\pi})$. Assume, for the purpose of a contradiction, that $x_{i'} = 1$ and that $\lambda_{i'}(\boldsymbol{\pi}) > 0$. Thus, $\lambda^1_{i'}(\boldsymbol{\pi}) \geq \boldsymbol{\pi}^\top \boldsymbol{x} \geq \pi_0 - \lambda_i(\boldsymbol{\pi})$. Moreover, $\lambda^0_{i'}(\boldsymbol{\pi}) = \pi_0$. This implies that $\lambda_{i'}(\boldsymbol{\pi}) = \lambda^0_{i'}(\boldsymbol{\pi}) - \lambda^1_{i'}(\boldsymbol{\pi}) \leq \pi_0 - \pi_0 + \lambda_i(\boldsymbol{\pi}) \leq \lambda_i(\boldsymbol{\pi})$ and hence $0 < \lambda_{i'}(\boldsymbol{\pi}) \leq \lambda_i(\boldsymbol{\pi})$. This cannot hold since $|\lambda_i(\boldsymbol{\pi})| < |\lambda_{i'}(\boldsymbol{\pi})|$.

Similarly, assume that $\lambda_{i'}(\boldsymbol{\pi}) < 0$ and $x_{i'} = 0$. Then, $\lambda^0_{i'}(\boldsymbol{\pi}) \geq \pi_0 - \lambda_i(\boldsymbol{\pi})$ and $\lambda^1_{i'}(\boldsymbol{\pi}) = \pi_0$. This implies that $\lambda_{i'}(\boldsymbol{\pi}) = \lambda^0_{i'}(\boldsymbol{\pi}) - \lambda^1_{i'}(\boldsymbol{\pi}) \geq \pi_0 - \lambda_i(\boldsymbol{\pi}) - \pi_0 = -\lambda_i(\boldsymbol{\pi})$. Thus, $0 > \lambda_{i'}(\boldsymbol{\pi}) \geq -\lambda_i(\boldsymbol{\pi})$. This contradicts $|\lambda_i(\boldsymbol{\pi})| < |\lambda_{i'}(\boldsymbol{\pi})|$. ■

Theorem 2 provides a simple choice rule based on picking i with the minimum absolute d-slack. It also indicates when this rule will converge to a facet-defining inequality. We formalize it in Corollary 1 below, which can be derived as a direct consequence of Theorem 2.

Corollary 1 *If* $\dim(F(\boldsymbol{\pi})) = |S^0(\boldsymbol{\pi})|$*, the sequential lifting procedure (Algorithm 1) with the minimum slack absolute rule produces a facet-defining inequality if, at each lifting iteration except the last, the chosen* $i \in I$ *is such that* $|\lambda_i(\boldsymbol{\pi})| < |\lambda_{i'}(\boldsymbol{\pi})|$ *for all* $i' \notin S^0(\boldsymbol{\pi})$ $(i' \neq i)$*.*

Finally, we note that, in general, it may not be possible to achieve a facet-defining inequality. For example, all non-zero d-slacks can have the same absolute value and the cardinality of $|S^0(\boldsymbol{\pi})|$ might increase by more than one while the dimension of $F(\boldsymbol{\pi})$ does not (see Example 3).

4.4 Relationship with Lifting based on Disjunctive Programming

We now formalize the connection between our lifting methodology and the n-step lifting procedure by Perregaard and Balas [50] mentioned in §3. Assume that $X = \{A\boldsymbol{x} \leq \boldsymbol{b},\ \boldsymbol{x} \in \{0,1\}^n\}$ for a matrix A and vector $\boldsymbol{b}$ of appropriate dimensions. We consider a relaxation of the form DP-L that includes one disjunctive term for each index $i \in I$ and removes all integrality constraints:

$$\max_{\boldsymbol{x}} \left\{ \boldsymbol{c}^\top \boldsymbol{x} :\ A\boldsymbol{x} \leq \boldsymbol{b},\ \bigvee_{i' \in I} (x_{i'} \leq 0) \vee (x_{i'} \geq 1),\ \boldsymbol{x} \in [0,1]^n \right\}. \qquad \text{(DP-L)}$$

Proposition 1 below shows that, for DP-L, the optimal rotation parameter in the n-step lifting is such that $\gamma^* = |\lambda_i(\boldsymbol{\pi})|$ when using the individual binary disjunctions as target inequalities.

Proposition 1 *Suppose* $\lambda_i(\boldsymbol{\pi}) \neq 0$ *for some* $i \in I$*. Then,* $\gamma^* = |\lambda_i(\boldsymbol{\pi})|$ *if we employ either* $x_i \leq 0$ *or* $x_i \geq 1$ *as a target inequality in the n-step procedure.*

Proof Consider the case when $i \in S^+(\boldsymbol{\pi})$ and let $\boldsymbol{e}_i$ be the i-th column of an $n \times n$ identity matrix. Since all $\boldsymbol{x} \in F(\boldsymbol{\pi})$ have $x_i = 0$, $\widetilde{\boldsymbol{\pi}}^\top \boldsymbol{x} = \boldsymbol{e}_i^\top \boldsymbol{x} = x_i \leq 0$ is an invalid target inequality for $\text{conv}(X)$ and satisfies $\widetilde{\boldsymbol{\pi}}^\top \boldsymbol{x} = x_i = 0$ for all $\boldsymbol{x} \in F(\boldsymbol{\pi})$. The system that defines γ^* is therefore

$$\gamma^* = \min_{\boldsymbol{x} \in [0,1]^n, x_0 \geq 0} \Big\{ \pi_0 x_0 - \boldsymbol{\pi}^\top \boldsymbol{x} : \ A\boldsymbol{x} \leq \boldsymbol{b}, \ -x_i = -1, \\ \bigvee_{i' \in I} (x_{i'} \leq 0) \vee (-x_{i'} + x_0 \leq 0) \Big\}. \tag{1}$$

It follows from (1) and $x_i = 1$ that $x_0 \leq 1$. Without loss of generality, we consider $x_0 = 1$. The system reduces to:

$$\begin{aligned} \gamma^* &= \min_{\boldsymbol{x}, x_0} \left\{ \pi_0 x_0 - \boldsymbol{\pi}^\top \boldsymbol{x} : A\boldsymbol{x} \leq \boldsymbol{b}, \ x_i = 1, \ \boldsymbol{x} \in \{0,1\}^n, \ x_0 = 1 \right\} \\ &= \pi_0 - \max_{\boldsymbol{x}} \{ \boldsymbol{\pi}^\top \boldsymbol{x} : \boldsymbol{x} \in X, \ x_i = 1 \} = \lambda_i^0(\boldsymbol{\pi}) - \lambda_i^1(\boldsymbol{\pi}) = \lambda_i(\boldsymbol{\pi}). \end{aligned}$$

The second to last equality comes from $i \in S^+(\boldsymbol{\pi})$ and $\lambda_i^0(\boldsymbol{\pi}) = \pi_0$. The proof for $i \in S^-(\boldsymbol{\pi})$ and target inequality $\widetilde{\boldsymbol{\pi}}^\top \boldsymbol{x} = x_i \geq 1$ is analogous. ■

Proposition 1 indicates when these techniques are equivalent. By taking $\widetilde{\boldsymbol{\pi}}^\top \boldsymbol{x} = x_i \leq 0$ as the target inequality, we obtain $\gamma^* = \lambda_i(\boldsymbol{\pi}) > 0$. The rotated inequality $(\boldsymbol{\pi} + \gamma^* \widetilde{\boldsymbol{\pi}})^\top \boldsymbol{x} = (\boldsymbol{\pi} + \lambda_i(\boldsymbol{\pi}) \boldsymbol{e}_i)^\top \boldsymbol{x} \leq \pi_0$ is equivalent to the lifted inequality in Theorem 1. Similarly, using target inequality $\widetilde{\boldsymbol{\pi}}^\top \boldsymbol{x} = -x_i \leq -1$ would result in $\gamma^* = -\lambda_i(\boldsymbol{\pi}) > 0$. Then, the rotated and lifted inequalities are equivalent, i.e., $\boldsymbol{\pi} + \gamma^* \widetilde{\boldsymbol{\pi}} = \boldsymbol{\pi} + \lambda_i(\boldsymbol{\pi}) \boldsymbol{e}_i$ and $\pi_0 + \gamma^* \widetilde{\pi}_0 = \pi_0 + \lambda_i(\boldsymbol{\pi})$.

While the techniques are equivalent in this restricted case, our approach is valid for any binary set X and, thus, can handle models where a BDD (or BDD relaxation) is a more advantageous representation in comparison to a linear description of X [14]. We also note that the BDD network structure allows us to efficiently compute the disjunctive terms in a combinatorial fashion.

5 Combinatorial Cutting-Plane Algorithm

While the BDD-based lifting procedure developed in §4 can enhance inequalities from any cutting-plane methodology, we now exploit similar concepts to derive new valid inequalities for X based on the network structure of $\mathcal{B}$. In particular, we design inequalities that separate points from X by only relying on the combinatorial structure encoded by $\mathcal{B}$. Thus, no other specific structure (e.g., linearity, submodularity, or gradient information) is required.

We assume, as before, that we are given an exact BDD $\mathcal{B}$ for X. Our cutting-plane method is based on an alternative linear description of $\mathcal{B}$ as an extended capacitated flow problem. We present this formulation in §5.1 and our BDD-based cut generation linear program in §5.2. For cases where solving such model is not computationally practical, in §5.3 we develop a weaker but more efficient combinatorial cutting-plane method based on a max-flow/min-cut problem over $\mathcal{B}$. Finally, we show in §5.4 the relationship between our approach and existing BDD cutting-plane techniques [27,55].

5.1 BDD Polytope

Existing BDD-based cut generation procedures [27,44,55] rely on the network-flow formulation $\text{NF}(\mathcal{B})$ introduced by Behle [12], described as follows:

$$\text{NF}(\mathcal{B}) := \Big\{(\boldsymbol{x};\boldsymbol{y}) \in [0,1]^n \times \mathbb{R}_+^{|\mathcal{A}|} :$$

$$\sum_{a\in\mathcal{A}^{\text{out}}(u)} y_a - \sum_{a\in\mathcal{A}^{\text{in}}(u)} y_a = 0, \qquad \forall u \in \mathcal{N}\setminus\{\mathbf{r},\mathbf{t}\}, \tag{2a}$$

$$\sum_{a\in\mathcal{A}^{\text{out}}(\mathbf{r})} y_a = \sum_{a\in\mathcal{A}^{\text{in}}(\mathbf{t})} y_a = 1, \tag{2b}$$

$$\sum_{a\in\mathcal{A}:s(a)\in\mathcal{N}_i,v_a=1} y_a = x_i, \qquad \forall i \in I\Big\}. \tag{2c}$$

Equalities (2a) and (2b) are balance-of-flow constraints over $\mathcal{B}$. Constraint (2c) links the arcs of $\mathcal{B}$ with solutions $\boldsymbol{x}$. In particular, the polytope $\text{NF}(\mathcal{B})$ projected over the $\boldsymbol{x}$ variables is equivalent to the convex hull of all solutions represented by $\mathcal{B}$, i.e., $\text{Proj}_{\text{x}}(\text{NF}(\mathcal{B})) = \text{conv}(X)$.

One drawback of $\text{NF}(\mathcal{B})$ is that constraints (2c) only consider flow variables associated with arc labels equal to one (i.e., $v_a = 1$). Thus, there is no constraint explicitly limiting the flow passing through zero-value arcs. CGLPs based on $\text{NF}(\mathcal{B})$ are potentially unbounded, which has been a fundamental challenge in existing works [55,27].

We propose an alternative formulation of $\text{NF}(\mathcal{B})$ that addresses its main limitations and use the reformulation to define our cutting-plane algorithms. The new formulation, here denoted by $\text{JNF}(\mathcal{B})$, corresponds to a joint capacitated network-flow polytope. The new model maintains the flow conservation constraints, (2a) and (2b), and replaces (2c) with (3a) and (3b) below. Both inequalities enforce a common capacity for arcs in a layer with the same value. Proposition 2 shows that the two formulations are equivalent and, thus, $\text{Proj}_{\text{x}}(\text{JNF}(\mathcal{B})) = \text{conv}(X)$.

$$\text{JNF}(\mathcal{B}) := \Big\{(\boldsymbol{x};\boldsymbol{y}) \in [0,1]^n \times \mathbb{R}_+^{|\mathcal{A}|} : (2a) - (2b),$$

$$\sum_{a\in\mathcal{A}:s(a)\in\mathcal{N}_i,v_a=1} y_a \le x_i, \qquad \forall i \in I, \tag{3a}$$

$$\sum_{a\in\mathcal{A}:s(a)\in\mathcal{N}_i,v_a=0} y_a \le 1 - x_i, \qquad \forall i \in I\Big\}. \tag{3b}$$

Cut generation methods based on capacitated network flows over BDDs have been previously studied in the context of two-stage stochastic programs [44]. In our model, the proposed polytope $\text{JNF}(\mathcal{B})$ is specially structured given, e.g., the use a single variable x_i per layer i to limit the capacity of the zero and one-value arcs. This structural property give us desired properties for separation in §5.2, and is key to when developing combinatorial cuts that do not depend on linear programs in §5.3.

Proposition 2 *JNF*($\mathcal{B}$) = *NF*($\mathcal{B}$).

Proof Consider $(\boldsymbol{x}';\boldsymbol{y}') \in \text{NF}(\mathcal{B})$, so $(\boldsymbol{x}';\boldsymbol{y}')$ satisfies (2a) and (2b). Since (2c) holds, $(\boldsymbol{x}';\boldsymbol{y}')$ also satisfies (3a). From the flow conservation constraints, (2a) and (2b), the flow traversing each layer $i \in I$ in $\mathcal{B}$ is exactly one, i.e.,

$$\sum_{a\in\mathcal{A}:\ s(a)\in\mathcal{N}_i} y'_a = 1 \quad \Rightarrow \quad \sum_{a\in\mathcal{A}:s(a)\in\mathcal{N}_i:\ v_a=1} y'_a + \sum_{a\in\mathcal{A}:s(a)\in\mathcal{N}_i:\ v_a=0} y'_a = 1$$
$$\Rightarrow \quad \sum_{a\in\mathcal{A}:s(a)\in\mathcal{N}_i:\ v_a=0} y'_a = 1 - x'_i.$$

Then, $(\boldsymbol{x}';\boldsymbol{y}')$ satisfies (3b) and $(\boldsymbol{x}';\boldsymbol{y}') \in \text{JNF}(\mathcal{B})$. Consider now $(\boldsymbol{x}';\boldsymbol{y}') \in \text{JNF}(\mathcal{B})$. Since flows traversing a layer sum to one, constraints (3a) and (3b) are satisfied as equalities and therefore (2c) holds for $(\boldsymbol{x}';\boldsymbol{y}')$. ■

5.2 General BDD Flow Cuts

Our cutting-plane procedure formulates a max-flow optimization problem over JNF($\mathcal{B}$) to identify and separate points $\boldsymbol{x}' \notin \text{conv}(X)$, given by (4) below:

$$z(\mathcal{B};\boldsymbol{x}') := \max_{\boldsymbol{y}\in\mathbb{R}_+^{|\mathcal{A}|}} \left\{ \sum_{a\in\mathcal{A}^{\text{out}}(\mathbf{r})} y_a :\ (2a), (3a)-(3b), \boldsymbol{x}=\boldsymbol{x}' \right\}. \tag{4}$$

This model omits constraint (2b) which enforces the flow to be equal to one. We argue in Lemma 3 that $z(\mathcal{B};\boldsymbol{x}') = 1$ is a necessary and sufficient condition to check if $\boldsymbol{x}'$ belongs to conv(X).

