manuscript No.
(will be inserted by the editor)

Outer Approximation for Integer Nonlinear Programs via Decision Diagrams

Danial Davarnia · Willem-Jan van Hoeve

Received: date / Accepted: date

Abstract As an alternative to traditional integer programming (IP), decision diagrams (DDs) provide a new solution technology for discrete problems exploiting their combinatorial structure and dynamic programming representation. While the literature mainly focuses on the competitive aspects of DDs as a stand-alone solver, we investigate their complementary role by introducing IP techniques that can be derived from DDs and used in conjunction with IP to enhance the overall performance. This perspective allows for studying problems with more general structure than those typically modeled via recursive formulations. In particular, we develop linear programming and subgradient-type methods to generate valid inequalities for the convex hull of the feasible region described by DDs. For convex IPs, these cutting planes dominate the so-called linearized cuts used in the outer approximation framework. These cutting planes can also be derived for nonconvex IPs, which leads to a generalization of the outer approximation framework. Computational experiments show significant optimality gap improvement for integer nonlinear programs over the traditional cutting plane methods employed in the state-of-the-art solvers.

Keywords Decision Diagrams · Cutting planes · Outer approximation · Dynamic programming · Integer nonlinear programs.

1 Introduction

As a new technique to solve integer programs (IPs), decision diagrams (DDs) provide a graphical model that compactly represents the solution set together with the objective rates of the IP; see [9] for an introduction. In principle, DDs share similar recursive modeling as that used in dynamic programming (DP). However, instead of relying on the complete enumeration of the entire state space, DDs employ key notions of IP such as relaxation, restriction and branching search to overcome the drastic dimensionality growth of DP. Recent computational studies suggest that a solver based entirely on DDs is competitive and even superior to existing solution methods for particular classes of combinatorial problems such as maximum independent sets, maximum cut and maximum 2-SAT problems [10], as well as problems appearing in applications such as sequencing and scheduling [17].

D. Davarnia
Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, USA
E-mail: ddavarni@andrew.cmu.edu

W.-J. van Hoeve
Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, USA
E-mail: vanhoeve@andrew.cmu.edu

Currently, designing a DD package that contains effective relaxation, restriction and branching search relies on the existence of special structural properties in the problem; see for instance the approach used in [8] for set covering and stable set problems. This poses a limitation on the problem classes that can be studied via DDs, and hence encourages development of alternative DD techniques to expand the application domain. From a different perspective, [4] and [34] investigate cutting plane techniques obtained from DDs, and evaluate their strength for certain classes of linear IPs. It follows from these studies that the effectiveness of the cutting planes heavily depends on the strength of DDs that model the *entire* solution space of the problem. For this reason, the implementation has been limited to problems with special linear structures for which a method—often based on a recursive DP formulation—to construct strong DDs is available. This arises the important question of how to develop DD techniques to study a broader class of IPs with general structure (not necessarily linear or structurally recursive), while achieving improvement over the existing methods.

In this paper, we address the above question by introducing a framework that incorporates DD-driven algorithms in classical IP techniques to enhance the overall solution performance. In particular, we use the DD structure to derive valid inequalities for the feasible region of the IP. Our derivation allows for both methodological extension and computational improvement of the techniques proposed in [4] and [34]. Further, we employ the valid inequalities inside an *outer approximation* framework typically used for convex mixed-integer nonlinear programs (MINLPs); see [13] for an introduction. Our framework has the following main contributions. (i) It applies to IPs with general structures as it only requires the DD representation of constraints individually, rather than the entire solution space. This distinction alleviates the dimensionality issue of typical DP models as well as the design limitation of typical DD models, both of which stem from the need to represent the entire solution space. (ii) It provides a generalization of the traditional outer approximation techniques for classes of nonconvex discrete programs. (iii) It exploits the combinatorial structure of the constraints via DD models, which leads to the derivation of cutting planes stronger than those used in the outer approximation methods. This is further supported by our computational experiments, which show significant optimality gap improvement for integer nonlinear programs over the traditional cutting plane methods employed in the state-of-the-art solvers.

Literature review

Originating from switching circuits [29], DDs found their way into discrete optimization as an alternative solution method to the traditional IP methods [25]. Certain combinatorial structures are hard to exploit properly by existing IP techniques due to complicated cost and constraint functions. For such structures, the flexibility of DDs with respect to the functional representation of optimization models leads to a successful solution paradigm. Applications in sequencing and scheduling [17], and healthcare and energy [7] show the effectiveness of using DDs in optimization. As another advantage, DDs offer branching strategies that can be utilized in massively parallel computation much more effectively than typical IP methods; see Section 6 of [9]. Other applications of DD can be found in computer science [36], constraint programming [1], and artificial intelligence [32].

In this paper, we study integer nonlinear programs (INLPs) as a subclass of MINLPs; see [6] for a survey on methods and applications of MINLP in optimization. INLPs are NP-Hard as they contain integer linear programs (ILPs) as a subclass; see [23] for a discussion on the complexity of ILPs. Typical solution methods for solving MINLPs use branch-and-bound, a refinement framework that recursively constructs and solves convex relaxations of the problem over smaller domains. We refer the reader to [33] for techniques to construct convex relaxations for nonconvex structures. With the goal of tightening convex relaxations, various cutting plane techniques have been developed for MINLPs; see [14] for a recent survey. If the functions defining the constraints of MINLP are convex, an alternative solution method based on linear outer approximations of the problem can

be used. Typical outer approximation methods establish a refinement framework that recursively constructs and solves MILP approximations of the problem. We refer the reader to [13] for a survey on outer approximation methods. In this paper, we focus on outer approximation schemes that can be adopted for discrete programs using DD elements.

The classical mathematical programming (MP) solution techniques and the new DD solution technology each have their own advantages over the other. For instance, MP can easily model continuous variables and nonseparable functions, while enjoying a mature library of solvers and rich solution method developments over several decades; see [12] for a brief history of such developments for MILPs, and see [15] for a survey on advances in MINLPs. On the other hand, DD is flexible with respect to the functional form of constraints (convex, nonconvex, nonsmooth and even those that are not algebraically representable in black box models), while enjoying an effective branch-and-bound procedure. Several works in the DD literature study the competitive role of the above two solution methods and compare their numerical performances; see the case studies in [9] for instance. In this paper, however, we aim at the complementary role of these solution methods to benefit from the strength of both simultaneously.

The remainder of this paper is organized as follows. In Section 2, we give a brief background of DDs. Section 3 contains two cut-generating schemes derived from DDs. In Sections 4 and 5, we develop outer approximation frameworks for convex and nonconvex structures that employ the cuts generated from DD. In Section 6, we preset computational experiments on two classes of INLPs inspired by applications in packing and marketing. We give concluding remarks in Section 7.

2 Background on Decision Diagrams

In this section, we give a brief review on the basics of DDs for optimization. A comprehensive review can be found in [9].

Define $N := \{1, \dots, n\}$. We denote a DD by $\mathcal{D} = (\mathcal{U}, \mathcal{A}, l(.))$, where $\mathcal{U}$ represents the set of nodes and $\mathcal{A}$ represents the set of arcs in a top-down directed multi-graph, and function $l : \mathcal{A} \to \mathbb{Z}$ indicates the label of arcs in $\mathcal{A}$. The multi-graph induced by $\mathcal{D}$ is composed of n arc layers $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n$, and $n+1$ node layers $\mathcal{U}_1, \mathcal{U}_2, \dots, \mathcal{U}_{n+1}$. Node layer $\mathcal{U}_1$ contains a single *source* node s, and the node layer $\mathcal{U}_{n+1}$ contains a single *terminal* node t. The tail node of each arc $a \in \mathcal{A}_j$ for $j \in N$ belongs to $\mathcal{U}_j$, and its head node belongs to $\mathcal{U}_{j+1}$. Referring to the number of nodes in the node layer $\mathcal{U}_j$ as its *width* $|\mathcal{U}_j|$, we define the width of $\mathcal{D}$ as the maximum width of its node layers, *i.e.,* $|\mathcal{D}| = \max_{j \in N} |\mathcal{U}_j|$.

DD $\mathcal{D}$ represents a set of points of the form $\boldsymbol{x} = (x_1, \dots, x_n)$ with the following encoding. The label $l(a)$ of each arc $a \in \mathcal{A}_j$, for $j \in N$, represents the value of x_j. Each node in layer $\mathcal{U}_j$ has a maximum outdegree equal to the number of distinct integer values in the domain of variable x_j. This definition implies that each arc-specified path $\mathrm{P} = (a_1, \dots, a_n) \in \mathcal{A}_1 \times \dots \times \mathcal{A}_n$ from s to t encodes an integral assignment to vector $\boldsymbol{x} = (x_1, \dots, x_n)$ such that $x_j = l(a_j)$ for all $j \in N$. We denote the vector encoded by path P as $\boldsymbol{x}^{\mathrm{P}}$. The collection of points encoded by all paths of $\mathcal{D}$ is referred to as the solution set of $\mathcal{D}$ denoted by $\mathrm{Sol}(\mathcal{D})$. When DD encodes binary points, *i.e.,* $l : \mathcal{A} \to \{0, 1\}$, it is referred to as a *binary decision diagram* (BDD). When the encoded points are general integer-valued, it is called a *multivalued decision diagram* (MDD).

Consider now a bounded integer set $\mathcal{P} \subseteq \mathbb{Z}^n$. Set $\mathcal{P}$ can be expressed by a DD $\mathcal{D}$ whose collection of all s-t paths precisely encodes the points in $\mathcal{P}$, *i.e.,* $\mathcal{P} = \mathrm{Sol}(\mathcal{D})$. We refer to such a DD as *exact*. In this context, we refer to a DD $\overline{\mathcal{D}}$ that has $\mathcal{P} \subset \mathrm{Sol}(\overline{\mathcal{D}})$ as a *relaxed* DD. Similarly, we refer to a DD $\underline{\mathcal{D}}$ that has $\mathcal{P} \supset \mathrm{Sol}(\underline{\mathcal{D}})$ as a *restricted* DD.

Using the above descriptions, we can model IPs with DDs. Consider the bounded IP

$$z^* = \max \{ f(\boldsymbol{x}) \,|\, \boldsymbol{x} \in \mathcal{P} \}, \tag{2.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $\mathcal{P} \subseteq \mathbb{Z}^n$. To model (2.1) with a DD, we first form a *weighted* DD, denoted by $[\mathcal{D}|w(.)]$, where (i) $\mathcal{D}$ represents an exact DD encoding solutions of $\mathcal{P}$, and (ii) $w(.) : \mathcal{A} \to \mathbb{R}$ is

a weight function associated with arcs of $\mathcal{D}$ so that for each s-t path $\mathrm{P} = (a_1, \ldots, a_n)$, its weight $w(\mathrm{P}) := \sum_{i=1}^{n} w(a_i)$ is equal to $f(\boldsymbol{x}^{\mathrm{P}})$, the objective value of the integral solution corresponding to P. Construction of the weight function is immediate when $f(\boldsymbol{x})$ is separable, i.e., $f(\boldsymbol{x}) = \sum_{i=1}^{n} f_i(x_i)$, as we can simply define $w(a_i) = f_i(l(a_i))$ for $a \in \mathcal{A}_i$ and $i \in N$.

The above definition implies that z^* is equal to the weight of the longest path from s to t in the corresponding weighted graph. We refer to such a weighted DD as *exact* since it formulates (2.1) precisely. Note that the longest path can be obtained in $\mathcal{O}(|\mathcal{U}| + |\mathcal{A}|)$, as it is solved over a directed acyclic graph. We define a *relaxed* weighted DD $[\overline{\mathcal{D}}|\overline{w}(.)]$ to represent a relaxation of (2.1) where $\mathcal{P} \subseteq \mathrm{Sol}(\overline{\mathcal{D}})$ and $f(\boldsymbol{x}^{\mathrm{P}}) \geq \overline{w}(\mathrm{P})$ for all $\boldsymbol{x}^{\mathrm{P}} \in \mathcal{P}$. Similarly, we define a *restricted* weighted DD $[\underline{\mathcal{D}}|\underline{w}(.)]$ to represent a restriction of (2.1) where $\mathcal{P} \supseteq \mathrm{Sol}(\underline{\mathcal{D}})$ and $f(\boldsymbol{x}^{\mathrm{P}}) \leq \underline{w}(\mathrm{P})$ for all $\boldsymbol{x}^{\mathrm{P}} \in \mathcal{P}$. It is clear that the weight of the longest path from s to t in the graph corresponding to a relaxed (*resp.* restricted) weighted DD provides an upper bound (*resp.* a lower bound) for z^*.