Lemma 3 $\boldsymbol{x}' \in \text{conv}(X)$ *if and only if* $z(\mathcal{B};\boldsymbol{x}') = 1$.

Proof Constraints (3a) and (3b) enforce that the flow in each layer i is at most $x'_i + 1 - x'_i = 1$. Thus, $z(\mathcal{B};\boldsymbol{x}') \leq 1$. Consider $\boldsymbol{x}' \in \text{conv}(X)$. From Proposition 2, there exists $\boldsymbol{y}' \in \mathbb{R}_+^{|\mathcal{A}|}$ such that $\sum_{a\in\mathcal{A}^{\text{out}}(\mathbf{r})} y'_a = 1$ and therefore $z(\mathcal{B};\boldsymbol{x}') = 1$. For the converse, suppose $z(\mathcal{B};\boldsymbol{x}') = 1$. It follows that there exists $\boldsymbol{y}' \in \mathbb{R}_+^{|\mathcal{A}|}$ such that $(\boldsymbol{x}';\boldsymbol{y}') \in \text{JNF}(\mathcal{B}) = \text{NF}(\mathcal{B})$, so $\boldsymbol{x}' \in \text{conv}(X)$. ■

Our BDD-based CGLP uses the dual of (4) to separate $\boldsymbol{x}' \notin \text{conv}(X)$. Consider $\boldsymbol{\omega} \in \mathbb{R}^{|\mathcal{N}|}$ and $\boldsymbol{\nu}, \boldsymbol{\eta} \in \mathbb{R}^n_+$ as the dual variables associated with constraints (2a), (3a), and (3b), respectively. The resulting model is

$$\min_{\boldsymbol{\omega},\boldsymbol{\nu},\boldsymbol{\eta}} \quad \sum_{i\in I} x'_i\nu_i + \sum_{i\in I}(1-x'_i)\eta_i \tag{BDD-CGLP}$$
$$\text{s.t.} \quad \omega_{t(a)} - \omega_{s(a)} + v_a\nu_i + (1-v_a)\eta_i \geq 0, \quad \forall i\in I, a\in\mathcal{A}, s(a)\in\mathcal{N}_i, \tag{5a}$$
$$\omega_{t(a)} + v_a\nu_1 + (1-v_a)\eta_1 \geq 1, \quad \forall a\in\mathcal{A}^{\text{out}}(\mathbf{r}), \tag{5b}$$
$$-\omega_{s(a)} + v_a\nu_n + (1-v_a)\eta_n \geq 0, \quad \forall a\in\mathcal{A}^{\text{in}}(\mathbf{t}), \tag{5c}$$
$$\boldsymbol{\omega}\in\mathbb{R}^{|\mathcal{N}|},\ \boldsymbol{\nu},\boldsymbol{\eta}\in\mathbb{R}^n_+. \tag{5d}$$

Let $w(\mathcal{B};\boldsymbol{x}')$ be the optimal solution value of BDD-CGLP. Strong duality and Lemma 3 imply that we can identify if a point $\boldsymbol{x}'$ belongs to $\text{conv}(X)$ if $w(\mathcal{B};\boldsymbol{x}')=1$. Furthermore, we can use the optimal solution $(\boldsymbol{\nu}^*;\boldsymbol{\eta}^*)$ to create a valid cut when $w(\mathcal{B};\boldsymbol{x}')<1$. Specifically, the cut is given by

$$\sum_{i\in I} x_i\nu_i^* + \sum_{i\in I}(1-x_i)\eta_i^* \geq 1. \tag{6}$$

Theorem 3 shows that the set of all cuts of the form (6) describes $\text{conv}(X)$.

Theorem 3 *Let $\Lambda(\mathcal{B})$ be the set of extreme points of the BDD-CGLP polyhedron defined by (5a)-(5d). Furthermore, let $P_{\mathcal{B}}$ be the set of points $\boldsymbol{x}\in[0,1]^n$ that satisfy (6) for all $(\boldsymbol{\nu};\boldsymbol{\eta})\in \text{Proj}_{\boldsymbol{\nu},\boldsymbol{\eta}}(\Lambda(\mathcal{B}))$. Then, $\text{conv}(X)=P_{\mathcal{B}}$.*

Proof Consider a point $\boldsymbol{x}'\in\text{conv}(X)$. Lemma 3 guarantees that $z(\mathcal{B};\boldsymbol{x}')=w(\mathcal{B};\boldsymbol{x}')=1$, so constraint (6) holds for any extreme point of BDD-CGLP. Now consider a point $\boldsymbol{x}'\in P_{\mathcal{B}}$. Since $\boldsymbol{x}'$ satisfies (6) for all extreme points in $\Lambda(\mathcal{B})$, we have that $w(\mathcal{B};\boldsymbol{x}')\geq 1$ and, thus, $w(\mathcal{B};\boldsymbol{x}')=z(\mathcal{B};\boldsymbol{x}')=1$. Finally, using Lemma 3, we have that $\boldsymbol{x}'\in\text{conv}(X)$. ■

Thus, we solve BDD-CGLP to separate points $\boldsymbol{x}'\notin\text{conv}(X)$. The procedure returns a cut (6) where $(\boldsymbol{\nu}^*;\boldsymbol{\eta}^*)$ is the optimal solution of BDD-CGLP.

5.3 Combinatorial BDD Flow Cuts

The above cutting-plane procedure requires solving a linear program with $|\mathcal{A}|$ constraints and $|\mathcal{N}|+2n$ variables. Obtaining $w(\mathcal{B};\boldsymbol{x}')$, thus, could be computationally expensive for instances where $\mathcal{B}$ is large (see §7). We propose an alternative cut-generation procedure based on BDD-CGLP that involves a combinatorial and more efficient max-flow solution over $\mathcal{B}$.

First, we consider a reformulation of $\text{JNF}(\mathcal{B})$ where the joint capacity constraints are replaced by individual constraints for each arc, i.e., a standard capacitated network flow polytope over $\mathcal{B}$:

$$\begin{aligned}\text{CNF}(\mathcal{B}) := \{(\boldsymbol{x};\boldsymbol{y})\in[0,1]^n\times\mathbb{R}_+^{|\mathcal{A}|} : &\text{ (2a)}-\text{(2b)},\\ y_a \leq x_i, \quad &\forall a\in\mathcal{A}, s(a)\in\mathcal{N}_i, v_a=1, i\in I, &\text{(7a)}\\ y_a \leq 1-x_i, \quad &\forall a\in\mathcal{A}, s(a)\in\mathcal{N}_i, v_a=0, i\in I\}. &\text{(7b)}\end{aligned}$$

Proposition 3 *JNF($\mathcal{B}$) $\subseteq$ CNF($\mathcal{B}$). Moreover, for any integer $\boldsymbol{x}\notin\text{conv}(X)$, we have that $\boldsymbol{x}\notin\text{Proj}_{\text{x}}(\text{CNF}(\mathcal{B}))$.*

Proof Consider $\boldsymbol{x}'\in\text{JNF}(\mathcal{B})$. By construction, $\boldsymbol{x}'$ satisfies (2a)-(2b). Since $\boldsymbol{x}'$ satisfies (3a) and (3b), it follows that $\boldsymbol{x}'$ holds for (7a) and (7b).

Now take an integer point $\boldsymbol{x}'\notin\text{conv}(X)$ and $\boldsymbol{y}'\in\mathbb{R}_+^{|\mathcal{A}|}$ such that $(\boldsymbol{x}';\boldsymbol{y}')$ satisfies (2a), (7a)-(7b). Notice that such a $\boldsymbol{y}'$ exists (e.g., $\boldsymbol{y}'=\boldsymbol{0}$). By construction, there is no path $p\in\mathcal{P}$ associated with $\boldsymbol{x}'$. From constraints (7a)-(7b), in any path $p\in\mathcal{P}$ there exists an arc $a\in p$ with capacity zero (i.e., $y_a\leq 0$). We can then deduce that $\boldsymbol{y}'=\boldsymbol{0}$, therefore $(\boldsymbol{x}';\boldsymbol{y}')$ violates (2b). Finally, for any $\boldsymbol{x}'\in\{0,1\}^n\setminus\text{conv}(X)$ there is no $\boldsymbol{y}'\in\mathbb{R}_+^{|\mathcal{A}|}$ such that $(\boldsymbol{x}';\boldsymbol{y}')\in\text{CNF}(\mathcal{B})$. ■

Proposition 3 shows that for any integer point $\boldsymbol{x}'$, $\boldsymbol{x}' \notin X$ implies $\boldsymbol{x}' \notin \text{Proj}_{\text{x}}(\text{CNF}(\mathcal{B}))$. Example 4 illustrates that, conversely, there might exist fractional points $\boldsymbol{x}' \notin \text{conv}(X)$ such that $\boldsymbol{x}' \in \text{Proj}_{\text{x}}(\text{CNF}(\mathcal{B}))$, and hence $\text{CNF}(\mathcal{B})$ is a weaker representation.

Example 4 Consider our example $X = \{\boldsymbol{x} \in \{0,1\}^4 : 7x_1+5x_2+4x_3+x_4 \leq 8\}$, a fractional point $\boldsymbol{x}' = (0.4, 0.6, 0.4, 1)$, and the exact BDD $\mathcal{B}_1$ in Figure 1. It is easy to see that $\boldsymbol{x}' \notin \text{conv}(X)$ since $7x'_1+5x'_2+4x'_3+x'_4 = 8.4 \geq 8$. However, there exists a $\boldsymbol{y}' \in \mathbb{R}_+^{|\mathcal{A}|}$ such that $(\boldsymbol{x}', \boldsymbol{y}') \in \text{CNF}(\mathcal{B})$ with value $y_{(\mathbf{r},u_1)} = 0.6$, $y_{(\mathbf{r},u_2)} = 0.4$, $y_{(u_1,u_4)} = 0.2$, $y_{(u_1,u_3)} = 0.4$, $y_{(u_2,u_4)} = 0.4$, $y_{(u_3,u_4)} = 0.4$, $y_{(u_4,u_5)} = 0.6$, $y_{(u_5,\mathbf{t})} = 1$, and all other arcs with flow equal to zero. □

Similar to the general BDD flow cuts, we use the dual of the max-flow version of $\text{CNF}(\mathcal{B})$ to identify points that do not belong to $\text{conv}(X)$. Consider $\boldsymbol{\omega} \in \mathbb{R}^{|\mathcal{N}|}$ as the dual variables associated with constraints (2a) and $\boldsymbol{\alpha}$ the dual variables associated with constraints (7a)-(7b). Then, the separation problem for the alternative BDD cuts is as follow:

$$\begin{aligned}
\min_{\boldsymbol{\omega},\boldsymbol{\alpha}} \quad & \sum_{i\in I}\left(\sum_{a\in\mathcal{A}:s(a)\in\mathcal{N}_i,v_a=1} x'_i\alpha_a \;+\; \sum_{a\in\mathcal{A}:s(a)\in\mathcal{N}_i,v_a=0}(1-x'_i)\alpha_a\right) && \text{(CN-CGLP)}\\
\text{s.t.} \quad & \omega_{t(a)} - \omega_{s(a)} + \alpha_a \geq 0, && \forall i \in I, a \in \mathcal{A}, s(a) \in \mathcal{N}_i,\\
& \omega_{t(a)} + \alpha_a \geq 1, && \forall a \in \mathcal{A}^{\text{out}}(\mathbf{r}),\\
& -\omega_{s(a)} + \alpha_a \geq 0, && \forall a \in \mathcal{A}^{\text{in}}(\mathbf{t}),\\
& \boldsymbol{\omega} \in \mathbb{R}^{|\mathcal{N}|},\ \boldsymbol{\alpha} \in \mathbb{R}_+^{|\mathcal{A}|}.
\end{aligned}$$

Let $w^{\text{r}}(\mathcal{B}; \boldsymbol{x}')$ be the optimal solution value of CN-CGLP. Proposition 3 implies that for any $\boldsymbol{x}' \in \text{conv}(X_{\mathcal{B}})$, $w^{\text{r}}(\mathcal{B}; \boldsymbol{x}') = 1$. It follows that inequality (9) holds for any $\boldsymbol{x} \in \text{conv}(X_{\mathcal{B}})$, where $\boldsymbol{\alpha}^*$ is optimal to CN-CGLP:

$$\sum_{i\in I}\left(\sum_{a\in\mathcal{A}:s(a)\in\mathcal{N}_i,v_a=1} x_i\alpha^*_a + \sum_{a\in\mathcal{A}:s(a)\in\mathcal{N}_i,v_a=0}(1-x_i)\alpha^*_a\right) \geq 1. \tag{9}$$

Of important note is that CN-CGLP is a classical min-cut problem, i.e., we are searching for a maximum-capacity arc cut in the network that certifies that a point does not belong to the convex hull of X. While the resulting inequalities are not as strong as the general BDD cuts from BDD-CGLP, we can leverage max-flow/min-cut combinatorial algorithms to solve it more efficiently in the size of the BDD. Several algorithms are readily available to that end [1] and provide both primal and dual solutions to CN-CGLP.

Furthermore, another consequence of the design of such cuts is that their strength depends on the BDD size. That is, two BDDs $\mathcal{B}$ and $\mathcal{B}'$ encoding the same set might generate different combinatorial flow cuts because of distinct min-cut solutions. We show in Theorem 4 that the reduced BDD, which is unique, generates the tightest $\text{CNF}(\mathcal{B})$ formulation and is hence critical in such a formulation. We note that a reduced BDD can be generated in polynomial time in $\mathcal{B}'$ for any $\mathcal{B}'$ representing the desired solution set [22].

Theorem 4 *Let $\mathcal{B}^{\mathsf{r}} = (\mathcal{N}^{\mathsf{r}}, \mathcal{A}^{\mathsf{r}})$ be the reduced version of $\mathcal{B}$, i.e., $X_{\mathcal{B}^{\mathsf{r}}} = X_{\mathcal{B}}$, and for each layer $i \in I$, $|\mathcal{N}_i^{\mathsf{r}}| \leq |\mathcal{N}_i|$. Then, CNF($\mathcal{B}^{\mathsf{r}}$) $\subseteq$ CNF($\mathcal{B}$).*

Proof Consider $\mathcal{P}^{\mathsf{r}}$ to be the set of $\mathbf{r}-\mathbf{t}$ paths in $\mathcal{B}^{\mathsf{r}}$. First, $X_{\mathcal{B}^{\mathsf{r}}} = X_{\mathcal{B}}$ implies that, for any $\mathbf{r}-\mathbf{t}$ path $p \in \mathcal{P}$, there exists a unique $\mathbf{r}-\mathbf{t}$ path $p' \in \mathcal{P}^{\mathsf{r}}$ such that $\boldsymbol{x}^p = \boldsymbol{x}^{p'}$. Thus, we will consider that the set of paths in both BDDs are equivalent, i.e., $\mathcal{P}^{\mathsf{r}} = \mathcal{P}$.

Let $\mathcal{A}_i = \{a \in \mathcal{A} \colon s(a) \in \mathcal{N}_i\}$ and $\mathcal{A}_i^{\mathsf{r}} = \{a \in \mathcal{A}^{\mathsf{r}} \colon s(a) \in \mathcal{N}_i^{\mathsf{r}}\}$. Since $\mathcal{B}^{\mathsf{r}}$ is unique, there exists a unique surjective function $f_i : \mathcal{A}_i \to \mathcal{A}_i^{\mathsf{r}}$ that maps arcs from $\mathcal{B}$ to $\mathcal{B}^{\mathsf{r}}$ for each layer $i \in I$. Thus, for every arc $a \in \mathcal{A}_i^{\mathsf{r}}$, let us define the pre-image of f_i as $f_i^{-1}(a) := \{a' \in \mathcal{A}_i : \ f(a') = a\}$, i.e., the subset of arcs in $\mathcal{A}_i$ that map to arc $a \in \mathcal{A}_i^{\mathsf{r}}$. Next, denote by $\Gamma(\mathcal{B}; a) := \{p \in \mathcal{B} : a \in p\}$ the set of paths in a BDD that traverse an arc $a \in \mathcal{A}$. From the construction procedure of $\mathcal{B}^{\mathsf{r}}$ given $\mathcal{B}$ [22], $\Gamma(\mathcal{B}^{\mathsf{r}}; a) = \bigcup_{a' \in f_i^{-1}(a)} \Gamma(\mathcal{B}; a')$ for all $a \in \mathcal{A}_i^{\mathsf{r}}$, i.e., the set of $\mathbf{r}-\mathbf{t}$ paths passing through a is equivalent to the set of $\mathbf{r}-\mathbf{t}$ paths passing through all the arcs in $f_i^{-1}(a)$.

Now consider the path formulation of CNF($\mathcal{B}$), CNF$^P(\mathcal{B})$. It suffices to show that CNF$^P(\mathcal{B}^{\mathsf{r}}) \subseteq$ CNF$^P(\mathcal{B})$. Since the paths in $\mathcal{B}$ and $\mathcal{B}^{\mathsf{r}}$ are equivalent, we will consider equivalent variables $\boldsymbol{w}$ for CNF$^P(\mathcal{B}^{\mathsf{r}})$ and CNF$^P(\mathcal{B})$.

$$\begin{aligned}
\text{CNF}^P(\mathcal{B}) := \{(\boldsymbol{x}; \boldsymbol{w}) \in [0,1]^n \times \mathbb{R}_+^{|X(\mathcal{B})|} :& \\
\sum_{p \in \mathcal{P} : a \in p} w_p \leq x_i, \qquad & \forall a \in \mathcal{A}_i, v_a = 1, i \in I, \qquad \text{(10a)} \\
\sum_{p \in \mathcal{P} : a \in p} w_p \leq 1 - x_i, \qquad & \forall a \in \mathcal{A}_i, v_a = 0, i \in I\}. \qquad \text{(10b)}
\end{aligned}$$

Using the path equivalence $\Gamma(\mathcal{B}^{\mathsf{r}}; a) = \bigcup_{a' \in f_i^{-1}(a)} \Gamma(\mathcal{B}; a')$ for any $a \in \mathcal{A}_i^{\mathsf{r}}$ and $i \in I$, we have that

$$\sum_{p \in \mathcal{P}^{\mathsf{r}} : a \in p} w_p = \sum_{a' \in f_i^{-1}(a)} \sum_{p \in \mathcal{P} : a' \in p} w_p, \qquad \forall a \in \mathcal{A}_i^{\mathsf{r}},\ i \in I.$$

Thus, constraints (10a) and (10b) of CNF$^P(\mathcal{B}^{\mathsf{r}})$ are tighter since they restrict more paths than for the case of CNF$^P(\mathcal{B})$. This implies that, for any $(\boldsymbol{x}'; \boldsymbol{w}') \in$ CNF$^P(\mathcal{B}^{\mathsf{r}})$, $(\boldsymbol{x}'; \boldsymbol{w}') \in$ CNF$^P(\mathcal{B})$. ■

Theorem 4 indicates that the reduced BDD can separate more points than any other BDD representing the same solution set. We also note in passing that the variable ordering plays a role on the size of the BDD and, hence, on the effectiveness of the combinatorial BDD flow cuts. Investigating variable orderings for specific problem classes and how they impact the cuts (theoretically and computationally) may lead to new research avenues.

5.4 Relationship with Existing BDD Cut Generation Procedures

The two existing BDD-based CGLPs rely on dual reformulations of NF($\mathcal{B}$), and, thus, also describe conv(X) [55,27]. These techniques rely on additional information: Tjandraatmadja et al. (2019) [55] CGLP requires an interior point of X and Davarnia et al. (2020) [27] must incorporate possibly non-linear normalization constraints. In contrast, BDD-CGLP exploits the structure of $\mathcal{B}$ directly to describe conv(X). We now detail these two BDD-based CGLPs and highlight the main theoretical differences to BDD-CGLP.

Consider $\boldsymbol{\omega} \in \mathbb{R}^{|\mathcal{N}|}$ as the dual variables associated with constraints (2a) and (2b), and $\boldsymbol{\theta} \in \mathbb{R}^n$ as the dual variables associated with (2c). The two BDD-based CGLP models employ flow inequalities of the form

$$\omega_{t(a)} - \omega_{s(a)} + \theta_i v_a \geq 0, \qquad \forall i \in I, a \in \mathcal{A}, s(a) \in \mathcal{N}_i. \tag{11}$$

Notice that (11) resembles the flow inequalities (5a)-(5c) of BDD-CGLP. However, our flow constraints use two sets of positive dual variables for each BDD layer (i.e., $\boldsymbol{\nu}, \boldsymbol{\eta} \in \mathbb{R}^n_+$) instead of the single unbounded set of variables $\boldsymbol{\theta} \in \mathbb{R}^n$. This difference emerges because (2c) only bounds the arc flow variables $\boldsymbol{y} \in \mathbb{R}^{|\mathcal{A}|}_+$ with value $v_a = 1$, while our joint-capacity constraints (3a)-(3b) bound all variables $\boldsymbol{y}$. This is one reason, e.g., why BDD-CGLP does not require any normalization as in previous techniques.

Tjandraatmadja et al. (2019) [55] propose a BDD-based CGLP (12) to generate target cuts that are facet-defining. Their CGLP yields a valid inequality that intersects the ray passing through an interior point $\boldsymbol{u} \in$ conv(X) and the fractional point $\boldsymbol{x}' \in [0,1]^n$ to be cut-off. The procedure returns a cut $\boldsymbol{\theta}^{*\top}\boldsymbol{x}' \leq 1 + \boldsymbol{\theta}^{*\top}\boldsymbol{u}$ whenever the optimal value of (12) is greater than one.

$$\max_{\boldsymbol{\omega},\boldsymbol{\theta}} \left\{ \boldsymbol{\theta}^\top(\boldsymbol{x}' - \boldsymbol{u}) : \ (11),\ \omega_{\mathbf{t}} = 0,\ \omega_{\mathbf{r}} = 1 + \boldsymbol{\theta}^\top \boldsymbol{u} \right\}. \tag{12}$$

Davarnia et al. (2020) [27] circumvent the need of an interior point by proposing a simpler but possibly non-linear BDD-based CGLP presented in (13). The model checks if $\boldsymbol{x}'$ can be represented as a linear combination of points in X, i.e., whether there exists $\boldsymbol{\theta}, \boldsymbol{\omega}$ such that $\boldsymbol{\theta}^\top \boldsymbol{x}' = \omega_{\mathbf{t}}$. Otherwise, their procedure returns a valid inequality $\boldsymbol{\theta}^{*\top}\boldsymbol{x} \leq \omega^*_{\mathbf{t}}$, which is not necessarily facet-defining. Since the model may be unbounded, the optimization problem (13) includes normalization constraints $\mathcal{C}(\boldsymbol{\omega}, \boldsymbol{\theta}) \leq 0$ which are potentially non-linear. The CGLP (13) is addressed by an iterative subgradient algorithm.

$$\max_{\boldsymbol{\omega},\boldsymbol{\theta}} \left\{ \boldsymbol{\theta}^\top \boldsymbol{x}' - \omega_{\mathbf{t}} : \ (11),\ \omega_{\mathbf{r}} = 0,\ \mathcal{C}(\boldsymbol{\omega}, \boldsymbol{\theta}) \leq 0 \right\}. \tag{13}$$

Note that, in our approach, we either solve BDD-CGLP (a linear program) or a single max-flow/min-cut problem, both relying only on $\mathcal{B}$. We compare our cutting-plane approaches with these procedures in §7.

6 Case Study: Second-order Cone Programming

For our numerical evaluation, we apply our combinatorial cut-and-lift procedure to binary problems with SOC inequalities with the following form

$$\max_{\boldsymbol{x}\in\{0,1\}^n} \left\{ \boldsymbol{c}^\top \boldsymbol{x} : \quad \boldsymbol{a}_j^\top \boldsymbol{x} + ||D_j^\top \boldsymbol{x} - \boldsymbol{h}_j||_2 \leq b_j, \quad \forall j \in \{1,...,m\} \right\}, \qquad \text{(SP)}$$

where $||\cdot||_2$ is the Euclidean norm and, for each j, $\boldsymbol{a}_j, \boldsymbol{h}_j$, and D_j are real vectors and matrices of appropriate dimension. SOC inequalities arise in many applications, including network design [3], assortment [53], overcommitment [26], and chance-constrained stochastic problems [49,40]. Moreover, SOCPs are supported by commercial solvers such as `CPLEX` [35] and Gurobi [31] which facilitate the evaluation with state-of-the-art techniques. Our methodology, thus, also aims at contributing to the active research in linearization/lifting methods [56,57] as well as cutting methods [5,6,4,18,42] in this area.

The general SOC inequalities can be rewritten as:

$$\boldsymbol{a}^\top \boldsymbol{x} + ||D^\top \boldsymbol{x} - \boldsymbol{h}||_2 \leq b \quad \Leftrightarrow \quad \boldsymbol{a}^\top \boldsymbol{x} + \sqrt{\sum_{k\in\{1,...,l\}} (\boldsymbol{d}_k^\top \boldsymbol{x} - h_k)^2} \leq b. \qquad (14)$$

We propose a novel BDD encoding for (14) using a recursive reformulation based on the methodology by Bergman and Cire (2018) [14]. Specifically, our formulation considers each linear component of (14) independently and applies a variant of the convex composition operator [14] to create relaxed BDDs, which is extended to handle multiple linear terms within a convex function and is tailored to SOCs. We present our recursive model below and refer to Appendix A for details of the BDD relaxation.

Our recursive formulation considers $l+1$ sets of state variables, $\boldsymbol{Q}_0, \boldsymbol{Q}_1, ..., \boldsymbol{Q}_l$, where each set of variables has $n+1$ stages, i.e., $\boldsymbol{Q}_k \in \mathbb{R}^{n+1}$ for each $k \in \{0,1,...,l\}$. State variables $\boldsymbol{Q}_0$ represent the value of the linear term (i.e., $\boldsymbol{a}^\top \boldsymbol{x}$), while $\boldsymbol{Q}_k$ encodes the k-th linear expression in the quadratic term (i.e., $\boldsymbol{d}_k^\top \boldsymbol{x} - h_k$). The recursive model for (14) is given by

$$\begin{aligned}
\text{RSOC} := \big\{ (\boldsymbol{x}; \boldsymbol{Q}) \in \{0,1\}^n \times \mathbb{R}^{(l+1)\times(n+1)} : \\
Q_{0,0} = 0, \; Q_{k,0} = h_k, & \qquad \forall k \in \{1,...,l\}, \quad (15a) \\
Q_{0,i} = Q_{0,i-1} + a_i x_i, & \qquad \forall i \in I, \quad (15b) \\
Q_{k,i} = Q_{k,i-1} + d_{ki} x_i, & \qquad \forall i \in I, \; k \in \{1,...,l\}, \quad (15c) \\
Q_{0,n} + \sqrt{\sum_{k\in\{1,...,l\}} (Q_{k,n})^2} \leq b & \qquad \big\}. \quad (15d)
\end{aligned}$$

The first set of equalities (15a) initialize the state variables at stage 0. Equalities (15b) and (15c) correspond to the recursive formulas for each linear expression, and constraint (15d) enforces the SOC inequality. Notice that $\text{Proj}_x(RSOC)$ is equivalent to the feasible set of the SOC inequality (14).

An exact BDD $\mathcal{B} = (\mathcal{N}, \mathcal{A})$ is the reduced state-transition graph of a dynamic program, where each BDD node maps to a state variable and each arc represents a state transition. If we consider, for example, the RSOC model, the root node $\mathbf{r}$ stores the stage-0 values, and each node u in layer $i \in I$ corresponds to a reachable state from the $(i-1)$-th stage. The recursions (15b) and (15c) are used to compute the transitions between nodes.

The proposed recursive model (and thereby the BDD) can be used for any type of SOC inequality. For our numerical evaluation, we consider two classes of SOC inequalities commonly found in the literature. The first is defined by SOC knapsack inequalities [5,36], i.e., where $D \in \mathbb{R}^{n\times n}$ is a diagonal matrix and the SOC constraint is given by

$$\boldsymbol{a}^\top \boldsymbol{x} + \Omega \sqrt{\sum_{i\in I} d_{ii}^2 x_i} \leq b. \tag{16}$$

We develop a simpler recursive model for (16) with only two sets of state variables, $\boldsymbol{Q}_0$ and $\boldsymbol{Q}_1$. As before, $\boldsymbol{Q}_0$ represents the linear term and $\boldsymbol{Q}_1$ encodes the linear term inside the square root. Thus, the recursive model is given by

$$\text{RSOC-K} := \Big\{(\boldsymbol{x};\boldsymbol{Q}) \in \{0,1\}^n \times \mathbb{R}^{2\times(n+1)} : \ (15a), (15b), \\ Q_{1,i} = Q_{1,i-1} + d_{ii}^2 x_i, \ \ \forall i \in I, \quad Q_{0,n} + \Omega \sqrt{\sum_{i\in I} Q_{1,n}} \leq b \quad \Big\}.$$

For our second class, we consider SOC inequalities derived from chance constraints of the form $\mathbb{P}(\boldsymbol{\xi}^\top \boldsymbol{x} \leq b) \geq \epsilon$ where $\boldsymbol{\xi}$ is a random variable with normal distribution $N(\boldsymbol{a}, D)$ and $\epsilon \in [0.5, 1]$ [49,40]. Specifically, the constraint can be reformulated as

$$\boldsymbol{a}^\top \boldsymbol{x} + \Phi^{-1}(\epsilon)||D\boldsymbol{x}||_2 \leq b \quad \Leftrightarrow \quad \boldsymbol{a}^\top \boldsymbol{x} + \Omega \sqrt{\sum_{k\in\{1,\ldots,n\}} (\boldsymbol{d}_k^\top \boldsymbol{x})^2} \leq b, \tag{17}$$

where we use $\Omega = \Phi^{-1}(\epsilon)$ for simplicity. Notice that (17) is a special case of (14) where D is square matrix and $\boldsymbol{h} = \mathbf{0}$.

An exact BDD for both problem classes can grow exponentially large. Therefore, we construct relaxed BDDs using a standard incremental refinement procedure [16]. For completeness, we provide a detailed explanation of our BDD construction procedure in Appendix A.

7 Empirical Evaluation and Discussion

This section presents an empirical evaluation of our combinatorial cut-and-lift procedure for SP (see §6). We create a BDD for each of the m SOC inequalities and apply our procedure for each such constraint at the root node of the branch-and-bound tree. For any fractional point $\boldsymbol{x} \in [0,1]^n$, we iterate over each BDD until one of them generates a cut, as we describe in detail below.

We then lift the inequality using Algorithm 1. The procedure ends when $\boldsymbol{x}$ cannot be cut-off by any BDD.