Designing relaxed and restricted DDs is crucial for solving IPs as the size of the exact DD can become very large and hence impractical to construct. In such situations, a width limit W is imposed on the maximum allowable width of the DD to control its size and keep its construction manageable. The idea is to use relaxed and restricted DDs that satisfy the width limit, and provide upper and lower bounds for the optimal solution of the original problem. A specific branch-and-bound strategy is then employed to successively refine the relaxed and restricted DDs to improve the bounds and reduce the gap until proving optimality. These three elements, relaxation, restriction and branch-and-bound, yield a stand-alone solver package for IPs that is entirely based on DDs. We refer the reader to [9, 10] for a detailed account on constructing relaxation, restriction and branch-and-bound techniques for various classes of IP problems.

Although designing a stand-alone package is the most common use of DDs in solving IPs, it is mainly effective for certain problem structures that admit a natural recursive DP representation. In this paper, we present a novel approach that uses DDs to complement existing techniques rather than designing a stand-alone package. This allows for studying discrete problems of more general structure. In particular, we exploit the graphical structure of DDs to derive valid inequalities that can be used in conjunction with traditional IP techniques to enhance their performance.

3 Cut-Generating Methods

In this section, we present two methods to derive valid inequalities for the convex hull of the feasible solutions of DDs. One is based on a cut-generating linear program (CGLP) and the other is based on a subgradient-type method. In the remainder of the paper for the sake of notational convenience, we use label component l_a and weight component w_a for each arc $a \in \mathcal{A}$ as shorthands for functional values $l(a)$ and $w(a)$, respectively.

3.1 A cut-generating linear program

In [34], the authors use a network representation of DDs in a higher dimension to generate the so-called *target cuts* via a CGLP; see [3] for a description of CGLPs. We next reproduce these results using a different approach from that used in [34], which has a simpler interpretation and an advantageous practical property that we describe later. We start with the following fundamental result which is a generalization of [4] for MDDs.

Proposition 3.1 *Consider a DD* $\mathcal{D} = (\mathcal{U}, \mathcal{A}, l(.))$ *with solution set* $\mathrm{Sol}(\mathcal{D})$. *Define* $\mathcal{F}(\mathcal{D}) = \left\{(\boldsymbol{x}; \boldsymbol{y}) \in \mathbb{R}^n \times \mathbb{R}_+^{|\mathcal{A}|} \middle| (3.1a), (3.1b)\right\}$ *where*

$$\sum_{a \in \delta^+(i)} y_a - \sum_{a \in \delta^-(i)} y_a = f_i, \qquad \forall i \in \mathcal{U} \tag{3.1a}$$

$$\sum_{a \in \mathcal{A}_k} l_a \, y_a = x_k, \qquad \forall k \in N, \tag{3.1b}$$

where $f_s = -f_t = 1$, $f_i = 0$ *for* $i \in \mathcal{U} \setminus \{s, t\}$, *and* $\delta^+(i)$ *(resp.* $\delta^-(i)$*) denotes the set of outgoing (resp. incoming) arcs at node* i. *Then,* $\mathrm{proj}_x \, \mathcal{F}(\mathcal{D}) = \mathrm{conv}(\mathrm{Sol}(\mathcal{D}))$. □

In view of Proposition 3.1, (3.1a) represents the flow-balance constraints for the network induced by $\mathcal{D}$ where $\boldsymbol{y}$ denotes the flow vector and $\boldsymbol{f}$ denotes the supply/demand vector. Constraint (3.1b) represents the relation between variables $\boldsymbol{x}$ and $\boldsymbol{y}$. In particular, arc a with label l_a in layer $\mathcal{A}_k$ corresponds to value l_a assigned to variable x_k. Under the assumption that the source has a unit supply and the terminal has a unit demand, the integer solutions of the underlying network are in a one-to-one correspondence with the points of $\mathcal{P}$. The proposition follows from the fact that the coefficient matrix of constraints (3.1a) is totally-unimodular, and that variables x_k are free and defined through (3.1b).

Remark 3.1 The main advantage of our presentation of $\mathrm{conv}(\mathcal{P})$ in Proposition 3.1 over the one presented in [34] is that it eliminates the need to identify an interior point of $\mathcal{P}$ to translate the set to contain the origin, a task that can be tedious in practice for particular set structures.

Since we are interested in the convex hull of $\mathcal{P}$ in the space of variables $\boldsymbol{x}$, we next present the projection results; see [2] for details of the derivation. Let $\boldsymbol{\theta} \in \mathbb{R}^{|\mathcal{U}|}$ and $\boldsymbol{\gamma} \in \mathbb{R}^n$ represent the dual variables corresponding to constraints (3.1a) and (3.1b), respectively.

Proposition 3.2 *Consider any* $(\bar{\boldsymbol{\theta}}; \bar{\boldsymbol{\gamma}})$ *that satisfies*

$$\theta_{t(a)} - \theta_{h(a)} + l_a \gamma_k \leq 0, \qquad \forall k \in N, a \in \mathcal{A}_k \tag{3.2a}$$

$$\theta_s = 0, \tag{3.2b}$$

where $t(a)$ *and* $h(a)$ *represent the tail and the head node of arc* a, *respectively. Then, the inequality*

$$\sum_{k \in N} x_k \bar{\gamma}_k \leq \bar{\theta}_t, \tag{3.3}$$

is valid for $\mathrm{conv}(\mathcal{P})$. *Further, any facet-defining inequality of* $\mathrm{conv}(\mathcal{P})$ *is of the form* (3.3) *for some extreme ray* $(\bar{\boldsymbol{\theta}}; \bar{\boldsymbol{\gamma}})$ *of* (3.2a) *and* (3.2b). □

In the above system, (3.2b) is imposed since constraint (3.1a) represents the flow-balance requirement and hence its rows are linearly dependent. Therefore, we fix one of the dual variables at zero. Note that the original form of (3.3) after performing the projection is $\sum_{k \in N} x_k \bar{\gamma}_k \leq -\sum_{i \in \mathcal{U}} f_i \bar{\theta}_i$. The supply/demand vector $\boldsymbol{f}$ with values given in Proposition 3.1 together with (3.2b) reduce the right-hand-side to $\bar{\theta}_t$.

Proposition 3.2 yields a natural CGLP to separate a given point $\bar{\boldsymbol{x}}$ from $\mathrm{conv}(\mathcal{P})$. In the following Proposition 3.3, $C(\boldsymbol{\theta}, \boldsymbol{\gamma}) \leq 0$ represents a normalization constraint that is added to ensure that the optimal value of the CGLP is bounded; see Section 5 of [18] for more details on the properties of normalization constraints. The most common normalization constraints are the norm constraints. For instance, the standard *linear* normalization constraint is the norm-1 bound, while the standard *convex* normalization constraint is the norm-2 bound.

Proposition 3.3 *Consider a point $\bar{\boldsymbol{x}} \in \mathbb{R}^n$, and define*

$$\omega^* = \max \left\{ \sum_{k \in N} \bar{x}_k \gamma_k - \theta_t \,\middle|\, \begin{array}{l} (3.2a), (3.2b) \\ C(\boldsymbol{\theta}, \boldsymbol{\gamma}) \leq 0 \end{array} \right\}. \tag{3.4}$$

Then, $\bar{\boldsymbol{x}} \in \text{conv}(\mathcal{P})$ if and only if $\omega^* = 0$. Otherwise, $\bar{\boldsymbol{x}}$ can be separated from $\text{conv}(\mathcal{P})$ via $\sum_{k \in N} x_k \gamma_k^* \leq \theta_t^*$ where $(\boldsymbol{\theta}^*; \boldsymbol{\gamma}^*)$ is an optimal solution of (3.4). □

For a combinatorial problem with n integer variables and a maximum domain size L for these variables, the CGLP has $\mathcal{O}(|\mathcal{D}|n)$ variables and $\mathcal{O}(|\mathcal{D}|nL)$ constraints. Although this model generates cutting planes systematically, it can be computationally demanding to implement inside branch-and-bound for large-size problems; see the discussion in Section 6. For this reason, it is desirable to seek an alternative method that generates cutting planes directly in the space of original variables more efficiently. We give such a procedure in the following section.

3.2 A subgradient cut-generating method

In this section, we present a cut-generating method based on projected subgradient algorithms for convex programs. This algorithm has the advantage of exploiting the network structure of DDs to generate inequalities more effectively. To this end, we reformulate the CGLP model (3.4) as a bilevel program that contains only variables $\boldsymbol{\gamma}$ at the higher level. To make use of the projected subgradient scheme, we would need to make the normalization constraint $C(\boldsymbol{\theta}, \boldsymbol{\gamma}) \leq 0$ in (3.4) independent from θ. The following proposition shows that a normalization constraint that contains only variables $\boldsymbol{\gamma}$ (without $\boldsymbol{\theta}$) is sufficient to make the CGLP (3.4) is bounded.

Proposition 3.4 *Assume that $C(\boldsymbol{\gamma}) \leq 0$ implies that $|\gamma_k| \leq K$ for all $k \in N$ and some $K \in \mathbb{R}_+$. Then, the CGLP (3.4) is bounded.*

Proof Assume by contradiction that (3.4) is unbounded. Therefore, there exists a recession direction $(\bar{\boldsymbol{\theta}}; \bar{\boldsymbol{\gamma}})$ of (3.4) such that $\sum_{k \in N} \bar{x}_k \bar{\gamma}_k - \bar{\theta}_t > 0$. Since variables γ_k are bounded because of $C(\boldsymbol{\gamma}) \leq 0$, their corresponding component is zero in the recession direction, implying that $\bar{\theta}_t < 0$. To satisfy constraint (3.2a) for each layer, we must have $\bar{\theta}_i < 0$ for all $i \in \mathcal{U}$ as the DD corresponding to $\mathcal{P}$ is connected. This is a contradiction to the fact that $\bar{\theta}_i = 0$ due to (3.2b). □

Using the result of Proposition 3.4, we can reformulate (3.4) as

$$\omega^* = \max \left\{ \sum_{k \in N} \bar{x}_k \gamma_k - \nu^*(\boldsymbol{\gamma}) \,\middle|\, C(\boldsymbol{\gamma}) \leq 0 \right\}, \tag{3.5}$$

where $C(\boldsymbol{\gamma}) \leq 0$ is a convex normalization constraint and

$$\nu^*(\boldsymbol{\gamma}) = \min \left\{ \theta_t \,\middle|\, \begin{array}{l} \theta_{t(a)} - \theta_{h(a)} \leq -l_a \gamma_k, \, \forall k \in N, a \in \mathcal{A}_k \\ \theta_s = 0 \end{array} \right\}. \tag{3.6}$$

Next, we show some properties of problem (3.5).

Proposition 3.5 *Let $\bar{\boldsymbol{x}} \in \mathbb{R}^n$. The objective function of (3.5)*

(i) is piecewise linear and concave in $\boldsymbol{\gamma}$,
(ii) has subgradient $\bar{\boldsymbol{x}} - \boldsymbol{x}^{\mathrm{P}^*(\bar{\boldsymbol{\gamma}})}$ at any point $\bar{\boldsymbol{\gamma}} \in \mathbb{R}^n$, where $\boldsymbol{x}^{\mathrm{P}^*(\bar{\boldsymbol{\gamma}})}$ is the point encoding the longest s-t path $\mathrm{P}^*(\bar{\boldsymbol{\gamma}})$ of the weighted DD $[\mathcal{D}|w(.)]$ with arc weights $w_a = l_a \bar{\gamma}_k$ for $a \in \mathcal{A}_k$ and $k \in N$.

Proof (i) It suffices to show that $\nu^*(\boldsymbol{\gamma})$ is piecewise linear and convex in $\boldsymbol{\gamma}$. It is easy to verify that the dual of (3.6) is equivalent to the following problem where variables $-\boldsymbol{y}$ represent the dual of the first set of constraints in (3.6).

$$\nu^*(\boldsymbol{\gamma}) = \max \left\{ \sum_{k\in N} \gamma_k \sum_{a\in\mathcal{A}_k} l_a y_a \;\middle|\; \begin{array}{ll} \sum_{a\in\delta^+(i)} y_a - \sum_{a\in\delta^-(i)} y_a = 0, & \forall i \in \mathcal{U}\setminus\{s,t\} \\ \sum_{a\in\delta^+(s)} y_a = 1 & \\ \sum_{a\in\delta^-(t)} y_a = 1 & \\ y_a \geq 0, & \forall i \in \mathcal{U} \end{array} \right\}. \tag{3.7}$$

The objective of (3.7) can be expressed as the maximum of a finite number of linear functions in $\boldsymbol{\gamma}$, each corresponding to an extreme point of the polytope describing the feasible region of (3.7). We conclude that $\nu^*(\boldsymbol{\gamma})$ is piecewise linear and convex in $\boldsymbol{\gamma}$.