Datasets. We test our approach on the SOC knapsack (SOC-K) benchmark [5,36] composed of 90 instances with $n \in \{100, 125, 150\}$ variables and $m \in \{10, 20\}$ constraints. We also generate a random set of instances (SOC-CC) for the more general SOC inequalities derived from chance constraints (i.e., inequality (17)) following a similar procedure to the one used for SOC-K. We consider $n \in \{75, 100, 125\}$, $m \in \{10, 20\}$, $\Omega \in \{1, 3, 5\}$, and a density of $2/\sqrt{n}$ over all the constraints. Parameters $\boldsymbol{a}_j$, D_j, and $\boldsymbol{c}$ are sampled from a discrete uniform distribution with $\boldsymbol{a}_j \in [-50, 50]^n$, $D_j \in [-20, 20]^{n \times n}$, and $\boldsymbol{c} \in [0, 100]^n$. Parameters b_j are given by

$$b_j = t \cdot \left(\sum_{i \in I} a_{ji}^+ + \Omega \sqrt{\sum_{i \in I} \max \left\{ \sum_{k \in I} d_{jik}^+, \sum_{k \in I} d_{jik}^- \right\}^2} \right), \qquad \forall j \in \{1, ..., m\},$$

where $t \in \{0.1, 0.2, 0.3\}$ is the constraint tightness, $f^+ := \max\{0, f\}$, and $f^- := \max\{0, -f\}$ for any $f \in \mathbb{R}$. Note that b_j with $t = 0.3$ will remove approximately 50% of the possible assignments for $\boldsymbol{x} \in \{0, 1\}^n$. Then, we generate 5 random instances for each parameter combination, i.e., 270 instances.

Benchmarks. We implement four basic variants of our approach to assess the BDD cuts and the lifting procedure. `B-Flow` computes the low-complexity combinatorial flow cuts in §5.3, while `B-Gen` also compute these cuts and, if the approach fails to produce any cut, it applies BDD flow cuts derived from our proposed CGPL in §5.2. The two other variants, `B-Gen+L` and `B-Flow+L` are the respective versions of these cutting algorithms augmented with the proposed BDD lifting from §4 for every new constraint generated.

Moreover, we implement the two most recent BDD-based cuts, namely target cuts (`B-Target`) [55] and projected cuts (`B-Proj`) [27]. As before, we use the suffix `+L` to denote if we apply lifting. Lastly, we implement the cover cuts and lifting procedure by Atamtürk and Narayanan (2009) [5] for the SOC-K dataset, here denoted by `Cover` and `CoverLift` for the version without lifting and with their lifting. We also test their cover cuts in conjunction with our BDD lifting, `Cover+L`. Finally, we evaluate combinations of different cut classes, which we will define in each relevant section.

The procedures are implemented in `C++` in the IBM ILOG CPLEX 12.9 solver [35] using the `UserCuts` callback at the root node of the search.[1] All experiments consider a single thread, a one-hour time limit, and the linearization strategy (i.e., `MIQCPStrat = 2`) to solve the SOC problems.[2] We deactivate all solver cuts when running the BDD techniques (i.e., `B-Flow`, `B-Gen`, `B-Target`, `B-Proj`) and the cover-cut variants (i.e., `Cover`, `CoverLift`, and `Cover+L`) to evaluate their effectiveness on their own. Notice that, given the `UserCuts` callback, our techniques omit the `Presolve` option and use `TraditionalSearch`.

[1] The code is available at https://github.com/MargaritaCastro/dd-cut-and-lift.

[2] Numerical testing suggested that this was the best strategy across all techniques.

Table 1 Aggregated root information for the SOC-K and SOC-CC datasets.

		Root Gap		Root Time (s)		# Cuts		% Lift	Sep.(s)
		+nL	+L	+nL	+L	+nL	+L		
SOC-K	CPLEX	2.8%	-	2.4	-	123.5	-	-	-
	Cover	3.3%	2.8%	**1.3**	1.8	96.4	95.7	98.6%	**0.001**
	Cover+L	-	2.7%	-	11.6	-	81.0	70.0%	**0.001**
	B-Flow	3.6%	2.8%	20.0	12.8	240.7	54.9	99.2%	**0.001**
	B-Gen	**1.3%**	**1.3%**	400.4	109.4	1087.2	191.1	80.7%	0.442
	B-Proj	3.9%	3.76%	16.6	16.5	8.0	6.5	72.2%	0.020
	B-Target	**1.3%**	-	118.0	-	108.1	-	-	0.329
SOC-CC	CPLEX	20.4%	-	**9.1**	-	127.6	-	-	-
	B-Flow	19.5%	15.4%	17.1	16.1	31.1	23.3	90.7%	**0.001**
	B-Gen	**13.0%**	**13.0%**	144.7	85.0	305.7	124.5	72.6%	0.096
	B-Proj	17.6%	16.3%	25.3	18.0	61.3	20.5	71.4%	0.017
	B-Target	13.3%	-	71.4	-	98.1	-	-	0.102

For a fair comparison, when running unaugmented CPLEX, we use the same configuration except with the solver cuts activated.[3]

We experimented with three BDD widths $\mathcal{W} \in \{2000, 3000, 4000\}$, i.e., the maximum number of nodes per layer allowed in a relaxed BDD. We present results for the width with the best overall performance, $\mathcal{W} = 4000$ (we include aggregated results for other widths in Appendix B). Notice that most of the created BDDs are relaxed due to width limit, especially when $n \geq 100$.

7.1 Effectiveness of BDD-based Lifting Procedure

We first evaluate our lifting algorithm over the BDD-based cuts and other cutting-plane procedures. Table 1 shows the average root node information for our two datasets, i.e., optimality gap, time, number of cuts, percentage of inequalities lifted at least once, and time to solve the separation problem. The first three columns are divided into two sub-columns, the first showing results without lifting (+nL) and the second with lifting (+L). Column +L refers to our BDD-based lifting, except in row Cover that shows the continuous lifting [5]. Appendices C and D present detailed results for each dataset.

Our lifting procedure significantly reduces the root gap and time for B-Flow. We observe a similar behavior for B-Proj but with a smaller impact due to fewer added cuts. B-Gen also benefits from our lifting, reducing the number of cuts added and, as a consequence, the root node time. However, there is no root gap improvement for B-Gen+L since the general BDD cuts separate all infeasible fractional points from each BDD. Lastly, our lifting has no impact on B-Target since these cuts are facet-defining and, thus, we omit this variant.

[3] Numerical testing suggested that CPLEX performs better on the problem sets when Presolve is *deactivated*.

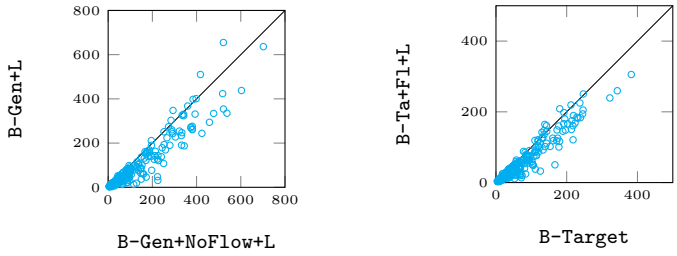

Fig. 2 Root time influence (in seconds) of combinatorial BDD flow cuts for SOC-CC instances.

Our BDD lifting also has a positive impact when applied to cuts obtained from other techniques. For the cover cut `Cover`, our lifting achieves a smaller root gap than the continuous lifting (i.e., 2.67% vs. 2.81%) and adds fewer valid inequalities. Lastly, our BDD lifting is faster than the continuous lifting (0.001 seconds vs. 0.006 seconds per individual cut). This is as expected given that the continuous cuts require solving a linear program.

7.2 BDD-based Cutting Planes Comparison

Table 1 also highlights the root performance differences between the BDD cuts. As expected, the complete methods based on a CGLP (i.e., `B-Gen`, `B-Gen+L`, and `B-Target`) achieve the lowest root gap. However, these techniques are computationally expensive since they solve LPs with as many variables as arcs in the BDD. In total time, `B-Target` is more efficient than `B-Gen` because it adds fewer cuts, possibly due to the fact that its inequalities are facet-defining. This difference is partially mitigated by our BDD lifting since `B-Gen+L` has similar average performance to `B-Target`.

The low-complexity BDD cuts (i.e., `B-Flow` and `B-Proj`) are orders of magnitude faster but have a larger root gap than the CGLP-based alternatives. As discussed in §5.3, `B-Flow` and `B-Flow+L` solve a min-cut problem over the BDD, which explains the fast separation time, but might not remove all infeasible fractional points. `B-Proj` is a complete algorithm but its subgradient routine struggles to separate points close to the convex hull [27], which explains its poor root gaps. Also, `B-Proj` is 15 to 20 times slower than `B-Flow` on average because it requires several subgradient iterations to derive an inequality.

Lastly, we evaluate the impact of combining the low-complexity BDD cuts with the CGLP variants; i.e., the CGLP is invoked after no more fast cuts can be added. Figure 2 depicts two plots where each (x, y) point represents the root time for an instance given by the x-axis and the y-axis techniques. `B-Ta+Fl+L` employs target and combinatorial flow cuts together, while `B-Gen+NoFlow+L` represents a pure general BDD cut approach. In both cases, all methods have approximately the same root gap. We observe that adding the low-complexity cuts improves the root time performance of the methods, as they decrease the number of calls to the corresponding CGLP.

Table 2 Aggregated results showing the overall performance of each technique.

	SOC-K Dataset				SOC-CC Dataset			
	# Solv	Gap	Time	# Nodes	# Solv	Gap	Time	# Nodes
`CPLEX`	70	0.39%	167.8	155,856.4	137	7.46%	530.2	169,802.5
`Cover`	71	0.41%	209.9	446,171.5	-	-	-	-
`CoverLift`	70	0.34%	149.3	303,020.2	-	-	-	-
`Cover+L`	70	0.27%	109.6	191,505.4	-	-	-	-
`B-Flow`	64	0.52%	458.2	867,798.4	135	7.01%	367.4	214,629.4
`B-Flow+L`	73	0.27%	139.6	270,937.6	149	5.55%	240.3	121,800.3
`B-Gen`	76	0.16%	441.1	38,879.7	162	4.31%	172.4	56,490.3
`B-Gen+L`	88	**0.01%**	126.9	32,289.6	167	4.18%	142.0	54,563.6
`B-Proj`	68	0.47%	422.7	1,019,999.4	136	6.63%	380.0	208,567.2
`B-Proj+L`	66	0.50%	335.1	846,437.0	139	6.34%	366.5	202,853.6
`B-Target`	**90**	**0.01%**	131.2	28,031.5	**172**	**4.04%**	132.9	51,319.9
`B-Ta+Fl+L`	**90**	**0.01%**	**83.5**	**24,685.9**	168	4.19%	**110.0**	**45,979.1**

7.3 Solution Performance - Cuts at the Root Node

We now present in Table 2 the average solution performance of the tested techniques when cuts are added only at the root node. The first two columns present the number of instances solved to optimality and the average final gap. The next columns show the average solution time and nodes explored for only the instances that all techniques solved.

We observe that `B-Gen+L`, `B-Target`, and `B-Ta+Fl+L` are the best performing techniques, the latter two solving all instances in SOC-K. These algorithms have small performance differences that are explained by the root information in Table 1, where `B-Target` is slightly more efficient than `B-Gen+L`. In terms of time, `B-Ta+Fl+L` is the fastest approach and explores the fewest number of nodes. We note, however, that `B-Target` is effective for SOC-CC and solves more instances in that benchmark, albeit more slowly than `B-Ta+Fl+L`. This behavior may be explained by the fact that `B-Target` only adds facet-defining cuts while `B-Ta+Fl+L` generates our combinatorial BDD cuts first, which significantly increases the total number of cuts added at the root node.

Figure 3 depicts the performance of each algorithm for the SOC-CC instances (similar results can be found for SOC-K in Appendix C). The graph illustrates the number of instances solved over time (left-hand side) and the accumulated number of instances over a final gap range (right-hand side). Once again, we see the positive impact of our lifting procedure and the dominance of `B-Gen+L`, `B-Target`, and `B-Ta+Fl+L`.

7.4 Solution Performance - Branch-and-Cut

Lastly, we evaluate the effectiveness of the BDD cuts when added in each node of the branch-and-bound tree search. We compare the best CGLP-based approach (i.e., `B-Target`) with our best low-budget alternative (i.e., `B-Flow+L`).

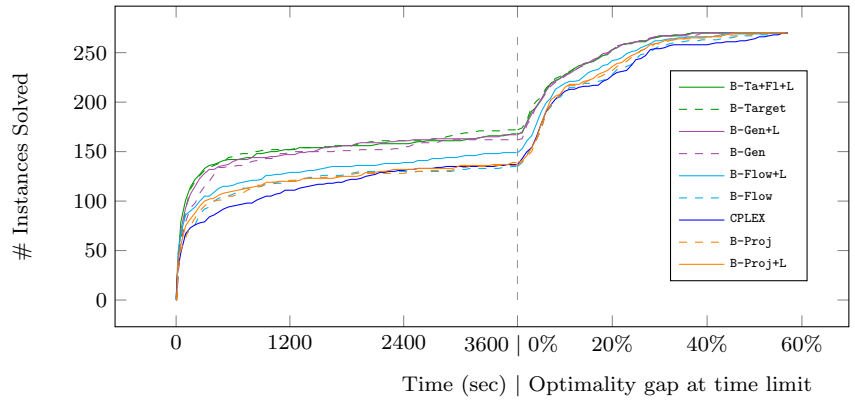

Fig. 3 Profile plot comparing the cumulative number of instances solved over time (left), and the cumulative number of instances over a final gap range (right) for SOC-CC dataset.

We also propose a hybrid `B-Hybrid` that adds BDD target cuts at the root and combinatorial BDD flow cuts during search. Table 3 presents the average results for our two datasets. The table shows the number of instances solved, final gap, average solution time, and nodes explored for instances solved by all techniques, in addition to the total number of cuts added. For ease of comparison, we include again the results when adding cuts only at the root node in row `Root`, while the new version is included in row `Search`.

`B-Flow+L` has a stronger performance when we add these cuts during search than just at the root node. In fact, `B-Flow+L` outperform `B-Target` over the SOC-CC dataset when we add the latter cuts during search (i.e., 177 vs. 172 instances solved). In contrast, `B-Target` performs best when the cuts are only added at the root node due to their long separation time (see Table 1).

Table 3 indicates that, for our datasets, `B-Target` has the best overall performance when adding cuts only at the root node, but performs poorly

Table 3 Aggregated results when adding cuts during tree search.

			# Solve	Gap	Time (s)	# Nodes	# Cuts
SOC-K	`B-Flow+L`	`Root`	73	0.3%	25.8	56,376.5	54.9
		`Search`	78	0.1%	51.5	8,355.5	2,918.4
	`B-Target`	`Root`	**90**	**0.0%**	**38.6**	10,949.5	108.1
		`Search`	47	0.7%	3516.0	**3,553.0**	700.5
	`B-Hybrid`		85	**0.0%**	57.0	3,900.0	1,866.7
SOC-CC	`B-Flow+L`	`Root`	149	5.6%	176.1	54,328.5	23.3
		`Search`	177	3.6%	**21.4**	2,046.8	1,317.2
	`B-Target`	`Root`	172	3.9%	54.2	4,118.2	98.1
		`Search`	102	6.7%	704.9	**1,524.7**	1,249.7
	`B-Hybrid`		**179**	**3.5%**	52.2	1,751.6	1,326.9

when the cuts are added at all nodes during search. In contrast, `B-Flow+L` performs best when cuts are added during search. Based on our analysis, this behavior is due to the fast computational time of combinatorial flow cuts, which can be derived in a significant shorter time than `B-Target` (Table 1). `B-Hybrid`, in turn, solves fewer instances than `B-Target` in the SOC-K dataset possibly due to the large number of combinatorial flow cuts added during search. However, the inverse is true for the SOC-CC dataset, where `B-Hybrid` solves 179 instances and `B-Flow+L` (`Search`) 177, while `B-Target` (`Root`) only solves 172. Overall, we highlight that `B-Target` has the best performance when applied at the root node, while `B-Hybrid` performs well when cuts are added during search. Thus, both could be considered as viable options when applying BDD cuts in general binary problems.

8 Conclusions

We introduce a novel lifting and cutting-plane procedure for binary programs that leverage their combinatorial structure via a binary decision diagram (BDD) encoding of their constraints. Our lifting procedure relies on 0-1 disjunctions to rotate valid inequalities and uses a BDD to efficiently compute the disjunctive sub-problems. While our combinatorial lifting can enhance any cutting-plane approach, we also propose two novel BDD-based cut generation algorithms based on an alternative network-flow representation of the BDD.

BDDs give us the flexibility to apply our cut-and-lift approach to a wide range of non-linear problems. As a case study, we tested our procedure over second-order conic inequalities and compare its performance against a state-of-the-art solver (`CPLEX`) and existing BDD cuts in the literature. Overall, our lifting procedure reduced the average root gap up to 29% in our benchmark when applied to cuts generated by all tested methods. Also, a hybrid technique combining our approach with existing BDD cuts proved to be the most efficient over the tested benchmarks.

References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice-Hall, Inc., USA (1993)
2. Andersen, H.R., Hadzic, T., Hooker, J.N., Tiedemann, P.: A constraint store based on multivalued decision diagrams. In: International Conference on Principles and Practice of Constraint Programming–CP 2007, pp. 118–132. Springer (2007)
3. Atamtürk, A., Bhardwaj, A.: Network design with probabilistic capacities. Networks **71**(1), 16–30 (2018)
4. Atamtürk, A., Muller, L.F., Pisinger, D.: Separation and extension of cover inequalities for conic quadratic knapsack constraints with generalized upper bounds. INFORMS Journal on Computing **25**(3), 420–431 (2013)
5. Atamtürk, A., Narayanan, V.: The submodular knapsack polytope. Discrete Optimization **6**(4), 333–344 (2009)
6. Atamtürk, A., Narayanan, V.: Conic mixed-integer rounding cuts. Mathematical programming **122**(1), 1–20 (2010)

7. Balas, E.: Facets of the knapsack polytope. Mathematical programming **8**(1), 146–164 (1975)
8. Balas, E.: Disjunctive Programming. Springer (2018)
9. Balas, E., Ceria, S., Cornuéjols, G.: A lift-and-project cutting plane algorithm for mixed 0–1 programs. Mathematical programming **58**(1-3), 295–324 (1993)
10. Balas, E., Ceria, S., Cornuéjols, G.: Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. Management Science **42**(9), 1229–1246 (1996)
11. Becker, B., Behle, M., Eisenbrand, F., Wimmer, R.: BDDs in a branch and cut framework. In: International Workshop on Experimental and Efficient Algorithms, pp. 452–463. Springer (2005)
12. Behle, M.: Binary decision diagrams and integer programming. Ph.D. Thesis (2007)
13. Bergman, D., Cardonha, C., Mehrani, S.: Binary decision diagrams for bin packing with minimum color fragmentation. In: International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research–CPAIOR 2019, pp. 57–66. Springer (2019)
14. Bergman, D., Cire, A.A.: Discrete nonlinear optimization by state-space decompositions. Management Science **64**(10), 4700–4720 (2018)
15. Bergman, D., Cire, A.A., van Hoeve, W.J., Hooker, J.N.: Variable ordering for the application of BDDs to the maximum independent set problem. In: International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research–CPAIOR 2012, pp. 34–49. Springer (2012)
16. Bergman, D., Cire, A.A., van Hoeve, W.J., Hooker, J.N.: Discrete optimization with decision diagrams. INFORMS Journal on Computing **28**(1), 47–66 (2016)
17. Bergman, D., Lozano, L.: Decision diagram decomposition for quadratically constrained binary optimization. Optimization Online e-prints (2018)
18. Bhardwaj, A.: Binary conic quadratic knapsacks. Ph.D. thesis, UC Berkeley (2015)
19. Bixby, R.E., Fenelon, M., Gu, Z., Rothberg, E., Wunderling, R.: Mixed-integer programming: A progress report. In: The sharpest cut: the impact of Manfred Padberg and his work, pp. 309–325. SIAM (2004)
20. van den Bogaerdt, P., de Weerdt, M.: Multi-machine scheduling lower bounds using decision diagrams. Operations Research Letters **46**(6), 616–621 (2018)
21. van den Bogaerdt, P., de Weerdt, M.: Lower bounds for uniform machine scheduling using decision diagrams. In: International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research–CPAIOR 2019, pp. 565–580. Springer (2019)
22. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. Computers, IEEE Transactions on **100**(8), 677–691 (1986)
23. Castro, M.P., Cire, A.A., Beck, J.C.: An MDD-based lagrangian approach to the multicommodity pickup-and-delivery tsp. INFORMS Journal on Computing (2019)
24. Castro, M.P., Piacentini, C., Cire, A.A., Beck, J.C.: Relaxed BDDs: An admissible heuristic for delete-free planning based on a discrete relaxation. In: Proceedings of the International Conference on Automated Planning and Scheduling, pp. 77–85 (2019)
25. Cire, A.A., van Hoeve, W.J.: Multivalued decision diagrams for sequencing problems. Operations Research **61**(6), 1411–1428 (2013)
26. Cohen, M.C., Keller, P.W., Mirrokni, V., Zadimoghaddam, M.: Overcommitment in cloud services: Bin packing with chance constraints. Management Science **65**(7), 3255–3271 (2019)
27. Davarnia, D., van Hoeve, W.J.: Outer approximation for integer nonlinear programs via decision diagrams. Mathematical Programming (2020)
28. Gomory, R.E.: Some polyhedra related to combinatorial problems. Linear Algebra and its Applications **2**(4), 451 – 558 (1969)
29. Gu, Z., Nemhauser, G.L., Savelsbergh, M.W.: Lifted cover inequalities for 0-1 integer programs: Computation. INFORMS Journal on Computing **10**(4), 427–437 (1998)
30. Gu, Z., Nemhauser, G.L., Savelsbergh, M.W.: Lifted cover inequalities for 0-1 integer programs: Complexity. INFORMS Journal on Computing **11**(1), 117–123 (1999)
31. Gurobi Optimization, L.: Gurobi optimizer reference manual (2020)
32. Hammer, P.L., Johnson, E.L., Peled, U.N.: Facet of regular 0–1 polytopes. Mathematical Programming **8**(1), 179–206 (1975)

33. Hoda, S., Van Hoeve, W.J., Hooker, J.N.: A systematic approach to MDD-based constraint programming. In: International Conference on Principles and Practice of Constraint Programming–CP 2010, pp. 266–280. Springer (2010)
34. Hooker, J.N.: Job sequencing bounds from decision diagrams. In: International Conference on Principles and Practice of Constraint Programming–CP 2017, pp. 565–578. Springer (2017)
35. IBM: ILOG CPLEX Studio 12.9 Manual (2019)
36. Joung, S., Park, S.: Lifting of probabilistic cover inequalities. Operations Research Letters **45**(5), 513–518 (2017)
37. Kılınç-Karzan, F.: On minimal valid inequalities for mixed integer conic programs. Mathematics of Operations Research **41**(2), 477–510 (2016)
38. Kılınç-Karzan, F., Yıldız, S.: Two-term disjunctions on the second-order cone. Mathematical Programming **154**(1-2), 463–491 (2015)
39. Kinable, J., Cire, A.A., van Hoeve, W.J.: Hybrid optimization methods for time-dependent sequencing problems. European Journal of Operational Research **259**(3), 887–897 (2017)
40. Lobo, M.S., Vandenberghe, L., Boyd, S., Lebret, H.: Applications of second-order cone programming. Linear algebra and its applications **284**(1-3), 193–228 (1998)
41. Lodi, A.: Mixed integer programming computation. In: 50 Years of Integer Programming 1958-2008, pp. 619–645. Springer (2010)
42. Lodi, A., Tanneau, M., Vielma, J.P.: Disjunctive cuts for mixed-integer conic optimization. arXiv preprint arXiv:1912.03166 (2019)
43. Louveaux, Q., Wolsey, L.A.: Lifting, superadditivity, mixed integer rounding and single node flow sets revisited. Quarterly Journal of the Belgian, French and Italian Operations Research Societies **1**(3), 173–207 (2003)
44. Lozano, L., Smith, J.C.: A binary decision diagram based algorithm for solving a class of binary two-stage stochastic programs. Mathematical Programming pp. 1–24 (2018)
45. Modaresi, S., Kılınç, M.R., Vielma, J.P.: Split cuts and extended formulations for mixed integer conic quadratic programming. Operations Research Letters **43**(1), 10–15 (2015)
46. Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. Wiley-Interscience, USA (1988)
47. Padberg, M.W.: On the facial structure of set packing polyhedra. Mathematical programming **5**(1), 199–215 (1973)
48. Padberg, M.W.: A note on zero-one programming. Operations Research **23**(4), 833–837 (1975)
49. Van de Panne, C., Popp, W.: Minimum-cost cattle feed under probabilistic protein constraints. Management Science **9**(3), 405–430 (1963)
50. Perregaard, M., Balas, E.: Generating cuts from multiple-term disjunctions. In: International Conference on Integer Programming and Combinatorial Optimization, pp. 348–360. Springer (2001)
51. Raghunathan, A.U., Bergman, D., Hooker, J.N., Serra, T., Kobori, S.: Seamless multimodal transportation scheduling. arXiv preprint arXiv:1807.09676 (2018)
52. Santana, A., Dey, S.S.: Some cut-generating functions for second-order conic sets. Discrete Optimization **24**, 51–65 (2017)
53. Şen, A., Atamtürk, A., Kaminsky, P.: A conic integer optimization approach to the constrained assortment problem under the mixed multinomial logit model. Operations Research **66**(4), 994–1003 (2018)
54. Stubbs, R.A., Mehrotra, S.: A branch-and-cut method for 0-1 mixed convex programming. Mathematical programming **86**(3), 515–532 (1999)
55. Tjandraatmadja, C., van Hoeve, W.J.: Target cuts from relaxed decision diagrams. INFORMS Journal on Computing **31**(2), 285–301 (2019)
56. Vielma, J.P., Ahmed, S., Nemhauser, G.L.: A lifted linear programming branch-and-bound algorithm for mixed-integer conic quadratic programs. INFORMS Journal on Computing **20**(3), 438–450 (2008)
57. Vielma, J.P., Dunning, I., Huchette, J., Lubin, M.: Extended formulations in mixed integer conic quadratic programming. Mathematical Programming Computation **9**(3), 369–418 (2017)
58. Wolsey, L.A.: Technical notefacets and strong valid inequalities for integer programs. Operations Research **24**(2), 367–372 (1976). DOI 10.1287/opre.24.2.367

59. Wolsey, L.A., Nemhauser, G.L.: Integer and combinatorial optimization, vol. 55. John Wiley & Sons (1999)
60. Zemel, E.: Easily computable facets of the knapsack polytope. Mathematics of Operations Research **14**(4), 760–764 (1989)

A Relaxed BDD Construction Procedure for Second-Order Cones

We now present the relaxed BDD construction procedure for SOC inequalities based on the RSOC model. The construction algorithm is analogous for the case of SOC knapsack constraints and the RSOC-K model.

The state information at the core of our BDD construction algorithm is based on recursive model RSOC. This information is stored in each node of the BDD and used to identify infeasible assignments and decide how to widen the BDD (i.e., split nodes). Our state information keeps track of each of the $l+1$ linear components in inequality (14), i.e., $\boldsymbol{a}^\top \boldsymbol{x}$ and $\boldsymbol{d}_k^\top \boldsymbol{x} - h_k$ for each $k \in \{1, ..., l\}$. Thus, each node $u \in \mathcal{N}_i$ has two $l+1$ dimensional vectors for top-down information, $Q^{\downarrow \mathsf{min}}(u)$ and $Q^{\downarrow \mathsf{max}}(u)$, that approximate the linear components considering the partial assignments from $\mathbf{r}$ to u. We set the top-down state information at the root node as $Q_0^{\downarrow \mathsf{min}}(\mathbf{r}) := Q_0^{\downarrow \mathsf{max}}(\mathbf{r}) := 0$ and $Q_k^{\downarrow \mathsf{min}}(\mathbf{r}) := Q_k^{\downarrow \mathsf{max}}(\mathbf{r}) := -h_k$ for all $k \in \{1, ..., l\}$. Then, for any $u \in \mathcal{N}_i$, $i \in \{2, ..., n\}$, and $k \in \{1, ..., l\}$ we update the states as:

$$Q_0^{\downarrow \mathsf{min}}(u) := \min_{a \in \mathcal{A}^{\mathsf{in}}(u)} \{Q_0^{\downarrow \mathsf{min}}(s(a)) + a_{i-1} \cdot v_a\},$$
$$Q_0^{\downarrow \mathsf{max}}(u) := \max_{a \in \mathcal{A}^{\mathsf{in}}(u)} \{Q_0^{\downarrow \mathsf{max}}(s(a)) + a_{i-1} \cdot v_a\},$$
$$Q_k^{\downarrow \mathsf{min}}(u) := \min_{a \in \mathcal{A}^{\mathsf{in}}(u)} \{Q_k^{\downarrow \mathsf{min}}(s(a)) + d_{ki-1} \cdot v_a\},$$
$$Q_k^{\downarrow \mathsf{max}}(u) := \max_{a \in \mathcal{A}^{\mathsf{in}}(u)} \{Q_k^{\downarrow \mathsf{max}}(s(a)) + d_{ki-1} \cdot v_a\}.$$

Similarly, we use two $l+1$ dimensional vectors, $Q^{\uparrow \mathsf{min}}(u)$ and $Q^{\uparrow \mathsf{max}}(u)$, for our bottom-up state information for each node $u \in \mathcal{N}$. The information is initialized at the terminal node as $Q_0^{\uparrow \mathsf{min}}(\mathbf{t}) := Q_0^{\uparrow \mathsf{max}}(\mathbf{t}) := 0$ and $Q_k^{\uparrow \mathsf{min}}(\mathbf{t}) := Q_k^{\uparrow \mathsf{max}}(\mathbf{t}) := 0$ for all $k \in \{1, ..., l\}$. Then, for any $u \in \mathcal{N}_i$, $i \in \{1, ..., n-1\}$, and $k \in \{1, ..., l\}$, we have:

$$Q_0^{\uparrow \mathsf{min}}(u) := \min_{a \in \mathcal{A}^{\mathsf{out}}(u)} \{Q_0^{\uparrow \mathsf{min}}(t(a)) + a_k \cdot v_a\},$$
$$Q_0^{\uparrow \mathsf{max}}(u) := \max_{a \in \mathcal{A}^{\mathsf{out}}(u)} \{Q_0^{\uparrow \mathsf{max}}(t(a)) + a_k \cdot v_a\},$$
$$Q_k^{\uparrow \mathsf{min}}(u) := \min_{a \in \mathcal{A}^{\mathsf{out}}(u)} \{Q_k^{\uparrow \mathsf{min}}(t(a)) + d_{ki} \cdot v_a\},$$
$$Q_k^{\uparrow \mathsf{max}}(u) := \max_{a \in \mathcal{A}^{\mathsf{out}}(u)} \{Q_k^{\uparrow \mathsf{max}}(t(a)) + d_{ki} \cdot v_a\}.$$

For each node $u \in \mathcal{N}$, the state information under and over approximates the value of the linear components of (14) for all $\mathbf{r} - u$ paths (i.e., top-down information) and for all $u - \mathbf{t}$ paths (i.e., bottom-up information). We use the state information to identify if an arc corresponds to an infeasible assignment, i.e., all paths traversing it correspond to infeasible solutions of (14). In particular, we can remove an arc $a = (u, u') \in \mathcal{A}_i$ if the following condition holds:

$$Q_0^{\downarrow \mathsf{min}}(u) + a_i \cdot v_a + Q_0^{\uparrow \mathsf{min}}(u') + \Omega \sqrt{\sum_{k=1}^{l} g_k(a)} > b, \tag{18}$$

where $g_k(a)$ for $k \in \{1, ..., l\}$ is given by:

$$g_k(a) := \begin{cases} (Q_k^{\downarrow \mathsf{min}}(u) + d_{ki} \cdot v_a + Q_k^{\uparrow \mathsf{min}}(u'))^2, & \text{if } Q_k^{\downarrow \mathsf{min}}(u) + d_{ki} \cdot v_a + Q_k^{\uparrow \mathsf{min}}(u') > 0, \\ (Q_k^{\downarrow \mathsf{max}}(u) + d_{ki} \cdot v_a + Q_k^{\uparrow \mathsf{max}}(u'))^2, & \text{if } Q_k^{\downarrow \mathsf{max}}(u) + d_{ki} \cdot v_a + Q_k^{\uparrow \mathsf{max}}(u') < 0, \\ 0, & \text{otherwise.} \end{cases}$$

Notice that $g_k(a)$ under approximates $(\boldsymbol{d}_k^\top \boldsymbol{x} - d_k)^2$ for all paths traversing arc $a \in \mathcal{A}$, and so the left-hand side (LHS) of (18) under approximates the LHS of (14) for all paths

traversing arc $a \in \mathcal{A}_i$. Then, all paths traversing an arc a that satisfy (18) correspond to invalid assignments for (14).