(ii) It suffices to show that $\boldsymbol{x}^{\mathrm{P}^*(\bar{\boldsymbol{\gamma}})}$ is a subgradient of $\nu^*(\boldsymbol{\gamma})$ at any point $\bar{\boldsymbol{\gamma}} \in \mathbb{R}^n$. Let $\boldsymbol{y}^*(\bar{\boldsymbol{\gamma}})$ be an optimal solution of (3.7) for $\bar{\boldsymbol{\gamma}}$. We have that $\nu^*(\bar{\boldsymbol{\gamma}}) = \sum_{k\in N} \bar{\gamma}_k \sum_{a\in\mathcal{A}_k} l_a y_a^*(\bar{\boldsymbol{\gamma}})$. It is clear that $\boldsymbol{y}^*(\bar{\boldsymbol{\gamma}})$ belongs to the feasible region of (3.7) for any choice of $\boldsymbol{\gamma}$ as this region is independent of $\boldsymbol{\gamma}$. Therefore, we obtain that $\nu^*(\boldsymbol{\gamma}) \geq \sum_{k\in N} \gamma_k \sum_{a\in\mathcal{A}_k} l_a y_a^*(\bar{\boldsymbol{\gamma}})$. Combining the above two relations, we obtain that $\nu^*(\boldsymbol{\gamma}) \geq \nu^*(\bar{\boldsymbol{\gamma}}) + \sum_{k\in N}(\gamma_k - \bar{\gamma}_k)\sum_{a\in\mathcal{A}_k} l_a y_a^*(\bar{\boldsymbol{\gamma}})$ for any $\boldsymbol{\gamma} \in \mathbb{R}^n$. This shows that the vector $\boldsymbol{s}(\bar{\boldsymbol{\gamma}})$ whose component k equals to $\sum_{a\in\mathcal{A}_k} l_a y_a^*(\bar{\boldsymbol{\gamma}})$ is a subgradient of $\nu^*(\boldsymbol{\gamma})$ at $\bar{\boldsymbol{\gamma}}$. It remains to show that $\boldsymbol{s}(\bar{\boldsymbol{\gamma}})$ is equal to $\mathrm{P}^*(\bar{\boldsymbol{\gamma}})$. It is easy to verify that (3.7) models the problem of finding the longest s-t path in the underlying network of $\mathcal{D}$ with arc weights defined as $w_a = l_a\gamma_k$ for arc $a \in \mathcal{A}_k$. Let $\boldsymbol{y}^*(\bar{\boldsymbol{\gamma}})$ be the optimal solution of this model. Referring to $\mathrm{P}^*(\bar{\boldsymbol{\gamma}})$ as the longest path associated with $\boldsymbol{y}^*(\bar{\boldsymbol{\gamma}})$, it follows from definition that the k^{th} component of $\boldsymbol{x}^{\mathrm{P}^*(\bar{\boldsymbol{\gamma}})}$ is equal to $\sum_{a\in\mathcal{A}_k} l_a y_a^*(\bar{\boldsymbol{\gamma}})$.

□

Proposition 3.5 provides the basis for a projected subgradient algorithm presented as Algorithm 1.

Algorithm 1 A cut-generating algorithm based on a projected subgradient method

Input: A DD $\mathcal{D} = (\mathcal{U}, \mathcal{A}, l(.))$ and a point $\bar{\boldsymbol{x}}$

1: Initialize $\tau \leftarrow 0$, $\boldsymbol{\gamma}^{(0)} \in \mathbb{R}^n$, $\tau^* \leftarrow 0$, $\Delta^* \leftarrow 0$
2: **while** The stopping criterion is NOT met **do**
3: Create a weighted DD $[\mathcal{D}|w(.)]$ with arc weights $w_a = l_a\gamma_k^{(\tau)}$ for all $k \in N$ and $a \in \mathcal{A}_k$
4: Find a longest s-t path $\mathrm{P}^{(\tau)}$ in $[\mathcal{D}|w(.)]$ and compute its encoding point $\boldsymbol{x}^{\mathrm{P}^{(\tau)}}$
5: **if** $\boldsymbol{\gamma}^{(\tau)}(\bar{\boldsymbol{x}} - \boldsymbol{x}^{\mathrm{P}^{(\tau)}}) > \max\{0, \Delta^*\}$ **then**
6: Update $\tau^* \leftarrow \tau$ and $\Delta^* \leftarrow \boldsymbol{\gamma}^{(\tau)}(\bar{\boldsymbol{x}} - \boldsymbol{x}^{\mathrm{P}^{(\tau)}})$
7: **end if**
8: Update $\boldsymbol{\phi}^{(\tau+1)} \leftarrow \boldsymbol{\gamma}^{(\tau)} + \rho_\tau(\bar{\boldsymbol{x}} - \boldsymbol{x}^{\mathrm{P}^{(\tau)}})$ for step size ρ_τ
9: Find the projection $\boldsymbol{\gamma}^{(\tau+1)}$ of $\boldsymbol{\phi}^{(\tau+1)}$ onto the set defined by $C(\boldsymbol{\gamma}) \leq 0$
10: $\tau \leftarrow \tau + 1$
11: **end while**
12: **if** $\Delta^* > 0$ **then**
13: Return inequality $\boldsymbol{\gamma}^{(\tau^*)}(\boldsymbol{x} - \boldsymbol{x}^{\mathrm{P}^{(\tau^*)}}) \leq 0$
14: **end if**

We next give a few remarks on Algorithm 1; we refer the reader to [11] for details on properties of subgradient algorithms. Since this method is a search algorithm, it starts with an initialization step and iteratively updates the vectors through computing subgradient values. As presented in line 1, the initial vector $\boldsymbol{\gamma}^{(0)}$ is chosen arbitrarily. The stopping criterion in line 2 can be defined by a certain number of iterations or an improvement rate error. The objective value, unlike gradient methods, does not uniformly improve at each iteration. For this reason, it is imperative for convergence

guarantee to keep track of the best improvement achieved so far. This is done in lines 5-7. The updating strategy in line 8 follows from the results of Proposition 3.5 that $(\bar{\boldsymbol{x}}-\boldsymbol{x}^{\mathrm{P}^{(\tau)}})$ is a subgradient of the objective of (3.5) at each point $\boldsymbol{\gamma}^{(\tau)}$. The convergence pattern of this algorithm depends on the choice of step size rule ρ_τ. For instance, for a constant step size, the convergence is guaranteed only to a vicinity of the optimal solution. In contrast, for a diminishing step size, the algorithm converges to the optimal solution. Since (3.5) is a convex program with a constraint, the employed algorithm needs to be of the projected subgradient type. In particular, an updated solution must be projected onto the convex set defined by the constraint. This is done in line 9. In order for this step to be efficient, we may choose the norm-2 bound as the normalization constraint, i.e., $C(\boldsymbol{\gamma}) = ||\boldsymbol{\gamma}||^2 - 1$. Therefore, the projection of $\bar{\boldsymbol{\gamma}}$ onto this set is $\bar{\boldsymbol{\gamma}}$ if $||\bar{\boldsymbol{\gamma}}|| \leq 1$, and $\frac{\bar{\boldsymbol{\gamma}}}{||\bar{\boldsymbol{\gamma}}||}$ otherwise. At each iteration τ, the obtained inequality is of the form $\boldsymbol{\gamma}^{(\tau)}(\boldsymbol{x} - \boldsymbol{x}^{\mathrm{P}^{(\tau)}}) \leq 0$, which is valid for $\mathrm{conv}(\mathrm{Sol}(\mathcal{D}))$. In this algorithm, Δ^* stores the most violated value of all such inequalities up to the current iteration. If $\Delta^* > 0$, then the corresponding inequality is a cutting plane. Note for the case $\Delta^* = 0$ that, depending on the number of iterations and the step size rule, it is possible that the point can be separated but the algorithm has not yet converged to a vicinity with positive violation. For this reason, in practice we may need to solve the CGLP (3.4) in case the algorithm does not generate cutting planes, to verify that a point cannot be separated. It is our experience, however, that a violated inequality is often detected by the algorithm much faster than the CGLP unless $\bar{\boldsymbol{x}}$ is very close to $\mathrm{conv}(\mathrm{Sol}(\mathcal{D}))$, in which case the impact of adding a violated inequality to improve the objective value is trivial.

We next give a geometric intuition of Algorithm 1. The main idea is to view $\boldsymbol{\gamma}^{(\tau)}$ as the normal vector of a valid inequality of the form $\boldsymbol{\gamma}^{(\tau)}\boldsymbol{x} \leq \psi$ whose right-hand-side ψ needs to be calculated. This calculation is done through lines 3 and 4 of the algorithm. These lines compute the minimum right-hand-side value for $\boldsymbol{\gamma}^{(\tau)}\boldsymbol{x} \leq \psi$ to be valid for $\mathrm{conv}(\mathrm{Sol}(\mathcal{D}))$, i.e., $\boldsymbol{\gamma}^{(\tau)}\boldsymbol{x} \leq \boldsymbol{\gamma}^{(\tau)}\boldsymbol{x}^{\mathrm{P}^{(\tau)}}$ is the strongest valid inequality with the fixed normal vector $\boldsymbol{\gamma}^{(\tau)}$. We refer to this inequality as *normal inequality* corresponding to $\boldsymbol{\gamma}^{(\tau)}$. It is easy to verify that the closure of normal inequalities for all vectors $\boldsymbol{\gamma}^{(\tau)}$ describes $\mathrm{conv}(\mathrm{Sol}(\mathcal{D}))$. With this perspective, the algorithm searches for the normal vector of a normal inequality that separates $\bar{\boldsymbol{x}}$. The updating step in this search is obtained by "tilting" the current normal vector $\boldsymbol{\gamma}^{(\tau)}$ toward the direction of the subgradient vector connecting $\boldsymbol{x}^{\mathrm{P}^{(\tau)}}$ (which can be interpreted as a feasible point that is tight at the current normal inequality) to $\bar{\boldsymbol{x}}$ (the point desired to be separated).

4 Outer Approximation for Convex Structures

The cut-generating methods described in Section 3 can be used to derive valid inequalities from DDs. These *DD inequalities* can be added as valid cuts inside branch-and-bound via existing frameworks. In this section, we present a particular framework that naturally conforms to our proposed DD analysis. This framework is based on the *outer approximation* method designed for convex MINLPs. We show how this method can be adapted for a larger class of INLPs with the help of DD inequalities.

Consider the MINLP

$$\max \quad \boldsymbol{c}^\intercal \boldsymbol{x} + \boldsymbol{d}^\intercal \boldsymbol{z} \tag{4.1a}$$

$$\text{s.t.} \quad g^j(\boldsymbol{x}, \boldsymbol{z}) \leq 0, \qquad \forall j \in J \tag{4.1b}$$

$$\boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}] \cap \mathbb{Z}^n, \boldsymbol{z} \in \mathbb{R}^m, \tag{4.1c}$$

where variables $\boldsymbol{x}$ are discrete with integer lower and upper bounds denoted by $[\boldsymbol{l}, \boldsymbol{u}]$, and variables $\boldsymbol{z}$ are continuous. Assume in the above problem that functions $g^j(\boldsymbol{x}, \boldsymbol{z}) : \mathbb{R}^{n+m} \to \mathbb{R}$ for $j \in J$ are convex and sufficiently smooth over the variables' domain. This problem is often referred to as

convex MINLP in the literature, since its continuous relaxation obtained by removing the integrality restriction on variables $\boldsymbol{x}$ is a convex program.

A popular scheme to solve convex MINLPs is based on the successive generation of the so-called *linearization cuts* that are added to a linear outer approximation of problem (4.1); see [13] for a survey. Then, a mixed integer linear program (MILP) relaxation of (4.1), composed of the tightened outer approximation and the integrality restriction on variables $\boldsymbol{x}$, is solved. If not feasible for the original problem, the optimal solution of this MILP relaxation is cut off via linearization cuts, and the procedure is repeated. It can be shown that under conditions such as constraint qualification, the outer approximation algorithms converge to the optimal solution or prove infeasibility after a finite number of iterations.

There are several algorithms that share the core ingredients of the outer approximation technique described above. For a detailed account on such algorithms, we refer the reader to [24, 21, 31, 28]. In this paper, we use DD inequalities as a substitute for the linearization cuts in an outer approximation method called the *Extended Cutting Plane* (ECP) technique [37].

4.1 ECP algorithm

Since our DD analysis applies to discrete programs, we first present the ECP algorithm under the assumption that all variables are integer-valued, and derive key properties of the algorithm that will be used later for our adaptation.