If $Q_k^{\downarrow\mathsf{max}}(u) = Q_k^{\downarrow\mathsf{min}}(u)$ for all nodes $u \in \mathcal{N}$, we recover the exact BDD based on the recursive model RSOC and condition (18) is equivalent to (15d). Thus, our splitting procedure tries to achieve this property by selecting nodes u with $Q_k^{\downarrow\mathsf{max}}(u) - Q_k^{\downarrow\mathsf{min}}(u) \geq \delta$ ($\delta > 0$) for some $k \in \{0, ..., l\}$ and then split it into two new nodes, u' and u'', so $Q_k^{\downarrow\mathsf{max}}(u') - Q_k^{\downarrow\mathsf{min}}(u') < \delta$ and $Q_k^{\downarrow\mathsf{max}}(u'') - Q_k^{\downarrow\mathsf{min}}(u'') < \delta$. The splitting procedure then duplicates the outgoing arcs of u and assigns them to both u' and u'' to keep the same set of paths in $\mathcal{B}$.

Algorithm 2 Relaxed (Exact) BDD Construction Procedure

```
1:  procedure ConstructBDD(SOC Constraint ⟨a, D, h, Ω, b, n⟩, W)
2:      B := WidthOneBDD(n)
3:      while B has been modified do
4:          UpdateBDDNodesBottom(N)
5:          for i ∈ {1, ..., n} do
6:              UpdateBDDNodesTop(N_i)
7:              SplitBDDNodes(N_i, W)
8:              FilterBDDOutgoingEdges(N_i)
9:          UpdateBDDNodesTop(N_{n+1})
10:     ReduceBDD(B)
11:     return B
```

Our construction procedure creates a relaxed BDD $\mathcal{B} = (\mathcal{N}, \mathcal{A})$ by limiting its width $w(\mathcal{B})$ by a positive value $\mathcal{W}$, where $w(\mathcal{B}) := \max_{i \in I}\{|\mathcal{N}_i|\}$ represents the maximum number of nodes in each layer. The complete BDD construction procedure is shown in Algorithm 2. The algorithm starts creating a width-one BDD for the SOC constraint (line 2) and then updates the bottom-up information for all the nodes (line 4). During the top-down pass through the BDD (lines 5-9), the procedure updates the top-down information of layer $\mathcal{N}_i$, splits the nodes until we reach the width limit $\mathcal{W}$, and filters the emanating arcs of $\mathcal{N}_i$. The algorithm then checks if the BDD has been updated (line 3) and repeats the bottom-up and top-down iterations until the BDD cannot be updated any more. Lastly, we reduce the BDD (line 10) following the standard procedure in the literature [22].

The resulting BDD starts with all possible variable assignments (i.e., a width-one BDD) and removes arcs using condition (18). Thus, the procedure is guaranteed to construct a relaxed BDD for a SOC constraint. Notice that for a big enough $\mathcal{W}$, the procedure will return an exact BDD.

Example 5 Consider the following binary set defined by an SOC inequality $X = \{\boldsymbol{x} \in \{0,1\}^3 : 3x_1 + x_2 + x_3 + \sqrt{(x_1 + x_2 + 2x_3)^2 + (x_1 + 3x_2 - x_3 + 3)^2} \leq 8\}$. Figure 4 depicts some of the steps to construct an exact BDD for X. The left most diagram corresponds to a width-one BDD for this problem. The top-down state information in the root node is $((Q_0^{\downarrow\mathsf{min}}(\mathbf{r}), Q_0^{\downarrow\mathsf{max}}(\mathbf{r})),\ (Q_1^{\downarrow\mathsf{min}}(\mathbf{r}), Q_1^{\downarrow\mathsf{max}}(\mathbf{r})),\ (Q_2^{\downarrow\mathsf{min}}(\mathbf{r}), Q_2^{\downarrow\mathsf{max}}(\mathbf{r}))) = ((0,0),\ (0,0),\ (3,3))$, while for node u_1 is $((0,3),\ (0,1),\ (3,4))$.

The middle BDD illustrates the resulting BDD after splitting node u_1. The resulting nodes, u_1' and u_1'', have top-down state information $((0,0),\ (0,0),\ (3,3))$ and $((3,3),\ (1,1),\ (4,4))$, respectively. In addition, the gray arc from u_1'' to u_2 corresponds to an invalid assignment: the bottom-up information of u_2 is $((0,1),\ (0,2),\ (-1,0))$, thus, (18) evaluates to $10.3 > 8$.

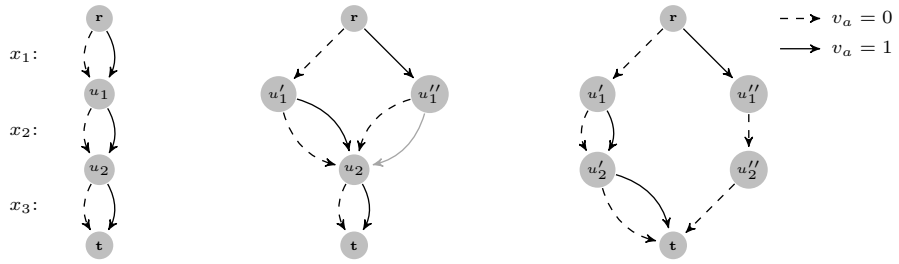

Fig. 4 BDD construction procedure for set X defined in Example 5. The figure depicts a width-one BDD (left), a BDD after the splitting and filtering procedure over $\mathcal{N}_2$ (middle), and the resulting exact reduced BDD (right).

B Experiments Comparing Different BDD Widths

Table 4 presents the average performance for `CPLEX` and our four alternatives (i.e., `B-Flow`, `B-Flow+L`, `B-Gen`, and `B-Gen+L`) with three different maximum width values, $\mathcal{W} \in \{2000, 3000, 4000\}$, over the SOC-CC instances. The table shows the number of instances solved, average root gap, and average final gap for all techniques. Our four alternatives with $\mathcal{W} \in \{2000, 3000, 4000\}$ each outperform `CPLEX`. $\mathcal{W} = 4000$ achieves the best overall performance across the four combinatorial cut-and-lift alternatives. Similarly, $\mathcal{W} = 4000$ achieves the best or comparable performance across the five BDD approaches over the SOC-K instances (right).

Table 4 Average performance of all techniques for different BDD widths for SOC-CC.

	Width	# Solve	Root Gap	Final Gap
`CPLEX`		137	20.82%	7.75%
`B-Flow`	2000	137	20.31%	7.35%
	3000	**140**	20.11%	**7.25%**
	4000	139	**20.01%**	**7.25%**
`B-Flow+L`	2000	149	16.61%	6.09%
	3000	**150**	16.03%	6.04%
	4000	**150**	**15.68%**	**5.89%**
`B-Gen`	2000	160	14.93%	5.21%
	3000	159	14.05%	4.87%
	4000	**166**	**13.47%**	**4.59%**
`B-Gen+L`	2000	160	14.91%	5.00%
	3000	**168**	14.03%	4.66%
	4000	**168**	**13.44%**	**4.43%**

Similarly, Table 5 presents the average performance over the SOC-K instances for `CPLEX`, our four alternatives (i.e., `B-Flow`, `B-Flow+L`, `B-Gen`, and `B-Gen+L`) with three different maximum width values, $\mathcal{W} \in \{2000, 3000, 4000\}$. Overall, $\mathcal{W} = 4000$ achieves the best or comparable performance across the five BDD approaches.

We note that these three BDD widths achieve competitive results with respect to `CPLEX` in our dataset. However, problems with more variables (i.e., $n > 125$) would probably required a larger width to create a tight BDD relaxation and, thus, strong cuts.

Table 5 Average performance of all techniques for different BDD widths for SOC-K.

	Width	# Solve	Root Gap	Final Gap
`CPLEX`		70	2.84%	0.39%
`B-Flow`	2000	66	3.64%	**0.52%**
	3000	**67**	3.63%	**0.52%**
	4000	64	**3.63%**	0.53%
`B-Flow+L`	2000	74	2.84%	0.27%
	3000	72	**2.80%**	0.28%
	4000	**74**	**2.80%**	**0.26%**
`B-Gen`	2000	75	1.62%	0.16%
	3000	**78**	1.45%	**0.15%**
	4000	76	**1.34%**	0.16%
`B-Gen+L`	2000	83	1.61%	0.04%
	3000	87	1.44%	0.02%
	4000	**88**	**1.33%**	**0.01%**

C Average Performance Comparison for Knapsack Chance Constraints

We now present additional results for the SOC-K dataset. As in Figure 3, Figure 5 illustrate the performance of each algorithm for the SOC-K dataset. We see a clear dominance of `B-Gen+L`, `B-Target`, and `B-Ta+Fl+L` and also the positive impact of our combinatorial lifting in instances solved and gap reduction.

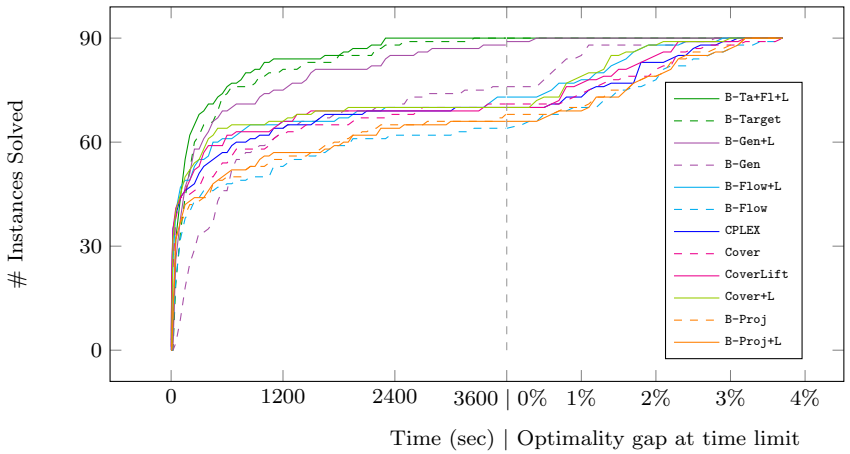

Fig. 5 Profile plot comparing the accumulated number of instances solved over time (left), and the accumulated number of instances over a final gap range (right) for SOC-K dataset.

The following tables show average results for each parameter configuration over the SOC-CC benchmark. We present results for our four variants (i.e., `B-Flow`, `B-Flow+L`, `B-Gen`, and `B-Gen+L`), cover cut variants (i.e., `Cover`, `CoverLift`, and `Cover+L`), `CPLEX`, and best performing BDD-based cuts (i.e., `B-Target` and `B-Ta+Fl+L`). All techniques add cuts only

Table 6 Instances solved to optimality comparison across all techniques for SOC-K benchmark.

			# Instances Solved									
n	m	Ω	CPLEX	Cover	CoverLift	Cover+L	B-Flow	B-Flow+L	B-Gen	B-Gen+L	B-Target	B-Ta+Fl+L
100	10	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
		3	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
		5	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	20	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
		3	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
		5	**5**	**5**	**5**	**5**	3	**5**	**5**	**5**	**5**	**5**
125	10	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
		3	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
		5	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	20	1	**5**	**5**	**5**	**5**	4	**5**	**5**	**5**	**5**	**5**
		3	1	2	1	1	1	3	4	**5**	**5**	**5**
		5	0	0	0	0	0	1	2	**5**	**5**	**5**
150	10	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
		3	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
		5	**5**	**5**	**5**	**5**	3	**5**	**5**	**5**	**5**	**5**
	20	1	4	4	4	4	3	4	4	**5**	**5**	**5**
		3	0	0	0	0	0	0	1	3	**5**	**5**
		5	0	0	0	0	0	0	0	**5**	**5**	**5**
Total			70	71	70	70	64	73	76	88	**90**	**90**

at the root node of the tree search. Tables 6, 7, and 8 show the number of instances solved, average root gap, and average final gap for each n, m, and Ω combination with $\mathcal{W} = 4000$, respectively. Similarly, Tables 9 and 10 show the average number of nodes in the branch-and-bound search and the average run time for the instances that all techniques solved to optimality.

Table 7 Root Gap comparison across all techniques for SOC-K benchmark.

			Root Gap (%)									
n	m	Ω	CPLEX	Cover	CoverLift	Cover+L	B-Flow	B-Flow+L	B-Gen	B-Gen+L	B-Target	B-Ta+Fl+L
100	10	1	0.9%	1.8%	1.4%	1.4%	2.0%	1.4%	**0.5%**	**0.5%**	**0.5%**	**0.5%**
		3	2.4%	2.9%	2.2%	1.9%	3.2%	2.0%	0.5%	**0.4%**	0.5%	**0.4%**
		5	3.7%	3.9%	2.8%	2.2%	4.4%	2.6%	**0.3%**	**0.3%**	**0.3%**	**0.3%**
	20	1	1.9%	2.8%	2.4%	2.3%	3.2%	2.5%	**1.1%**	**1.1%**	**1.1%**	**1.1%**
		3	4.5%	5.0%	4.5%	4.4%	5.4%	4.3%	**1.5%**	**1.5%**	**1.5%**	**1.5%**
		5	6.1%	6.3%	5.9%	5.5%	7.1%	5.4%	**1.5%**	**1.5%**	**1.5%**	**1.5%**
125	10	1	**0.7%**	1.4%	0.9%	1.0%	1.8%	1.2%	**0.7%**	**0.7%**	**0.7%**	**0.7%**
		3	1.8%	2.2%	1.6%	1.5%	2.5%	1.6%	**0.8%**	**0.8%**	**0.8%**	**0.8%**
		5	2.7%	2.8%	2.0%	1.7%	3.3%	1.8%	**0.9%**	**0.9%**	**0.9%**	**0.9%**
	20	1	2.1%	2.9%	2.6%	2.5%	3.1%	2.7%	**1.9%**	**1.9%**	**1.9%**	**1.9%**
		3	4.0%	4.3%	4.1%	4.1%	4.7%	4.1%	**2.4%**	**2.4%**	**2.4%**	**2.4%**
		5	5.8%	6.0%	5.6%	5.5%	6.5%	5.6%	**2.7%**	**2.7%**	**2.7%**	**2.7%**
150	10	1	**0.5%**	1.1%	0.9%	0.9%	1.3%	1.1%	0.7%	0.7%	0.7%	0.7%
		3	1.6%	1.9%	1.5%	1.4%	2.1%	1.5%	**0.9%**	**0.9%**	**0.9%**	**0.9%**
		5	2.7%	2.8%	2.4%	2.3%	3.2%	2.6%	**1.3%**	**1.3%**	**1.3%**	**1.3%**
	20	1	1.6%	2.3%	1.9%	1.9%	2.5%	2.2%	**1.6%**	**1.6%**	**1.6%**	**1.6%**
		3	3.3%	3.7%	3.3%	3.1%	3.9%	3.3%	**2.1%**	**2.1%**	**2.1%**	**2.1%**
		5	4.7%	5.1%	4.5%	4.3%	5.3%	4.4%	2.6%	**2.5%**	**2.5%**	**2.5%**
Average			2.8%	3.3%	2.8%	2.7%	3.6%	2.8%	**1.3%**	**1.3%**	**1.3%**	**1.3%**

Table 8 Final Gap comparison across all techniques for SOC-K benchmark.

			Final Gap (%)									
n	m	Ω	CPLEX	Cover	CoverLift	Cover+L	B-Flow	B-Flow+L	B-Gen	B-Gen+L	B-Target	B-Ta+Fl+L
100	10	1	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
		3	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
		5	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	20	1	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
		3	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
		5	**0.0%**	**0.0%**	**0.0%**	**0.0%**	0.4%	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
125	10	1	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
		3	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
		5	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	20	1	**0.0%**	**0.0%**	**0.0%**	**0.0%**	0.1%	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
		3	1.0%	0.9%	0.8%	0.8%	1.2%	0.4%	0.1%	**0.0%**	**0.0%**	**0.0%**
		5	2.5%	2.5%	2.0%	1.6%	2.8%	1.6%	0.5%	**0.0%**	**0.0%**	**0.0%**
150	10	1	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
		3	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
		5	**0.0%**	**0.0%**	**0.0%**	**0.0%**	0.2%	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	20	1	0.1%	0.1%	0.2%	0.1%	0.3%	0.1%	0.2%	**0.0%**	**0.0%**	**0.0%**
		3	1.3%	1.3%	1.0%	0.7%	1.6%	0.9%	0.6%	0.1%	**0.0%**	**0.0%**
		5	2.1%	2.3%	1.8%	1.6%	2.7%	1.6%	1.5%	**0.0%**	**0.0%**	**0.0%**
Average			0.4%	0.4%	0.3%	0.3%	0.5%	0.3%	0.2%	**0.0%**	**0.0%**	**0.0%**

Table 9 Nodes explored comparison across all techniques for SOC-K benchmark.

			# Nodes Explored									
n	m	Ω	CPLEX	Cover	CoverLift	Cover+L	B-Flow	B-Flow+L	B-Gen	B-Gen+L	B-Target	B-Ta+Fl+L
100	10	1	2,187	9,493	3,407	3,808	13,279	5,055	191	170	173	**159**
		3	15,911	38,018	18,616	13,374	77,135	11,973	127	100	**92**	167
		5	56,024	73,318	34,888	11,065	197,240	24,390	182	171	214	**165**
	20	1	12,093	84,852	36,813	37,479	152,371	56,573	1,929	1,128	1,088	**1,015**
		3	275,046	498,671	367,255	326,495	1,061,138	271,931	3,062	1,465	1,591	**1,181**
		5	796,298	1,030,628	1,073,455	431,804	1,648,577	655,462	1,461	607	578	**559**
125	10	1	**2,557**	26,109	5,254	6,707	59,136	22,296	3,864	4,875	2,747	3,476
		3	16,810	89,336	35,721	28,278	190,740	39,481	8,114	**4,436**	4,720	4,751
		5	157,561	301,041	253,128	87,198	1,290,400	89,930	6,137	4,415	3,665	**3,521**
	20	1	243,540	1,134,038	628,234	584,492	2,016,914	568,773	120,849	90,177	83,596	**68,425**
		3	445,816	902,386	476,674	513,874	2,206,133	1,047,598	86,749	30,173	**24,841**	26,485
		5	-	-	-	-	-	-	-	-	-	-
150	10	1	**1,835**	39,165	14,878	18,218	61,289	33,738	17,673	8,190	16,377	10,179
		3	267,752	1,156,661	1,084,202	400,829	1,900,415	399,942	56,718	36,842	**29,805**	31,726
		5	394,755	877,686	423,235	337,258	2,718,140	658,492	88,738	111,968	**63,792**	64,269
	20	1	453,059	2,707,252	1,202,011	1,186,773	3,237,491	2,514,477	541,442	482,927	442,424	**377,666**
		3	-	-	-	-	-	-	-	-	-	-
		5	-	-	-	-	-	-	-	-	-	-
Average			209,416	597,910	377,185	265,843	1,122,027	426,674	62,482	51,843	45,047	**39,583**

Table 10 Solving time comparison across all techniques for SOC-K benchmark.

			Solving Time (sec)									
n	m	Ω	CPLEX	Cover	CoverLift	Cover+L	B-Flow	B-Flow+L	B-Gen	B-Gen+L	B-Target	B-Ta+Fl+L
100	10	1	3.3	2.7	**1.4**	7.2	13.5	7.6	138.5	31.5	37.2	24.4
		3	9.4	12.0	**6.5**	9.8	46.2	9.6	170.2	48.8	61.1	36.4
		5	32.3	24.6	13.0	**9.0**	97.2	13.1	391.8	55.2	72.5	33.0
	20	1	19.1	38.5	**18.8**	30.0	121.3	39.7	457.6	149.4	135.5	84.2
		3	317.8	273.7	217.1	190.3	762.6	**162.8**	753.6	260.4	235.7	179.6
		5	970.4	718.5	744.6	331.2	1330.4	473.2	1232.0	374.7	373.7	**220.1**
125	10	1	3.3	6.8	**2.4**	12.1	36.2	16.1	100.3	29.1	44.2	27.0
		3	11.8	23.7	**12.7**	18.4	80.6	20.9	415.5	67.2	88.9	46.6
		5	105.5	101.0	83.4	41.9	400.6	**39.6**	826.3	80.9	113.5	55.3
	20	1	357.6	567.0	322.3	323.5	1031.0	279.5	409.6	227.1	220.0	**146.7**
		3	645.0	668.3	338.0	425.5	1716.2	652.4	946.3	518.6	523.2	**275.1**
		5	-	-	-	-	-	-	-	-	-	-
150	10	1	2.9	10.0	**5.2**	18.6	36.1	22.8	85.2	32.1	41.7	29.1
		3	183.1	387.3	345.0	140.5	759.0	129.2	323.6	94.8	98.9	**61.2**
		5	252.5	260.5	156.5	127.5	1080.5	213.2	559.7	182.4	148.7	**100.9**
	20	1	766.9	1506.0	808.8	750.1	2167.0	1341.7	914.8	391.4	375.5	**324.3**
		3	-	-	-	-	-	-	-	-	-	-
		5	-	-	-	-	-	-	-	-	-	-
Average			245.4	306.7	205.0	162.4	645.2	228.1	515.0	169.6	171.4	**109.6**

D Average Performance Comparison for General Chance Constraints

We now present additional results for the SOC-CC dataset. Figure 6 shows two plots comparing the root gap of `B-Gen+L` and `CPLEX` for the SOC-CC instances and different values of Ω and t. In each plot, an (x, y) point represents the root gap for an instance given by the x-axis and the y-axis technique, respectively. Overall, we can see that `B-Gen+L` achieves a smaller or equal root gap to `CPLEX`, however, the difference is considerably larger when $\Omega \geq 3$ and $t = 0.1$.

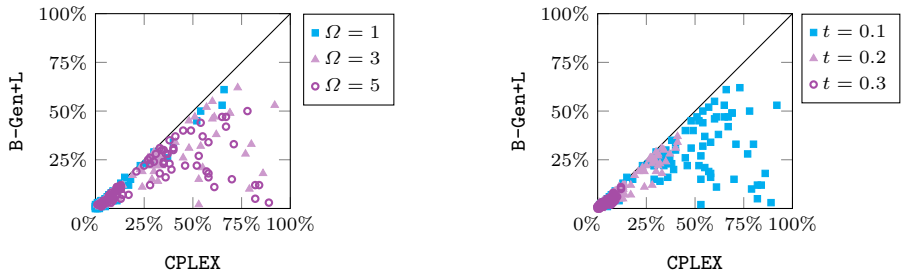

Fig. 6 Root node gap comparison with CPLEX for the SOC-CC instances. The plot on the left considers different values of Ω, and the plot on the right different values of t.

The problems become more challenging with a larger Ω (i.e., a predominant quadratic term) due to a weak SOC relaxation and linearization. Thus, our procedure can potentially generate stronger cuts than CPLEX. In fact, the left plot in Figure 6 shows all instances with $\Omega = 1$ close to the diagonal, while problems with $\Omega \in \{3, 5\}$ have larger gap reductions. Lastly, the right plot of Figure 6 shows that B-Gen+L has significantly smaller gaps than CPLEX over instances with a small t (i.e., small solution sets). Our relaxed BDDs are close to exact BDDs in these cases, thus, making our cuts more effective.

The following tables show average results for each parameter configuration over the SOC-CC benchmark. We present results for our four variants (i.e., B-Flow, B-Flow+L, B-Gen, and B-Gen+L), CPLEX, and best performing BDD-based cuts (i.e., B-Target and B-Ta+Fl+L). All techniques add cuts only at the root node of the tree search. Tables 11, 12, and 13 show the number of instances solved, average root gap, and average final gap for each n, m, Ω, and t combination, with $\mathcal{W} = 4000$. Similarly, Tables 14 and 15 show the average number of nodes in the branch-and-bound search and the average run time for the instances that all techniques solved to optimality.

We note that in most instances the BDD cut algorithms have better performance than CPLEX. However, CPLEX is competitive when $t = 0.3$ and $\Omega = 1$, as expected from the behavior observed in Figure 6. We also note that CPLEX has similar performance to the BDD techniques when $n = 125$, which suggest that a tighter BDD relaxation (i.e., bigger BDD width) might be needed for larger problem instances.

Table 11 Instances solved to Optimality comparison across all techniques for SOC-CC benchmark.

				# Instances Solved						
t	n	m	Ω	CPLEX	B-Flow	B-Flow+L	B-Gen	B-Gen+L	B-Target	B-Ta+Fl+L
0.1	75	10	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**
			3	**5**	**5**	**5**	**5**	**5**	**5**	**5**
			5	**5**	**5**	**5**	**5**	**5**	**5**	**5**
		20	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**
			3	**5**	**5**	**5**	**5**	**5**	**5**	**5**
			5	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	100	10	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**
			3	**5**	**5**	**5**	**5**	**5**	**5**	**5**
			5	**5**	**5**	**5**	**5**	**5**	**5**	**5**
		20	1	0	0	0	1	1	**2**	**2**
			3	**5**	**5**	**5**	**5**	**5**	**5**	**5**
			5	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	125	10	1	4	3	3	4	4	**5**	4
			3	0	0	0	0	0	**1**	0
			5	0	0	**2**	1	1	1	1
		20	1	0	0	0	0	0	0	0
			3	0	0	1	1	1	**2**	1
			5	0	0	1	2	2	**3**	2
0.2	75	10	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**
			3	2	3	4	**5**	**5**	**5**	**5**
			5	1	1	3	**5**	**5**	**5**	**5**
		20	1	2	2	3	3	**4**	**4**	3
			3	0	0	0	0	**1**	0	**1**
			5	0	0	0	0	**1**	**1**	**1**
	100	10	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**
			3	0	0	1	2	2	**3**	2
			5	0	0	0	0	0	0	0
		20	1	0	0	1	1	**2**	**2**	**2**
			3	0	0	0	0	0	0	0
			5	0	0	0	0	0	0	0
	125	10	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**
			3	0	0	0	0	0	0	0
			5	0	0	0	0	0	0	0
		20	1	0	0	0	0	0	0	0
			3	0	0	0	0	0	0	0
			5	0	0	0	0	0	0	0
0.3	75	10	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**
			3	**5**	**5**	**5**	**5**	**5**	**5**	**5**
			5	4	4	**5**	**5**	**5**	**5**	**5**
		20	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**
			3	0	0	1	**4**	**4**	**4**	**4**
			5	0	0	0	2	3	**4**	**4**
	100	10	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**
			3	**5**	**5**	**5**	**5**	**5**	**5**	**5**
			5	**5**	**5**	**5**	**5**	**5**	**5**	**5**
		20	1	4	3	4	**5**	**5**	**5**	**5**
			3	0	0	0	**1**	**1**	**1**	**1**
			5	0	0	0	0	0	0	0
	125	10	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**
			3	**5**	**5**	**5**	**5**	**5**	**5**	**5**
			5	**5**	**5**	**5**	**5**	**5**	**5**	**5**
		20	1	**5**	4	**5**	**5**	**5**	4	**5**
			3	0	0	0	0	0	0	0
			5	0	0	0	0	0	0	0
Total				137	135	149	162	167	**172**	168

Table 12 Root Gap comparison across all techniques for SOC-CC benchmark.