Assumption 1 *All variables are integral.*

Imposing Assumption 1 on problem (4.1), we obtain an integer nonlinear program (INLP)

$$\begin{aligned} \max \quad & \boldsymbol{c}^\intercal \boldsymbol{x} \\ \text{s.t.} \quad & g^j(\boldsymbol{x}) \leq 0, && \forall j \in J \\ & \boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}] \cap \mathbb{Z}^n, \end{aligned} \tag{4.2}$$

where functions $g^j(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ are convex, sufficiently smooth. We refer to this INLP as $(\mathcal{E})$. Given any finite set $\mathcal{K} \subseteq \mathbb{R}^n$, we define $(\mathcal{E}^L(\mathcal{K}))$ as

$$\begin{aligned} \max \quad & \boldsymbol{c}^\intercal \boldsymbol{x} \\ \text{s.t.} \quad & g^j(\bar{\boldsymbol{x}}) + \nabla g^j(\bar{\boldsymbol{x}})(\boldsymbol{x} - \bar{\boldsymbol{x}}) \leq 0, && \forall \bar{\boldsymbol{x}} \in \mathcal{K}, \quad j \in \bar{J}(\bar{\boldsymbol{x}}) \\ & \boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}] \cap \mathbb{Z}^n, \end{aligned} \tag{4.3}$$

where $\bar{J}(\bar{\boldsymbol{x}}) := \{j \in J \mid g^j(\bar{\boldsymbol{x}}) > 0\}$, i.e., the index set of all constraints that are violated at point $\bar{\boldsymbol{x}}$. The difference between $(\mathcal{E})$ and $(\mathcal{E}^L(\mathcal{K}))$ is that the nonlinear constraints (4.2) in $(\mathcal{E})$ are replaced with the linearized constraints (4.3) violated by points of $\mathcal{K}$ in $(\mathcal{E}^L(\mathcal{K}))$. The ECP method can then be described as Algorithm 2.

Algorithm 2 ECP algorithm for convex INLP

1: Initialize $\mathcal{K} \leftarrow \emptyset$, $i \leftarrow 1$
2: Solve $(\mathcal{E}^L(\mathcal{K}))$
3: **if** $(\mathcal{E}^L(\mathcal{K}))$ is infeasible **then**
4: Return infeasibility for $(\mathcal{E})$
5: **else**
6: Find an optimal solution $\bar{\boldsymbol{x}}^i$ of $(\mathcal{E}^L(\mathcal{K}))$
7: **end if**
8: **if** $g^j(\bar{\boldsymbol{x}}^i) \leq 0$ for all $j \in J$ **then**
9: Return $\bar{\boldsymbol{x}}^i$ as the optimal solution of $(\mathcal{E})$
10: **else**
11: $\mathcal{K} \leftarrow \mathcal{K} \cup \{\bar{\boldsymbol{x}}^i\}$, $i \leftarrow i + 1$, go to 2
12: **end if**

Algorithm 2 returns the optimal solution of $(\mathcal{E})$ or proves its infeasibility after a finite number of iterations (without the need for additional conditions such as constraint qualification) as shown below. For any $j \in J$, define $\mathcal{G}^j_{\mathbb{R}} := \{\boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}] \,|\, g^j(\boldsymbol{x}) \leq 0\}$ and $\mathcal{G}^j_{\mathbb{Z}} := \{\boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}] \cap \mathbb{Z}^n \,|\, g^j(\boldsymbol{x}) \leq 0\}$.

Proposition 4.1 *(i) Consider $\bar{\boldsymbol{x}} \in [\boldsymbol{l}, \boldsymbol{u}] \cap \mathbb{Z}^n$ such that $g^j(\bar{\boldsymbol{x}}) > 0$ for some $j \in J$. Then $\bar{\boldsymbol{x}} \notin \text{conv}(\mathcal{G}^j_{\mathbb{Z}})$.*
(ii) For any $j \in J$, the linearization inequality $g^j(\bar{\boldsymbol{x}}) + \nabla g^j(\bar{\boldsymbol{x}})(\boldsymbol{x} - \bar{\boldsymbol{x}}) \leq 0$ is valid for $\text{conv}(\mathcal{G}^j_{\mathbb{R}})$ for any $\bar{\boldsymbol{x}} \in \mathbb{R}^n$. Further, this inequality is violated at $\bar{\boldsymbol{x}}$ if $\bar{\boldsymbol{x}} \in [\boldsymbol{l}, \boldsymbol{u}] \cap \mathbb{Z}^n$ and $\bar{\boldsymbol{x}} \notin \text{conv}(\mathcal{G}^j_{\mathbb{Z}})$.

Proof (i) We write that $\bar{\boldsymbol{x}} \notin \mathcal{G}^j_{\mathbb{R}} = \text{conv}(\mathcal{G}^j_{\mathbb{R}}) \supseteq \text{conv}(\mathcal{G}^j_{\mathbb{Z}})$, where the exclusion follows from $g^j(\bar{\boldsymbol{x}}) > 0$, the equality holds because $g^j(\boldsymbol{x})$ is convex, and the inclusion follows from the fact that $\mathcal{G}^j_{\mathbb{Z}} \subseteq \mathcal{G}^j_{\mathbb{R}}$.
(ii) We write that $g^j(\bar{\boldsymbol{x}}) + \nabla g^j(\bar{\boldsymbol{x}})(\boldsymbol{x} - \bar{\boldsymbol{x}}) \leq g^j(\boldsymbol{x}) \leq 0$ for any $\boldsymbol{x} \in \mathcal{G}^j_{\mathbb{R}}$, where the first inequality holds because $g^j(\boldsymbol{x})$ is convex, and the second inequality follows from the definition of $\mathcal{G}^j_{\mathbb{R}}$. Since $\mathcal{G}^j_{\mathbb{R}} = \text{conv}(\mathcal{G}^j_{\mathbb{R}})$, we conclude that $g^j(\bar{\boldsymbol{x}}) + \nabla g^j(\bar{\boldsymbol{x}})(\boldsymbol{x} - \bar{\boldsymbol{x}}) \leq 0$ is valid for $\text{conv}(\mathcal{G}^j_{\mathbb{R}})$. For the next part, assume that $\bar{\boldsymbol{x}} \in [\boldsymbol{l}, \boldsymbol{u}] \cap \mathbb{Z}^n$ and $\bar{\boldsymbol{x}} \notin \text{conv}(\mathcal{G}^j_{\mathbb{Z}})$. It is clear that $\bar{\boldsymbol{x}} \notin \text{conv}(\mathcal{G}^j_{\mathbb{Z}}) \supseteq \mathcal{G}^j_{\mathbb{Z}}$. Since $\bar{\boldsymbol{x}} \in [\boldsymbol{l}, \boldsymbol{u}] \cap \mathbb{Z}^n$, it follows from the definition of $\mathcal{G}^j_{\mathbb{Z}}$ that $g^j(\bar{\boldsymbol{x}}) > 0$. Therefore, the linearization inequality is violated at $\bar{\boldsymbol{x}}$.

□

It follows from property(i) of Proposition 4.1 that, to separate an infeasible point from the feasible region of $(\mathcal{E})$, it is sufficient to consider constraints individually. Property(ii) of Proposition 4.1 gives a class of valid inequalities for $(\mathcal{E})$ that separate infeasible points. These two properties are sufficient to show the finite convergence of the ECP Algorithm 2 as described next.

Proposition 4.2 *Algorithm 2 converges to the optimal solution of $(\mathcal{E})$ or proves that it is infeasible in a finite number of iterations.*

Proof Let $\mathcal{FE}$ and $\mathcal{FE}^L(\mathcal{K})$ represent the feasible region of $(\mathcal{E})$ and $(\mathcal{E}^L(\mathcal{K}))$, respectively. The algorithm starts with solving $(\mathcal{E}^L(\mathcal{K}))$ where $\mathcal{K} = \emptyset$. It follows from its description that $(\mathcal{E}^L(\emptyset))$ is an MILP relaxation of $(\mathcal{E})$. Therefore, its optimal solution $\bar{\boldsymbol{x}}^1 \in [\boldsymbol{l}, \boldsymbol{u}] \cap \mathbb{Z}^n$ is optimal for $(\mathcal{E})$ if $g^j(\bar{\boldsymbol{x}}^1) \leq 0$ for all $j \in J$. Otherwise, there exists $j \in J$ such that $g^j(\bar{\boldsymbol{x}}^1) > 0$. It follows from Proposition 4.1(i) that $\bar{\boldsymbol{x}}^1 \notin \text{conv}(\mathcal{G}^j_{\mathbb{Z}})$. Since $\bar{\boldsymbol{x}}^1 \in [\boldsymbol{l}, \boldsymbol{u}] \cap \mathbb{Z}^n$ and $\bar{\boldsymbol{x}}^1 \notin \text{conv}(\mathcal{G}^j_{\mathbb{Z}})$, Proposition 4.1(ii) implies that the linearization inequality $g^j(\bar{\boldsymbol{x}}^1) + \nabla g^j(\bar{\boldsymbol{x}}^1)(\boldsymbol{x} - \bar{\boldsymbol{x}}^1) \leq 0$ is valid for $\text{conv}(\mathcal{G}^j_{\mathbb{Z}}) \supseteq \text{conv}(\mathcal{FE})$. Further, this inequality separates $\bar{\boldsymbol{x}}^1$ from $\text{conv}(\mathcal{FE}^L(\emptyset))$. Therefore, adding $\bar{\boldsymbol{x}}^1$ to $\mathcal{K}$ tightens the MILP relaxation while cutting off the integer point $\bar{\boldsymbol{x}}^1$. The proposition result follows from the facts that there is a finite number of integer points in the bounded feasible region of $(\mathcal{E}^L(\emptyset))$, and that at least one integer point is cut off from this feasible region at each iteration of the algorithm. □

4.2 DD-ECP algorithm

Our goal here is to use the idea of constructing DDs in conjunction with the ECP algorithm to obtain an alternative solution algorithm for $(\mathcal{E})$. As described in Section 4.1, a key feature of the ECP Algorithm 2 is that it produces cutting planes with respect to individual constraints (4.2) of $(\mathcal{E})$. This is a desirable property for applying DD analysis to INLPs, since the construction of DD is more effective when it considers one constraint at a time, rather than the entire solution space. In particular, define the set of discrete points described by a single constraint over variable domains as $\mathcal{G}_{\mathbb{Z}} := \{\boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}] \cap \mathbb{Z}^n \,|\, g(\boldsymbol{x}) \leq 0\}$, where $g(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$. Let $\mathcal{D}_j$ be the DD representing the solutions of $\mathcal{G}^j_{\mathbb{Z}}$, i.e., $\text{Sol}(\mathcal{D}_j) = \mathcal{G}^j_{\mathbb{Z}}$; see the discussion in Section 6 for the construction technique. Next, we define $(\mathcal{E}^D(\mathcal{K}))$ for any finite set $\mathcal{K} \subseteq \mathbb{R}^n$ as

$$\begin{aligned} \max \quad & \boldsymbol{c}^\intercal \boldsymbol{x} \\ \text{s.t.} \quad & h_j(\bar{\boldsymbol{x}}, \boldsymbol{x}) \leq 0, && \forall \bar{\boldsymbol{x}} \in \mathcal{K}, \quad j \in \bar{J}(\bar{\boldsymbol{x}}) \\ & \boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}] \cap \mathbb{Z}^n, \end{aligned} \tag{4.4}$$

where the inequality (4.4) is obtained via the CGLP of Proposition 3.3 to separate point $\bar{\boldsymbol{x}} \in \mathcal{K}$ from the convex hull of the solution set $\text{Sol}(\mathcal{D}_j)$. This inequality replaces the linearization cut (4.3) in $(\mathcal{E}^L(\mathcal{K}))$.

Definition 4.1 Define the *DD-ECP Algorithm* similarly to the ECP Algorithm 2 with the difference that $(\mathcal{E}^D(\mathcal{K}))$ replaces $(\mathcal{E}^L(\mathcal{K}))$.

To prove the finite convergence results for the DD-ECP algorithm, note that the property given in Proposition 4.1(i) holds as the constraints of $(\mathcal{E})$ remain unchanged, while we replace the property described in Proposition 4.1(ii) with the following property obtained directly from the definition of the DD inequalities in Proposition 3.3.

Proposition 4.3 *For any point $\bar{\boldsymbol{x}} \in [\boldsymbol{l}, \boldsymbol{u}] \cap \mathbb{Z}^n$ such that $\bar{\boldsymbol{x}} \notin \text{conv}(\mathcal{G}^j_{\mathbb{Z}})$ for some $j \in J$, the DD inequality $h_j(\bar{\boldsymbol{x}}, \boldsymbol{x}) \leq 0$ obtained from the CGLP of Proposition 3.3 that separates $\bar{\boldsymbol{x}}$ from $\text{conv}(\text{Sol}(\mathcal{D}_j))$ is valid for $\text{conv}(\mathcal{G}^j_{\mathbb{Z}})$ and separates $\bar{\boldsymbol{x}}$ from it.* □

Using Proposition 4.1(i) and Proposition 4.3, an argument similar to that of the Proposition 4.2 gives the convergence results for the DD-ECP algorithm.

Proposition 4.4 *Algorithm DD-ECP of Definition 4.1 converges to the optimal solution of $(\mathcal{E})$ or proves that it is infeasible in a finite number of iterations.* □

Remark 4.1 Although the DD inequalities (4.4) in the definition of $(\mathcal{E}^D(\mathcal{K}))$ are represented based on the CGLP model of Proposition 3.3, we can replace them with the inequalities obtained from the subgradient method of Algorithm 1. The latter approach is better suited in practice due to its computational advantages, however additional care must be taken because of its convergence characteristics; see the discussion following Algorithm 1.

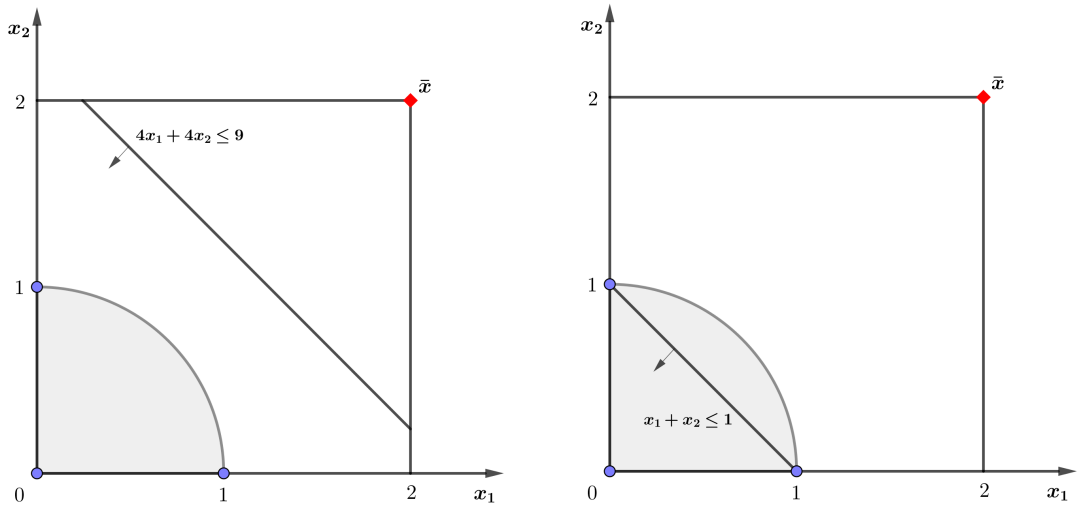
Next, we discuss some of the advantages of using DD-ECP over the traditional ECP algorithm. The first advantage stems from the strength of the DD cuts (4.4) of $(\mathcal{E}^D(\mathcal{K}))$ compared to the linearization cuts (4.3) of $(\mathcal{E}^L(\mathcal{K}))$. In particular, the strength of the linearization cuts depends on the distance of the point to be separated from the region defined by the constraint, whereas the DD cuts are independent of this distance and are derived with respect to the convex hull of the region defined by the constraint. Furthermore, the linearization cuts are always valid for the convex hull of the continuous points satisfied by the constraint, whereas the DD cuts are valid for the convex hull of the discrete points satisfied by the constraint. This leads to the derivation of DD inequalities that can cut through the continuous region defined by the constraint. We illustrate these advantages in Example 4.1.

The second advantage is the flexibility of the constraint forms that can be modeled by DD. As a result, DDs can model convex, non-convex, non-smooth, and even functions that do not have explicit algebraic form such as black-box constraints. We discuss examples of non-convex models in Sections 5 and 6. As for the convex functions at the focus of this section, the linearization cuts require smoothness whereas DD cuts do not.

The third advantage is the computational efficiency of constructing DDs for individual constraints. Since DDs used to generate DD cuts in $(\mathcal{E}^D(\mathcal{K}))$ represent each constraint (4.2) separately, they can be constructed in a *preprocessing* phase. The constructed DDs are then used during the solution process for solving a CGLP or a sequence of longest path problems for the subgradient methods. This will accelerate the solution process significantly as described in Section 6.

Example 4.1 Consider the constraint $g(x_1, x_2) \leq 0$ where $g(x_1, x_2) = x_1^2 + x_2^2 - 1$. Define bounds on variables x_1 and x_2 as $[l_1, u_1] = [l_2, u_2] = [0, 2]$. It follows from the definition that $\mathcal{G}_{\mathbb{Z}} = \{(0,0), (0,1), (1,0)\}$. Let $(\bar{x}_1, \bar{x}_2) = (2, 2)$ be the point that is to be separated from $\text{conv}(\mathcal{G}_{\mathbb{Z}})$. The linearization cut (4.3) at this point is $4x_1 + 4x_2 \leq 9$; see Figure 4.1a. Let $\mathcal{D}$ represent the set of points

in $\mathcal{G}_{\mathbb{Z}}$. The DD cut obtained from solving the CGLP gives a valid inequality of conv(Sol($\mathcal{D}$)) that is most violated at $(\bar{x}_1, \bar{x}_2)$. It is easy to verify that such inequality is $x_1 + x_2 \leq 1$; see Figure 4.1b. It is clear that (i) the DD inequality dominates the linearization inequality, (ii) the DD inequality cuts through the continuous region described by $g(x_1, x_2) \leq 0$ over the box domain, whereas the linearization inequality is valid for this region.



(a) Linearization cut (b) DD cut

Fig. 4.1: DD cut vs. linearization cut for the set of Example 4.1

5 Outer Approximation for Nonconvex Structures

As described in [16], nonconvex MINLPs have a much more complicated structure than those of the convex case. It is known that the traditional outer approximation algorithms are not viable for nonconvex structures; see [13]. Consider problem ($\mathcal{E}$) defined in the previous section with the difference that $g^j(\boldsymbol{x})$ is a general function (not necessarily convex). It follows from the proof of Proposition 4.2 that the main elements to guarantee the convergence of the ECP algorithm are the properties given in Proposition 4.1. These properties, however, do not hold when constraints of ($\mathcal{E}$) are nonconvex. Property (i) does not hold as the convex hull of the feasible region described by a nonconvex constraint can strictly subsume this feasible region. Property (ii) does not hold as the linearization inequalities are not generally valid for nonconvex constraints.

In this section, we investigate the implementation of the DD inequalities in the ECP algorithm for nonconvex INLPs. Consider an inequality $g^j(\boldsymbol{x}) \leq 0$ of ($\mathcal{E}$) where $g^j(\boldsymbol{x})$ is a general function. Let $\mathcal{G}^j_{\mathbb{Z}}$ represent the discrete feasible region satisfied by this inequality over the variables' domain $[\boldsymbol{l}, \boldsymbol{u}]$ as given in Section 4. Let $\mathcal{D}_j$ be the DD representing $\mathcal{G}^j_{\mathbb{Z}}$. It is clear that the result of Proposition 4.3 holds for such nonconvex structure. Using this result as a substitute for the property described in Proposition 4.1(ii), we can generate valid inequalities for the convex hull of the feasible region of ($\mathcal{E}$) through the implementation of the DD inequalities. In the next two sections, we study generalization of the DD-ECP algorithm for different classes of nonconvex INLPs.

5.1 Integer-quasiconvex functions

In this section, we show that the DD-ECP algorithm of Definition 4.1 can be used to solve a class of nonconvex problems that subsumes the convex INLPs discussed in Section 4. This generalized class contains nonconvex problems that appear in marketing applications when we model product price versus demand; see the pricing problem in Section 6.2. To introduce this class, we first recall the definition of quasiconvex and integer-convex functions.

Definition 5.1 A function $g(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ is *quasiconvex* if its α-level set $S^\alpha(g) := \{\boldsymbol{x} \in \mathbb{R}^n \,|\, g(\boldsymbol{x}) \leq \alpha\}$ is convex for any $\alpha \in \mathbb{R}$.

Definition 5.2 A function $g(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ is *integer-convex* if $g(\sum_{i=1}^t \lambda^i \boldsymbol{x}^i) \leq \sum_{i=1}^t \lambda^i g(\boldsymbol{x}^i)$ for all $\lambda^i \in \mathbb{R}_+$, $\boldsymbol{x}^i \in \mathbb{Z}^n$ such that $\sum_{i=1}^t \lambda^i = 1$ and $\sum_{i=1}^t \lambda^i \boldsymbol{x}^i \in \mathbb{Z}^n$.

We refer the reader to [22] and [26] for applications of quasiconvex and integer-convex functions in optimization. We next define a new class of functions that contains the above two functions as subclass.

Definition 5.3 A function $g(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ is *integer-quasiconvex* if $\text{conv}(S^\alpha_{\mathbb{Z}}(g)) \cap \mathbb{Z}^n = S^\alpha_{\mathbb{Z}}(g)$ for any integer-α-level set $S^\alpha_{\mathbb{Z}}(g) := \{\boldsymbol{x} \in \mathbb{Z}^n \,|\, g(\boldsymbol{x}) \leq \alpha\}$. In words, any integer point of the convex hull of the integer-α-level set is feasible to the level set.

The following results show that an integer-quasiconvex function is a generalization of both integer-convex and quasiconvex functions, and that such function satisfies property (i) of Proposition 4.1.

Proposition 5.1 *Consider the function $g(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$.*

(i) If $g(\boldsymbol{x})$ is quasiconvex, then it is integer-quasiconvex.
(ii) If $g(\boldsymbol{x})$ is integer-convex, then it is integer-quasiconvex.
(iii) If $g(\boldsymbol{x})$ is integer-quasiconvex, then for any $\bar{\boldsymbol{x}} \in \mathbb{Z}^n$ such that $g(\bar{\boldsymbol{x}}) > \alpha$, we have that $\bar{\boldsymbol{x}} \notin \text{conv}(S^\alpha_{\mathbb{Z}}(g))$.

Proof (i) We need to show that $S^\alpha_{\mathbb{Z}}(g) = \text{conv}(S^\alpha_{\mathbb{Z}}(g)) \cap \mathbb{Z}^n$. The direct inclusion follows from the fact that $S^\alpha_{\mathbb{Z}}(g) \subseteq \text{conv}(S^\alpha_{\mathbb{Z}}(g))$. For the reverse inclusion, we write that $\text{conv}(S^\alpha_{\mathbb{Z}}(g)) \cap \mathbb{Z}^n \subseteq \text{conv}(S^\alpha(g)) \cap \mathbb{Z}^n = S^\alpha(g) \cap \mathbb{Z}^n = S^\alpha_{\mathbb{Z}}(g)$, where the inclusion follows from the fact that $S^\alpha_{\mathbb{Z}}(g) \subseteq S^\alpha(g)$, the first equality holds because of the definition of quasiconvex functions that $\text{conv}(S^\alpha(g)) = S^\alpha(g)$, and the last equality follows from the description of integer level sets.

(ii) The direct inclusion is similar to the previous part. For the reverse inclusion, consider a point $\bar{\boldsymbol{x}} \in \text{conv}(S^\alpha_{\mathbb{Z}}(g)) \cap \mathbb{Z}^n$. Since $\bar{\boldsymbol{x}} \in \text{conv}(S^\alpha_{\mathbb{Z}}(g))$, there are $\lambda^i \in \mathbb{R}_+$, $\boldsymbol{x}^i \in S^\alpha_{\mathbb{Z}}(g)$ for $i \in \{1, \ldots, t\}$ such that $\sum_{i=1}^t \lambda^i = 1$ and $\sum_{i=1}^t \lambda^i \boldsymbol{x}^i = \bar{\boldsymbol{x}}$. We write that $g(\bar{\boldsymbol{x}}) = g(\sum_{i=1}^t \lambda^i \boldsymbol{x}^i) \leq \sum_{i=1}^t \lambda^i g(\boldsymbol{x}^i) \leq \alpha$, where the equality follows from the construction above, the first inequality follows from the definition of the integer-convex functions as $\bar{\boldsymbol{x}} \in \mathbb{Z}^n$, and the last inequality holds because $g(\boldsymbol{x}^i) \leq \alpha$ as $\boldsymbol{x}^i \in S^\alpha_{\mathbb{Z}}(g)$ by construction. We conclude that $\bar{\boldsymbol{x}} \in S^\alpha_{\mathbb{Z}}(g)$.

(iii) Since $g(\bar{\boldsymbol{x}}) > \alpha$, it follows from the definition of integer level sets and that of integer-quasiconvex functions that $\bar{\boldsymbol{x}} \notin S^\alpha_{\mathbb{Z}}(g) = \text{conv}(S^\alpha_{\mathbb{Z}}(g)) \cap \mathbb{Z}^n$. Since $\bar{\boldsymbol{x}} \in \mathbb{Z}^n$, we conclude that $\bar{\boldsymbol{x}} \notin \text{conv}(S^\alpha_{\mathbb{Z}}(g))$. □

It follows from Proposition 5.1(iii) that if constraints $g^j(\boldsymbol{x})$ of $(\mathcal{E})$ are integer-quasiconvex, then property (i) of Proposition 4.1 is satisfied. As implied by Proposition 4.3, we can replace linearization cuts with DD cuts to satisfy property (ii) of Proposition 4.1. The combination of these two properties gives the following result.

Proposition 5.2 *Assume that all constraints $g^j(\boldsymbol{x})$ of $(\mathcal{E})$ are integer-quasiconvex. Algorithm DD-ECP of Definition 4.1 converges to the optimal solution of $(\mathcal{E})$ or proves that it is infeasible in a finite number of iterations.* □

Proposition 5.2 shows that the outer approximation algorithms can be applied to a more general class of INLPs with the help of DD cuts. Example 5.1 illustrates that the traditional linearization cuts in the outer approximation methods does not provide valid inequalities for integer-convex and quasiconvex functions. Therefore, these outer approximation algorithms cannot be applied to this broader class of INLPs.