				Root Gap (%)						
t	n	m	Ω	CPLEX	B-Flow	B-Flow+L	B-Gen	B-Gen+L	B-Target	B-Ta+Fl+L
0.1	75	10	1	7.6%	10.0%	6.9%	3.8%	3.8%	**3.5%**	3.7%
			3	47.6%	45.4%	20.9%	16.4%	16.0%	**15.6%**	16.0%
			5	56.1%	47.3%	22.0%	17.8%	17.4%	17.9%	**17.3%**
		20	1	54.7%	56.5%	54.9%	**47.0%**	47.1%	47.1%	47.1%
			3	83.0%	75.7%	31.1%	25.3%	23.2%	30.0%	**23.1%**
			5	81.3%	26.0%	8.2%	**5.9%**	8.2%	16.6%	8.2%
	100	10	1	6.1%	7.5%	6.5%	5.4%	**5.3%**	**5.3%**	**5.3%**
			3	34.3%	34.0%	27.3%	23.3%	**23.2%**	23.3%	**23.2%**
			5	40.8%	39.7%	27.1%	**22.4%**	22.6%	**22.4%**	**22.4%**
		20	1	26.9%	28.8%	26.7%	**23.2%**	**23.2%**	**23.2%**	**23.2%**
			3	65.3%	62.9%	48.6%	**46.0%**	**46.0%**	**46.0%**	**46.0%**
			5	67.5%	59.6%	42.5%	**39.7%**	40.0%	**39.7%**	39.9%
	125	10	1	4.4%	5.5%	4.7%	**4.1%**	**4.1%**	**4.1%**	**4.1%**
			3	33.5%	33.3%	30.2%	**29.2%**	**29.2%**	**29.2%**	**29.2%**
			5	38.0%	36.4%	32.1%	31.2%	31.2%	**31.1%**	31.2%
		20	1	18.9%	19.8%	19.3%	**18.0%**	**18.0%**	**18.0%**	**18.0%**
			3	56.4%	56.4%	53.0%	47.8%	**47.7%**	47.8%	**47.7%**
			5	56.2%	55.0%	43.0%	**40.4%**	**40.4%**	**40.4%**	**40.4%**
0.2	75	10	1	2.9%	4.5%	3.6%	**2.0%**	**2.0%**	**2.0%**	**2.0%**
			3	10.1%	10.7%	9.0%	**5.4%**	**5.4%**	**5.4%**	**5.4%**
			5	13.1%	13.8%	11.1%	**7.0%**	**7.0%**	**7.0%**	**7.0%**
		20	1	11.6%	13.9%	11.4%	**7.8%**	**7.8%**	**7.8%**	**7.8%**
			3	30.6%	31.4%	30.2%	**21.0%**	**21.0%**	**21.0%**	**21.0%**
			5	32.5%	32.9%	29.2%	**20.5%**	**20.5%**	**20.5%**	**20.5%**
	100	10	1	2.6%	3.8%	3.1%	**2.4%**	**2.4%**	**2.4%**	**2.4%**
			3	8.3%	8.8%	8.0%	**6.2%**	**6.2%**	6.3%	**6.2%**
			5	11.8%	12.1%	11.5%	**9.8%**	**9.8%**	**9.8%**	**9.8%**
		20	1	7.7%	9.2%	8.6%	**6.5%**	**6.5%**	**6.5%**	**6.5%**
			3	24.6%	24.9%	24.1%	**20.7%**	**20.7%**	**20.7%**	**20.7%**
			5	30.1%	30.2%	29.1%	**25.6%**	**25.6%**	**25.6%**	**25.6%**
	125	10	1	**1.6%**	2.8%	2.6%	2.4%	2.4%	2.4%	2.4%
			3	7.7%	7.9%	7.7%	7.4%	7.4%	**7.3%**	7.4%
			5	9.4%	9.5%	9.2%	**8.6%**	**8.6%**	**8.6%**	**8.6%**
		20	1	**5.4%**	6.3%	6.2%	5.5%	5.5%	5.5%	5.5%
			3	20.6%	20.7%	20.5%	**19.0%**	**19.0%**	**19.0%**	**19.0%**
			5	25.1%	25.2%	24.9%	**22.9%**	**22.9%**	**22.9%**	**22.9%**
0.3	75	10	1	1.2%	2.7%	1.9%	**0.9%**	**0.9%**	**0.9%**	**0.9%**
			3	3.6%	4.4%	3.4%	**1.8%**	**1.8%**	**1.8%**	**1.8%**
			5	4.3%	5.0%	4.0%	**2.1%**	**2.1%**	**2.1%**	**2.1%**
		20	1	3.7%	5.5%	4.4%	2.4%	2.4%	**2.3%**	2.4%
			3	7.7%	9.0%	7.5%	**4.3%**	**4.3%**	**4.3%**	**4.3%**
			5	8.8%	9.8%	8.0%	**4.8%**	**4.8%**	**4.8%**	**4.8%**
	100	10	1	**1.0%**	2.1%	1.7%	1.2%	1.2%	1.2%	1.2%
			3	2.3%	3.1%	2.8%	**2.1%**	**2.1%**	**2.1%**	**2.1%**
			5	2.9%	3.6%	3.5%	**2.6%**	**2.6%**	**2.6%**	**2.6%**
		20	1	3.3%	4.6%	3.8%	**3.1%**	**3.1%**	**3.1%**	**3.1%**
			3	6.0%	6.7%	6.0%	**4.6%**	**4.6%**	**4.6%**	**4.6%**
			5	7.9%	8.6%	7.9%	**6.1%**	**6.1%**	**6.1%**	**6.1%**
	125	10	1	**0.3%**	1.1%	1.0%	0.9%	0.9%	0.9%	0.9%
			3	**1.6%**	2.2%	2.1%	1.9%	1.9%	1.9%	1.9%
			5	**1.8%**	2.3%	2.0%	2.0%	2.0%	2.0%	2.0%
		20	1	**1.7%**	2.8%	2.6%	2.4%	2.4%	2.4%	2.4%
			3	4.8%	5.4%	5.2%	**4.5%**	**4.5%**	**4.5%**	**4.5%**
			5	6.7%	7.2%	7.0%	**6.0%**	6.1%	**6.0%**	6.1%
Average				20.4%	19.5%	15.4%	**13.0%**	**13.0%**	13.3%	**13.0%**

Table 13 Final Gap comparison across all techniques for SOC-CC benchmark.

				Final Gap (%)						
t	n	m	Ω	CPLEX	B-Flow	B-Flow+L	B-Gen	B-Gen+L	B-Target	B-Ta+Fl+L
0.1	75	10	1	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
			3	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
			5	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
		20	1	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
			3	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
			5	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	100	10	1	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
			3	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
			5	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
		20	1	17.4%	18.4%	12.4%	7.5%	7.5%	**5.1%**	5.5%
			3	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
			5	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	125	10	1	0.3%	0.6%	0.7%	0.3%	0.4%	**0.0%**	0.2%
			3	25.1%	22.8%	17.6%	15.0%	16.1%	**14.5%**	16.0%
			5	28.0%	23.7%	13.0%	11.3%	**11.3%**	11.5%	**11.3%**
		20	1	16.0%	16.9%	16.5%	14.6%	14.4%	14.4%	**14.2%**
			3	52.0%	50.5%	32.3%	22.5%	21.8%	**19.8%**	22.1%
			5	49.4%	28.3%	10.4%	6.7%	4.1%	**2.5%**	6.9%
0.2	75	10	1	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
			3	1.4%	0.6%	0.1%	**0.0%**	**0.0%**	**0.0%**	**0.0%**
			5	4.5%	3.6%	1.2%	**0.0%**	**0.0%**	**0.0%**	**0.0%**
		20	1	3.2%	3.8%	2.5%	1.4%	**0.5%**	0.6%	0.7%
			3	24.5%	23.7%	22.7%	13.6%	12.9%	13.6%	**12.5%**
			5	25.1%	25.8%	21.9%	12.5%	12.3%	12.1%	**12.0%**
	100	10	1	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
			3	4.4%	4.5%	3.1%	1.8%	**1.3%**	**1.3%**	1.5%
			5	8.4%	8.9%	8.0%	6.2%	6.1%	**6.0%**	**6.0%**
		20	1	3.6%	4.6%	3.8%	2.1%	2.0%	**1.7%**	1.9%
			3	21.3%	22.3%	21.6%	17.6%	17.4%	**17.2%**	17.3%
			5	27.3%	28.0%	26.5%	22.7%	22.5%	22.5%	**22.4%**
	125	10	1	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
			3	5.4%	5.7%	5.4%	5.1%	**5.0%**	5.1%	5.1%
			5	7.2%	7.7%	7.7%	7.2%	7.1%	**7.0%**	7.1%
		20	1	3.6%	4.3%	4.1%	3.5%	**3.2%**	3.3%	3.5%
			3	19.2%	19.9%	19.7%	17.8%	17.8%	**17.7%**	17.8%
			5	23.7%	24.5%	24.2%	21.8%	**21.7%**	**21.7%**	**21.7%**
0.3	75	10	1	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
			3	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
			5	0.3%	0.4%	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
		20	1	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
			3	3.1%	3.5%	2.1%	0.6%	0.4%	**0.2%**	**0.2%**
			5	4.8%	5.4%	3.5%	1.0%	0.5%	0.5%	**0.4%**
	100	10	1	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
			3	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
			5	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
		20	1	0.2%	0.4%	0.3%	**0.0%**	**0.0%**	**0.0%**	**0.0%**
			3	3.4%	3.8%	3.3%	1.9%	**1.8%**	**1.8%**	1.9%
			5	5.9%	6.2%	5.7%	4.0%	**3.9%**	**3.9%**	**3.9%**
	125	10	1	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
			3	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
			5	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**	**0.0%**
		20	1	**0.0%**	0.1%	**0.0%**	**0.0%**	**0.0%**	0.1%	**0.0%**
			3	3.3%	3.9%	3.4%	3.0%	2.9%	**2.8%**	**2.8%**
			5	5.5%	5.9%	5.6%	4.8%	4.8%	**4.7%**	**4.7%**
		Average		7.4%	7.0%	5.5%	4.2%	4.1%	**3.9%**	4.1%

Table 14 Nodes explored comparison across all techniques for SOC-CC benchmark.

				# Nodes Explored						
t	n	m	Ω	CPLEX	B-Flow	B-Flow+L	B-Gen	B-Gen+L	B-Target	B-Ta+Fl+L
0.1	75	10	1	12,185	17,956	4,756	676	426	422	**399**
			3	56,780	11,892	674	363	338	327	**292**
			5	48,045	3,821	356	243	208	**178**	214
		20	1	202,904	215,447	142,030	26,079	**16,928**	24,498	18,745
			3	13,220	1,767	13	**6**	7	8	7
			5	11,944	24	**4**	5	**4**	**4**	**4**
	100	10	1	225,098	419,363	276,059	96,340	105,972	112,095	**89,448**
			3	887,680	783,566	109,802	58,768	53,628	**46,936**	53,654
			5	645,651	199,487	20,966	15,700	18,833	**11,710**	12,749
		20	1	-	-	-	-	-	-	-
			3	351,411	45,407	7,779	5,275	5,923	**5,054**	5,124
			5	270,319	14,754	5,028	2,206	2,372	2,212	**2,192**
	125	10	1	247,942	448,522	193,963	92,868	78,944	71,829	**53,197**
			3	-	-	-	-	-	-	-
			5	-	-	-	-	-	-	-
		20	1	-	-	-	-	-	-	-
			3	-	-	-	-	-	-	-
			5	-	-	-	-	-	-	-
0.2	75	10	1	12,121	20,715	7,593	1,244	**1,094**	1,128	1,264
			3	467,593	1,009,053	318,850	65,594	**26,086**	36,381	26,449
			5	465,102	501,715	260,340	141,485	105,020	**58,315**	117,946
		20	1	188,806	276,917	131,661	14,144	12,410	14,528	**9,460**
			3	-	-	-	-	-	-	-
			5	-	-	-	-	-	-	-
	100	10	1	95,393	248,309	151,732	52,426	33,054	51,027	**29,836**
			3	-	-	-	-	-	-	-
			5	-	-	-	-	-	-	-
		20	1	-	-	-	-	-	-	-
			3	-	-	-	-	-	-	-
			5	-	-	-	-	-	-	-
	125	10	1	**49,518**	312,285	255,539	312,597	204,109	213,868	230,317
			3	-	-	-	-	-	-	-
			5	-	-	-	-	-	-	-
		20	1	-	-	-	-	-	-	-
			3	-	-	-	-	-	-	-
			5	-	-	-	-	-	-	-
0.3	75	10	1	1,749	4,503	1,480	617	628	**471**	693
			3	105,042	289,610	88,043	4,729	3,330	3,172	**2,798**
			5	106,290	166,623	40,851	**2,344**	2,990	3,066	3,140
		20	1	46,165	92,516	64,630	7,094	3,820	**3,340**	3,928
			3	-	-	-	-	-	-	-
			5	-	-	-	-	-	-	-
	100	10	1	4,668	13,241	12,104	2,445	4,007	**2,414**	3,902
			3	98,320	232,323	215,941	41,555	54,620	**41,508**	48,554
			5	325,561	779,990	647,941	**130,800**	160,442	160,147	136,534
		20	1	73,317	104,970	110,269	**37,389**	39,225	37,523	40,692
			3	-	-	-	-	-	-	-
			5	-	-	-	-	-	-	-
	125	10	1	**621**	7,117	5,397	6,667	3,226	7,361	4,736
			3	115,889	289,327	310,125	140,877	117,999	**105,835**	108,763
			5	**225,090**	447,631	374,133	292,988	495,361	332,646	334,653
		20	1	**130,174**	374,273	234,563	304,234	188,323	319,374	155,635
			3	-	-	-	-	-	-	-
			5	-	-	-	-	-	-	-
Average				182,820	244,438	133,087	61,925	57,977	55,579	**49,844**

Table 15 Solving time comparison across all techniques for SOC-CC benchmark.

				Solving Time (s)						
t	n	m	Ω	CPLEX	B-Flow	B-Flow+L	B-Gen	B-Gen+L	B-Target	B-Ta+Fl+L
0.1	75	10	1	22.8	18.5	**9.5**	50.1	17.9	18.1	12.1
			3	54.2	14.4	**4.3**	17.8	5.1	10.4	5.1
			5	52.5	8.2	**3.4**	10.5	3.5	6.7	3.7
		20	1	1,184.5	641.2	531.6	438.8	244.4	178.2	**165.2**
			3	58.3	24.9	**10.3**	32.3	11.1	29.5	11.7
			5	59.8	15.2	**8.5**	15.7	8.9	13.8	9.0
	100	10	1	557.1	494.3	368.6	218.8	176.5	173.3	**141.8**
			3	1,648.1	948.1	216.3	194.8	**122.0**	135.2	132.3
			5	1,039.0	327.3	46.1	87.1	55.3	54.9	**41.7**
		20	1	-	-	-	-	-	-	-
			3	1,733.5	234.3	**49.2**	99.3	50.1	108.3	52.4
			5	1,177.4	87.7	36.9	53.1	**31.2**	81.6	31.2
	125	10	1	547.3	469.8	366.5	154.1	140.2	95.1	**83.3**
			3	-	-	-	-	-	-	-
			5	-	-	-	-	-	-	-
		20	1	-	-	-	-	-	-	-
			3	-	-	-	-	-	-	-
			5	-	-	-	-	-	-	-
0.2	75	10	1	20.8	21.4	**13.6**	39.6	29.8	25.7	18.6
			3	1,851.3	1,889.4	892.7	192.2	99.6	**74.5**	84.4
			5	1,418.2	854.6	490.0	356.2	337.9	**164.8**	199.9
		20	1	1,361.3	1,216.9	776.6	307.7	192.4	176.1	**114.0**
			3	-	-	-	-	-	-	-
			5	-	-	-	-	-	-	-
	100	10	1	209.1	257.1	169.8	93.5	63.5	83.8	**47.5**
			3	-	-	-	-	-	-	-
			5	-	-	-	-	-	-	-
		20	1	-	-	-	-	-	-	-
			3	-	-	-	-	-	-	-
			5	-	-	-	-	-	-	-
	125	10	1	**112.0**	239.0	228.6	255.0	190.2	210.2	205.3
			3	-	-	-	-	-	-	-
			5	-	-	-	-	-	-	-
		20	1	-	-	-	-	-	-	-
			3	-	-	-	-	-	-	-
			5	-	-	-	-	-	-	-
0.3	75	10	1	**4.7**	11.3	9.3	33.3	19.1	25.7	20.6
			3	249.8	392.1	109.6	137.0	93.4	63.0	**53.4**
			5	214.8	233.3	78.6	130.3	66.1	63.5	**40.3**
		20	1	236.1	270.0	200.0	161.2	84.8	78.1	**64.7**
			3	-	-	-	-	-	-	-
			5	-	-	-	-	-	-	-
	100	10	1	**11.0**	21.2	21.8	31.6	25.4	27.0	24.3
			3	245.3	374.1	319.2	118.7	144.6	**105.6**	122.1
			5	1,081.2	1,282.5	1,448.4	**348.5**	438.7	446.1	390.4
		20	1	550.6	315.5	317.7	166.8	144.2	**137.7**	176.7
			3	-	-	-	-	-	-	-
			5	-	-	-	-	-	-	-
	125	10	1	**2.5**	19.2	18.5	24.2	19.2	25.0	22.9
			3	301.2	468.9	445.3	238.9	**146.6**	196.7	152.8
			5	655.2	703.1	566.7	439.0	817.1	**418.0**	466.0
		20	1	1,255.8	1,490.4	**639.7**	1,251.0	962.6	1,211.9	752.7
			3	-	-	-	-	-	-	-
			5	-	-	-	-	-	-	-
Average				597.18	444.80	279.91	189.90	158.05	147.95	**121.54**