Example 5.1 Consider function $g^1(x) = (x-1)^2(x-2)^2$, and assume bounds $[l,u] = [0,3]$. It is clear that $g^1(x)$ is not convex as the line connecting point $x = 1$ and $x = 2$ lies below the function curve. On the other hand, it is easy to verify that $g^1(x)$ is integer-convex according to the definition. It also follows from the definition that $\mathcal{G}^1_{\mathbb{Z}} = \{1,2\}$. Now consider point $\bar{x} = \frac{5}{4}$. The linearization cut at this point is $23 - 16x \leq 0$, which is not valid for $\mathcal{G}^1_{\mathbb{Z}}$ as it is violated at $x = 1$. Consider next the function $g^2(x) = \ln(x)$, and assume bounds $[l,u] = [1,3]$. It is clear that $g^2(x)$ is quasiconvex but not convex. It also follows from the definition that $\mathcal{G}^2_{\mathbb{Z}} = \{1\}$. Now consider point $\bar{x} = 2$. The linearization cut at this point is $2(\ln(2) - 1) + x \leq 0$, which is not valid for $\mathcal{G}^1_{\mathbb{Z}}$.

5.2 General nonconvex functions

In this section, we study an adoption of the DD-ECP algorithm for a general nonconvex INLP. The key idea is to reconsider the role of the two properties of Proposition 4.1 in the convergence of the ECP algorithm. Property (i) ensures that if a point is infeasible, there exists a constraint whose convex hull can separate the point from the ILP relaxation. The fact that this property does not hold for general nonconvex INLPs suggests that cut-generation with respect to individual constraints is not sufficient to separate an infeasible point. Property (ii) however concerns the existence of a scheme to derive valid inequalities for individual constraints. The fact that this property holds for the DD inequalities makes the DD-ECP algorithm a valuable tool to obtain strong relaxations for $(\mathcal{E})$. Our computational experiments in Section **??** suggest that these relaxations can be much stronger than those obtained by the existing methods employed in the state-of-the-art solvers for nonconvex structures. In the remainder of this section, we will pursue the theoretical aspects of this algorithm. First, we adopt the DD-ECP algorithm for nonconvex INLPs in Algorithm 3.

Algorithm 3 DD-ECP algorithm for nonconvex INLP

Initialize $\mathcal{K} \leftarrow \emptyset$, $i \leftarrow 1$, $Flag \leftarrow False$
Solve $(\mathcal{E}^D(\mathcal{K}))$
if $(\mathcal{E}^D(\mathcal{K}))$ is infeasible **then**
    Return infeasibility for $(\mathcal{E})$
else
    Find an optimal solution $\bar{\boldsymbol{x}}^i$ of $(\mathcal{E}^D(\mathcal{K}))$
end if
if $Flag = True$ **then**
    Return $\boldsymbol{c}^\intercal \bar{\boldsymbol{x}}^i$ as a dual bound for $(\mathcal{E})$
else
    $\mathcal{K} \leftarrow \mathcal{K} \cup \{\bar{\boldsymbol{x}}^i\}$, $i \leftarrow i+1$, update $Flag$, go to 2
end if

After the addition of $\bar{\boldsymbol{x}}^i$ to $\mathcal{K}$, $Flag$ is updated as follows. If a new DD inequality $h_j(\bar{\boldsymbol{x}}^i, \boldsymbol{x}) \leq 0$ that is violated at $\bar{\boldsymbol{x}}^i$ is added to $(\mathcal{E}^D(\mathcal{K}))$ for some $j \in \bar{J}(\bar{\boldsymbol{x}}^i)$, we assign $Flag \leftarrow False$, otherwise we assign $Flag \leftarrow True$. In words, if the optimal solution of the current ILP relaxation $(\mathcal{E}^D(\mathcal{K}))$ cannot be cut off with respect to any individual constraints of $(\mathcal{E})$, the algorithm stops and returns the current objective value as the best possible bound that can obtained via individual constraint considerations. Otherwise, the optimal solution is cut off from the feasible region of $(\mathcal{E}^D(\mathcal{K}))$ through

DD inequalities and the steps are repeated. These arguments imply the following convergence results for the DD-ECP algorithm.

Proposition 5.3 *Algorithm 3 converges to* $p^* = \{\boldsymbol{c}^\intercal \boldsymbol{x} \,|\, \boldsymbol{x} \in \cap_{j\in J} \text{conv}(\mathcal{G}^j_{\mathbb{Z}}), \boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}] \cap \mathbb{Z}^n\}$ *or proves that it is infeasible in a finite number of iterations.* □

It is clear that the optimal value p^* provides an upper (dual) bound for the optimal value of $(\mathcal{E})$ as $\text{conv}(\cap_{j\in J}\mathcal{G}^j_{\mathbb{Z}}) \subseteq \cap_{j\in J}\text{conv}(\mathcal{G}^j_{\mathbb{Z}})$. For this reason, unlike the convex case of Section 4 and the integer-quasiconvex case of Section 5.1, the nonconvex DD-ECP does not necessarily return an optimal solution at termination. To achieve optimality, we may use this algorithm in conjunction with a branch-and-bound scheme that partitions the feasible region through bound reduction for variables. For each resulting partition, the DD formulation of the constraints are modified according to the bound reduction, and then the DD-ECP algorithm is invoked. Such bound reduction can be efficiently applied to the existing DD structures by removing (or alternatively setting weights equal to $-\infty$ for) arcs whose labels do not belong to the updated range. As common in convexification techniques, these bound reductions lead to the generation of stronger DD cuts which in turn produce tighter relaxations at the partitions of the problem through branch-and-bound. We refer the reader to [33] and [5] for a detailed account on the convergence of the branch-and-bound schemes through bound reduction and convexification.

When used inside branch-and-bound, the DD-ECP algorithm plays the role of constructing a convex relaxation and then solving it to obtain a bound. Therefore, it is only natural to compare the quality of the bounds obtained from the DD-ECP Algorithm 3 and those obtained from the conventional methods that construct convex relaxations of $(\mathcal{E})$. We address this by considering the most common convexification technique used for the class of *factorable* problems. A factorable problem consists of factorable functions, which can be expressed as a finite recursion of a finite sums and products of univariate functions. Global solvers exploit this property to decompose complicated terms in the problem description into simpler predefined univariate functions through introduction of new variables. Convex relaxations for these predefined functions are readily available and are used by the solvers to automatically construct convex relaxations for the entire problem in a higher dimension.

A weakness of the factorable decomposition technique is that it constructs convex relaxations for each newly formed constraint separately. Since each of these constraints contain a single univariate function, the resulting relaxations involve a small number of the original variables. This leads to the lack of capturing deep interactions between original variables that appear in the initial nonlinear constraints. As a remedy, disjunctive cuts for nonlinear MINLP have been proposed to cut off unwanted points while incorporating deeper interactions between variables; see [14] for a discussion. We refer the interested reader to [5] for a detailed account of using disjunctive cuts for factorable MINLPs.

In addition to the theoretical and computational advantages described in Section 4, the DD-ECP Algorithm 3 mitigates the above weakness of the factorable decomposition techniques. In particular, it derives valid inequalities with respect to each constraint in the space of original variables without decomposing its terms into several constraints with small number of variables. This results in capturing deeper interactions between original variables and thereby producing stronger relaxations. Our computational experiments in Section 6 assert this statement by showing a significant gap improvement achieved by the DD cuts upon the default convex relaxations of the state-of-the-art global solvers.

6 Computational Results

In this section, we conduct a computational study to show the potential of the DD-ECP algorithm for both convex and nonconvex INLPs. These results are obtained on a Windows 10.1 (64-bit)

operating system, 8 GB RAM, 2.20 GHz Core i7 CPU. The DD-ECP Algorithm is written in Julia v1.1 via JuMP v0.19 [20] and the ILP optimization models are solved with CPLEX v12.7.1. The results are compared with those obtained from a collection of the state-of-the-art solvers via GAMS v24.8.5 modeling language.

6.1 Implementation Settings

We first present the basic settings of the DD-ECP algorithm in our experiments.

6.1.1 DD construction

As a key step in applying DD-ECP, we introduce a systematic approach to construct DDs for a given INLP. As discussed in Section 4.2, DDs are constructed based on each constraint of the model individually. When function $g(\boldsymbol{x})$ describing constraint $\mathcal{G}_{\mathbb{Z}} := \{\boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}] \cap \mathbb{Z}^n \,|\, g(\boldsymbol{x}) \leq 0\}$ is separable, the construction of the corresponding DD is based directly on its dynamic programming (DP) formulation, as follows.

Let $g(\boldsymbol{x}) = \sum_{i=1}^{n} g_i(x_i) - c$, and let $\mathcal{D} = (\mathcal{U}, \mathcal{A}, l(.))$ be its corresponding DD. Variable x_i represents arc layer $\mathcal{A}_i$, and its values over the domain represent arc labels $l(a)$ for $a \in \mathcal{A}_i$. The i-th *stage* of the DP model is represented by node layer $\mathcal{U}_i$. The DP *states* at layer i are associated with the nodes in $\mathcal{U}_i$. The state value $S_i(u)$ of a node $u \in \mathcal{U}_i$ denotes the cumulative *resource consumption* level obtained by the variable assignments along the paths from the source node to u. The initial state at the source node is zero. The DP *state transition function* at layer $i \in N$, denoted by $T_i(u, a)$ for $u \in \mathcal{U}_i$ and $a \in \mathcal{A}_i$ where a is an outgoing arc of u, defines the state value of a child node in layer $i + 1$ that is reached from node u through arc a. Since $g(\boldsymbol{x})$ is separable, this transition function is simply computed as $T_i(u, a) = S_i(u) + g_i(l(a))$. For the last layer, the outgoing arc a of node $u \in \mathcal{U}_n$ is included in $\mathcal{A}_n$ only if $T_i(u, a) \leq c$, i.e, the consumption level is no more than the available resource. We refer the reader to Chapter 3 of [9] for a detailed account on constructing DDs based on DP formulation.

When function $g(\boldsymbol{x})$ describing constraint $\mathcal{G}_{\mathbb{Z}}$ is not separable, the construction follows a similar method but can become more tedious. Namely, in this case the computation of the state value of each node is not solely dependent on the variable value on that layer, which necessitates backtracking to multiple previous layers. This backtracking can result in a slow construction procedure depending on the density of non-separable terms in the constraints.

The major challenge in constructing exact DDs is the exponential size growth with the number of variables in $\mathcal{G}_{\mathbb{Z}}$. In practice, whenever the size of an exact DD becomes too large (as indicated by width limit), we may use the idea of relaxed DDs to control its size while obtaining a discrete relaxation of $\mathcal{G}_{\mathbb{Z}}$; see Section 2 for the definition of relaxed DDs. Since DD inequalities remain valid for the original set when derived with respect to relaxed DDs, they can be used in the DD-ECP framework to obtain an outer approximation. To construct a relaxed DD for $\mathcal{G}_{\mathbb{Z}}$, we apply the following immediate procedure: during the construction procedure at a node layer, we may simply round the state value of the nodes down within a predefined decimal precision. For example, if we have three nodes with state values 1.46, 1.233 and 1.9, rounding them down to the closest integer will *merge* them into a single node with state value 1. This reduces the node number in a layer, while providing a relaxation for the feasible region defined by $\mathcal{G}_{\mathbb{Z}}$ since the modified state values under-represent the true value of terms on the left-hand-side of $g(\boldsymbol{x}) \leq 0$. We refer the reader to Chapter 4 of [9] for an in-depth discussion on relaxed DDs.

6.1.2 DD-ECP algorithm

After constructing DDs for each nonlinear constraint in a *preprocessing* phase, we apply the DD-ECP algorithm. We next give implementation details for this algorithm. In all of our experiments,

we apply DD-ECP at the root node and report the best dual bound at termination. On the contrary, the solvers we employ in this section use their default arsenal of branch-and-cut tools including preprocessing, cutting planes, branching, etc. The initial relaxation used in the DD-ECP algorithm is the box relaxation obtained by removing all nonlinear constraints and keeping variable bounds only. Then the optimal solution of the box relaxation is found and a call to Algorithm 2 for convex INLPs or Algorithm 3 for nonconvex INLPs is invoked. We solve each iteration of the DD-ECP algorithm as LP and continue the procedure until the improvement rate of the objective value becomes trivial (fixed at 10^{-3}), after which we switch to solve ILP to improve the dual bound.

At each iteration of the DD-ECP algorithm, we check whether the current optimal solution can be separated from the DDs representing constraints of the model. As discussed in Section 3, we have two options to perform separation. As for the CGLP, in all test instances and for all specified time limits, the algorithm was not able to solve a CGLP to optimality. This is due to the large size of the CGLP corresponding to the constructed DDs. On the other hand, the subgradient method manages to solve hundreds of separation problems and close gaps in a fast and consistent rate as reported. This shows a clear computational advantage of the subgradient method over CGLP. We set a constant step size rule $\rho = 1$, and use origin as the starting point for the subgradient algorithm. We define the termination criterion to be a certain number of iterations as reported in the ensuing experiments.

We set the termination criteria for the DD-ECP algorithm to be the time limit (values are reported for each test instance) and the improvement rate of the objective value (fixed at 10^{-3}). At each iteration of the DD-ECP algorithm, the cuts corresponding to c most violated inequalities are added, where c represents the maximum number of cutting planes per iteration (values are reported for each test instance).

6.2 Pricing problem

The first problem class we study is motivated by a pricing problem in marketing applications. In particular, we consider the problem where a firm seeks to determine the price of several new products to enter a competitive market. To form a basic model, we use the following strategies.

The first strategy is the *psychological pricing*, which implies that certain prices have more significant psychological impacts on customers' behavior. According to a study [27], approximately 97% of product prices in advertisements end in digits 0, 5, or 9. Using this observation, we may discretize the price space so that price variables are integers within a range, and then rescale them to achieve the desired numeral precision. We define $x_i \in [l_i, u_i] \cap \mathbb{Z}$ to be the price of product i. The next important factor in marketing models is the demand function in terms of price. To define an appropriate demand function, we consider the *fairness effect*, which implies that customers are more sensitive to higher prices in a competitive market due to their perception of fairness in their purchase context; see [30]. This definition induces the popular exponential decay function of the form $e^{-x_i^{k_i}}$ as the demand function, where k_i captures the price sensitivity of product i. The third strategy is the *penetration pricing*, which is popular for startup companies that enter a highly competitive market; see [19]. The goal is to set the price low initially to gain customer attention and market share. Once the desired share is secured, the price gradually increases to make profit. To model this strategy, we define the objective function to be minimizing a nonnegative weighted combination of product prices, i.e., $\min \sum_{i=1}^{n} c_i x_i$. Coupled with the penetration pricing strategy is a *profit satisfaction* strategy. While the objective is to set the price of products low, there must be a limit for these low prices to prevent an excessive loss. To model this restriction, we compute the profit function as the product of price and demand, i.e., $g_i(x_i) = x_i e^{-x_i^{k_i}}$ for each i. The profit satisfaction constraint is then modeled by $\sum_{i=1}^{n} a_i x_i e^{-x_i^{k_i}} \geq b$, where a_i represents a nonnegative weight and b denotes the profit margin. We consider multiple constraints of this form to capture

the fact that each major geographical district of the firm can have different demand trends to be taken into account. The collection of the above strategies give the following pricing model.

$$\min \quad \sum_{i=1}^{n} c_i x_i \tag{6.1a}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} a_i^j x_i e^{-x_i^{k_i^j}} \geq b_j, \qquad \forall j \in J \tag{6.1b}$$

$$\boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}] \cap \mathbb{Z}^n. \tag{6.1c}$$

It is easy to verify that the above pricing model is of the form ($\mathcal{E}$), studied in Section 5, with a suitable change of sign in the objective and constraints. Notice that the profit function $g_i(x_i) = x_i e^{-x_i^{k_i}}$ over the discrete space of variables' domain is integer-quasiconvex; see Section 5.1. Since constraints (6.1b) are nonconvex, we use Algorithm 3. We evaluate the performance of our method by comparing the bounds obtained from the DD-ECP algorithm with those obtained from the state-of-the-art global solvers ANTIGONE, BARON, COUENNE and SCIP. Implementation details are set as outlined in Section 6.1. We set the default problem size $n = 200$, time limit $t = 300$ seconds, maximum number of cuts per iteration $c = 3$, maximum iteration number for subgradient algorithm $s = 20$, and width limit for DD $w = 5000$. To investigate the marginal impact of these factors on the solution performance, we consider three categories of different values for each factor as reported in Tables 6.1 –6.5.

For each category, we consider 5 randomly generated instances with the following characteristics. We set the number of the profit satisfaction constraints $|J| = 5$. We set the bounds on integer variables $[l_i, u_i] = [0, 10]$, and fix the scaling factor 10, i.e., the prices are chosen among $\{0, 0.1, 0.2, \ldots, 1.0\}$. The objective coefficients c_i are generated randomly from a uniform discrete distribution (u.d.d.) between $[0, 20]$, and constraint coefficients a_i^j are generated randomly from a u.d.d. between $[0, 100]$. The right-hand-side values b_j are randomly generated from a u.d.d. between $[10n, 20n]$ where n is the number of variables. The monomial degree of the exponent of the exponential demand k_i^j is randomly selected from $\{1, 2, 3\}$.

In Table 6.1, we evaluate the performance of DD-ECP as compared to global solvers for three categories of problem sizes, $n = 50$, $n = 200$, and $n = 500$. For these problem sizes, we construct relaxed DDs to satisfy the default width limit; see Section 6.1.1 for relaxation technique. In all instances, the invoked termination criterion has been the time limit. Column "#" represents instance number in a category. Column "UB" contains the best primal bound obtained across all solvers within the default time limit. The dual bound obtained from the DD-ECP algorithm at termination is reported in column "DD LB". Columns "ECP #" and "cut #" show the number of DD-ECP iterations and the number of DD cuts added to the model at termination, respectively. The time it takes to construct DDs for the constraints of the model is reported in column "Time". Columns "ATGN", "BARN", "COUN" and "SCIP" contain the dual bound obtained at termination by solvers ANTIGONE, BARON, COUENNE and SCIP, respectively. Columns "Δ_1", "Δ_2", "Δ_3" and "Δ_4" report the *relative* gap improvement achieved by DD-ECP algorithm with respect to each solver in the order above. For instance, this value is computed as $\frac{\text{DD LB - ATGN}}{\text{UB - ATGN}}$ for ANTIGONE. As observed in Table 6.1, the bound obtained by the DD-ECP algorithm achieves a significant gap improvement upon those obtained by the solvers for all problem sizes. These results assert the conclusion in Section 5.2 that the DD-ECP algorithm can provide a tighter relaxation than those obtained from the default convexification of factorable functions used by global solvers.

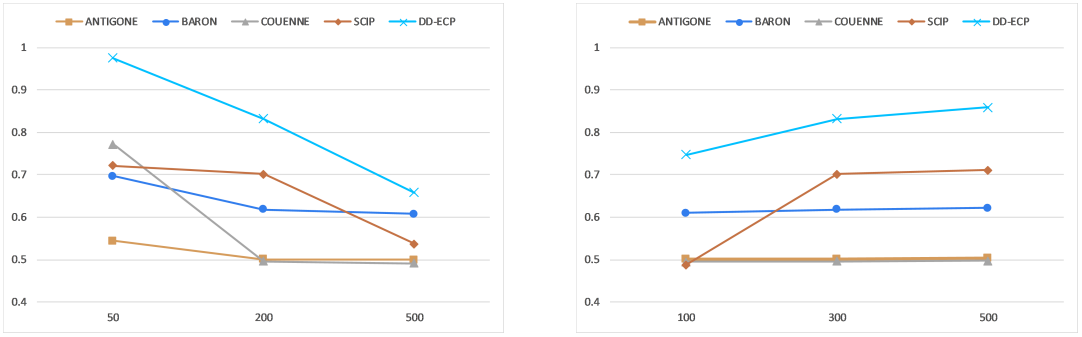
Figure 6.1a illustrates the *absolute* gap improvement rate for different problem sizes. To this end, the average of absolute gap closure for each algorithm is computed along all instances of each size category. For instance, the absolute gap closure is computed as $\frac{\text{UB - ATGN}}{\text{UB}}$ for ANTIGONE. While the DD-ECP outperforms all solvers, the improvement rate decreases as the problem size

increases. This is due to the fact that for a fixed width limit, larger DD sizes result in weaker relaxations as common in standard convexification techniques.

In Table 6.2, we investigate the marginal impact of varying time limit on solution performance. We solve each instance of size $n = 200$ described in Table 6.1 with time limits 100, 300 and 500 seconds. Column values are defined similarly to those of Table 6.1. Figure 6.1b shows the absolute gap closure at various time limits. Even though increasing time limit does not have a positive impact on improving solvers' results, it improves the gap percentage for the DD-ECP algorithm thanks to the consistent effectiveness of the DD cuts.

Next, we perform a sensitivity analysis on important algorithmic parameters of DD-ECP. We solve instances of size $n = 200$ with time limit $t = 300$ for each parameter choice as presented in Tables 6.3–6.5. In these tables, column "Δ" represents the absolute gap closure achieved by the DD-ECP algorithm. Other columns are defined similarly to those of the previous tables. Table 6.3 shows the marginal impact of width limit on solution performance by considering three width limit choices 2000, 3000 and 5000. Increasing width limit increases the size of the graph representing DD, which leads to a stronger relaxation at the price of increasing the construction and cut generation time. The values reported in Table 6.3 show this trade-off. Despite this trade-off, the results of Table 6.3 and Figure 6.2a imply that the gap percentage improves with the width limit, outweighing the computational burden.

Table 6.4 and Figure 6.2b present the sensitivity analysis on the maximum number of iterations in the subgradient method by considering three values 5, 20 and 40. Increasing the number of iterations in the subgradient method leads to detecting stronger cuts at the price of increasing the cut generation time. Similarly, Table 6.5 and Figure 6.2c illustrate the sensitivity analysis on the maximum number of DD cuts that can be added at each iteration of the DD-ECP by considering three values 1, 3 and 5. Increasing the number of DD cuts results in a tighter relaxation at each iteration at the price of slowing down the resolve process. As observed in Figures 6.2b and 6.2c, none of the above choices fully dominate others due to the aforementioned trade-offs.



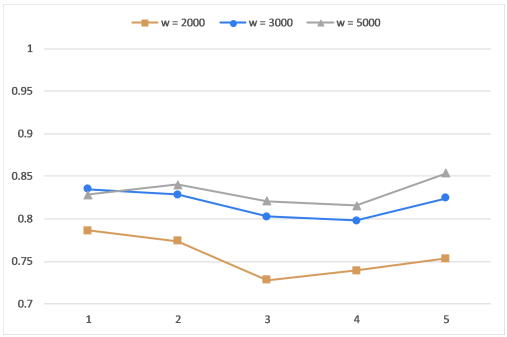
(a) Gap closure vs problem size

(b) Gap closure vs time limit

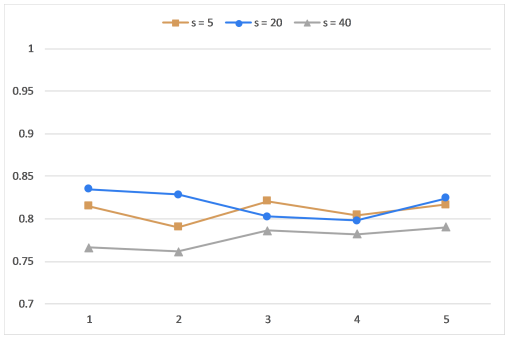
Fig. 6.1: DD-ECP performance for the pricing problem

6.3 Polynomial knapsack problem

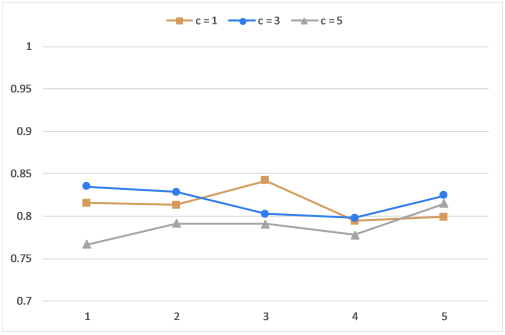
In this section, we study a general polynomial convex INLP, where the nonlinear functions are of the knapsack form. Consider $(\mathcal{E})$ described in Section 4 with separable functions $g^j(\boldsymbol{x})$ that are defined as $g^j(\boldsymbol{x}) := \sum_{i=1}^{n} g_i^j(x_i) - b_j$, where $g_i^j(x_i) = a_i^j x_i^{k_i^j}$. We evaluate the performance of the DD-ECP algorithm by comparing it with the state-of-the-art solvers that use outer approximation



(a) Width limit



(b) Subgradient iterations



(c) DD cut number

Fig. 6.2: Sensitivity analysis for the pricing problem

Table 6.1: Size comparison for the pricing problem

Size	#	UB	DD LB	ECP #	cut #	Time	ATGN	Δ_1	BARN	Δ_2	COUN	Δ_3	SCIP	Δ_4
50	1	1592	1548.4	286	754	22.4	986.0	93%	1270.0	86%	1289.7	86%	1260.3	87%
	2	1825	1786.3	313	643	40.0	1063.3	95%	1299.0	93%	1452.5	90%	1367.2	92%
	3	1891	1846.8	287	713	16.9	1076.9	95%	1408.0	91%	1512.7	88%	1391.8	91%
	4	2403	2343.9	308	667	24.0	1178.8	95%	1484.0	94%	1766.2	91%	1576.6	93%
	5	1800	1753.1	279	678	27.6	824.4	95%	1100.0	93%	1295.4	91%	1218.7	92%
200	1	5893	4883.4	323	966	58.8	3237.7	62%	3903.0	49%	3224.3	62%	4360.3	34%
	2	6264	5262.8	349	1044	48.6	3389.9	65%	4327.0	48%	3374.5	65%	4678.2	37%
	3	8055	6610.5	350	1047	49.7	3559.6	68%	4364.0	61%	3525.3	68%	5075.7	52%
	4	6997	5704.0	383	1144	50.9	3399.4	64%	4170.0	54%	3353.3	65%	4920.7	38%
	5	8299	7084.6	455	1362	44.1	4004.2	72%	4951.0	64%	3959.3	72%	5700.4	53%
500	1	24539	16108.1	186	555	60.6	12631.9	29%	15384.0	8%	12489.6	30%	12203.6	32%
	2	19898	12988.7	185	551	87.9	9427.5	34%	11721.0	16%	9237.5	35%	11716.1	16%
	3	16106	10856.9	163	486	90.5	8678.3	29%	10429.0	8%	8556.2	30%	8399.3	32%
	4	21398	13808.9	189	564	79.4	11205.1	26%	13457.0	4%	11049.1	27%	10777.2	29%
	5	21524	14197.4	175	520	92.7	9583.7	39%	11704.0	25%	9366.0	40%	12318.6	20%

schemes as reported in [13]. Since the marginal impact of varying parameters such as time limit, width limit, subgradient iterations and DD cut number follow a similar pattern as those established in Section 6.2, we present the computational results for different problem sizes only as they can lead to drastic differences depending on the problem structure. In particular, we fix the default time limit $t = 300$, width limit $w = 3000$, subgradient iteration $s = 20$, and DD cut number $c = 3$. We consider two problem sizes with $n = 100$ and $n = 500$. We generate 5 random instances for

Table 6.2: Time comparison for the pricing problem

Time	#	UB	DD LB	ECP #	cut #	ATGN	Δ_1	BARN	Δ_2	COUN	Δ_3	SCIP	Δ_4
100	1	5893	4292.7	44	128	3237.7	40%	3822.0	23%	3208.6	40%	3157.3	42%
	2	6264	4426.0	55	160	3389.9	36%	4267.0	8%	3372.6	36%	3303.2	38%
	3	8055	6264.2	50	145	3559.6	60%	4321.0	52%	3516.9	61%	3464.2	61%
	4	6997	5382.7	59	173	3399.4	55%	4134.0	44%	3346.9	56%	3305.5	56%
	5	8299	6278.7	89	263	4004.2	53%	4903.0	41%	3949.2	54%	3899.8	54%
300	1	5893	4883.4	323	966	3237.7	62%	3903.0	49%	3224.3	62%	4360.3	34%
	2	6264	5262.8	349	1044	3389.9	65%	4327.0	48%	3374.5	65%	4678.2	37%
	3	8055	6610.5	350	1047	3559.6	68%	4364.0	61%	3525.3	68%	5075.7	52%
	4	6997	5704.0	383	1144	3399.4	64%	4170.0	54%	3353.3	65%	4920.7	38%
	5	8299	7084.6	455	1362	4004.2	72%	4951.0	64%	3959.3	72%	5700.4	53%
500	1	5893	5070.2	471	1405	3238.0	69%	3933.0	58%	3232.3	69%	4407.0	45%
	2	6264	5440.4	524	1567	3398.5	71%	4354.0	57%	3381.3	71%	4743.5	46%
	3	8055	6794.3	531	1583	3614.6	72%	4397.0	66%	3541.3	72%	5142.5	57%
	4	6997	5865.9	567	1698	3418.2	68%	4200.0	60%	3361.4	69%	4972.6	44%
	5	8299	7315.4	700	2097	4015.9	77%	4997.0	70%	3967.1	77%	5803.4	61%

Table 6.3: Sensitivity analysis of width limit for the pricing problem

Width	#	UB	DD LB	Δ	ECP #	cut #	Time
2000	1	5893	4632.5	79%	733	2110	27.0
	2	6264	4848.2	77%	649	1944	27.1
	3	8055	5862.4	73%	670	2003	28.6
	4	6997	5173.6	74%	657	1968	29.4
	5	8299	6252.4	75%	675	2022	26.4
3000	1	5893	4919.4	83%	527	1566	35.0
	2	6264	5189.6	83%	501	1500	34.2
	3	8055	6464.4	80%	528	1579	35.5
	4	6997	5583.3	80%	542	1623	34.2
	5	8299	6842.0	82%	584	1747	33.4
5000	1	5893	4883.4	83%	323	966	58.8
	2	6264	5262.8	84%	349	1044	48.6
	3	8055	6610.5	82%	350	1047	49.7
	4	6997	5704.0	82%	383	1144	50.9
	5	8299	7084.6	85%	455	1362	44.1

each problem size as follows. The number of the polynomial knapsack constraints is set as $|J| = 5$. We set the bounds on integer variables $[l_i, u_i] = [0, 5]$. The objective coefficients c_i and constraint coefficients a_i^j are generated randomly from a u.d.d. between $[0, 50]$. The right-hand-side values b_j are randomly generated from a u.d.d. between $[1000, 5000]$. The degree k_i^j of each monomial $g_i^j(x_i)$ is randomly generated from a u.d.d. between $[1, 10]$.

To evaluate the strength of the DD cuts, we use the DD-ECP algorithm of Definition 4.1 to solve $(\mathcal{E})$. Table 6.6 compares the bounds obtained from the DD-ECP algorithm with those obtained from solvers at their default settings. These bounds are reported in columns "AECP" for α-ECP solver, "BMNM" for BONMIN, "DCPT" for DICOPT and "SBB" for SBB. Other columns share similar definition as those of Table 6.1. Symbol "–" indicates that the solver was not able to return a dual bound during the solution time. Figure 6.3 shows the absolute gap closure obtained for all instances of the two problem sizes.

Table 6.4: Sensitivity analysis of subgradient iterations for the pricing problem

subgrad	#	UB	DD LB	Δ	ECP #	cut #
5	1	5893	4800.6	81%	462	1269
	2	6264	4951.6	79%	462	1382
	3	8055	6610.4	82%	487	1312
	4	6997	5626.6	80%	489	1462
	5	8299	6779.3	82%	498	1491
20	1	5893	4919.4	83%	527	1566
	2	6264	5189.6	83%	501	1500
	3	8055	6464.4	80%	528	1579
	4	6997	5583.3	80%	542	1623
	5	8299	6842.0	82%	584	1747
40	1	5893	4513.4	77%	109	324
	2	6264	4769.6	76%	116	345
	3	8055	6330.7	79%	122	361
	4	6997	5473.5	78%	125	371
	5	8299	6558.5	79%	168	501

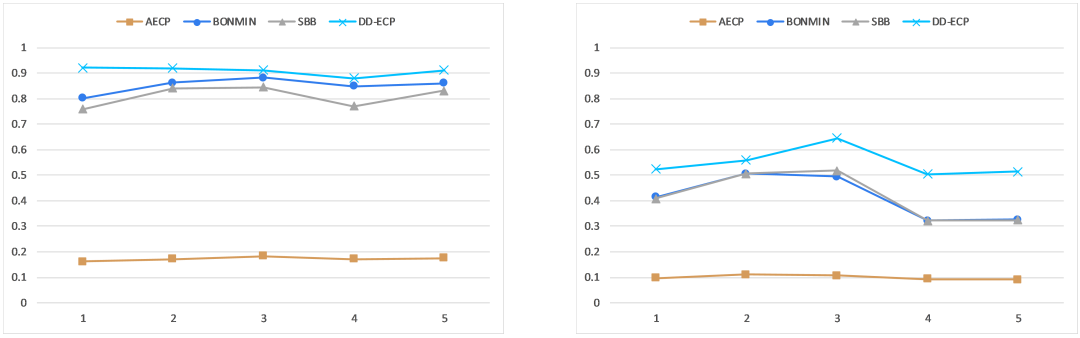
Table 6.5: Sensitivity analysis of DD cut numbers for the pricing problem

Cut #	#	UB	DD LB	Δ	ECP #	cut #
1	1	5893	4805.3	82%	662	662
	2	6264	5095.2	81%	651	651
	3	8055	6779.3	84%	782	782
	4	6997	5558.7	79%	682	682
	5	8299	6633.2	80%	746	746
3	1	5893	4919.4	83%	527	1566
	2	6264	5189.6	83%	501	1500
	3	8055	6464.4	80%	528	1579
	4	6997	5583.3	80%	542	1623
	5	8299	6842.0	82%	584	1747
5	1	5893	4518.2	77%	124	613
	2	6264	4956.0	79%	148	651
	3	8055	6368.2	79%	148	657
	4	6997	5444.3	78%	132	632
	5	8299	6764.2	82%	288	897

As observed in Table 6.6 and Figure 6.3, the DD-ECP algorithm outperforms other solvers in both problem size categories. Note that solvers like BONMIN, DICOPT and SBB start from the continuous convex relaxation of the problem which provides an initial relaxation much stronger than the box relaxation used in the DD-ECP method. Despite this disadvantage for the DD-ECP algorithm compared to these solvers, it still manages to consistently improve the bounds obtained by these solvers.

Table 6.6: Gap improvement achieved by the DD-ECP algorithm for the polynomial knapsack

Size	#	LB	DD UB	ECP #	cut #	Time	AECP	Δ_1	BNMN	Δ_2	DCPT	SBB	Δ_3
100	1	1905	2054.8	440	1131	16.7	11746.1	98%	2284.0	60%	2402.9	2363.4	67%
	2	2082	2249.7	382	1118	14.7	12075.6	98%	2369.2	42%	–	2411.9	49%
	3	2202	2398.5	402	1184	20.5	12038.5	98%	2462.7	25%	–	2541.5	42%
	4	1835	2056.5	215	600	32.4	10701.3	98%	2113.3	20%	–	2254.8	47%
	5	1897	2064.5	410	1134	21.2	10832.2	98%	2159.6	36%	–	2217.5	48%
500	1	5839	8615.0	145	430	88.9	59966.0	95%	9261.6	19%	–	9289.5	20%
	2	7192	10359.0	114	339	106.4	64459.0	94%	10739.1	11%	–	10742.0	11%
	3	6191	8390.5	107	317	107.4	58065.0	96%	9325.0	30%	–	9171.3	26%
	4	5981	8945.7	132	392	92.1	63711.4	95%	10030.3	27%	–	10033.4	27%
	5	5629	8366.4	121	358	94.2	61103.5	95%	9417.9	28%	–	9433.3	28%



(a) Gap closure for $n = 100$

(b) Gap closure for $n = 500$

Fig. 6.3: DD-ECP performance for the polynomial knapsack

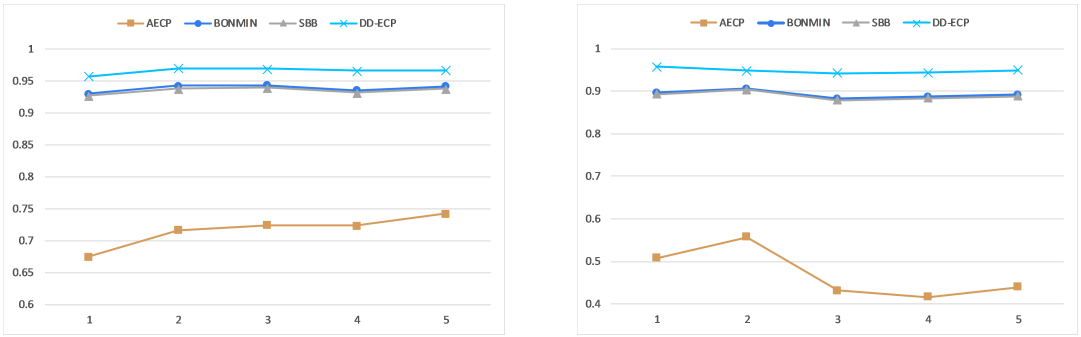
6.4 Ellipsoid problem

In this section, we study a test problem adapted from the benchmark instance **ball_mk3_30** available in the Library of Mixed Integer Nonlinear Programming Instances (MINLPLib) [35]. This convex INLP maximizes a linear function over a *vertical* ellipse in a discrete space. The ellipsoid constraint is of the form $(\mathcal{E})$ described in Section 4 where $g(\boldsymbol{x}) := \sum_{i=1}^{n} \frac{(x_i - 0.5)^2}{r_i^2} - 1$. Similarly to the analysis in Section 6.3, we compare the performance of the DD-ECP algorithm with outer approximation-based solvers. We consider two problem sizes with $n = 500$ and $n = 1000$. We fix the default time limit $t = 300$, width limit $w = 4000$ for $n = 500$, and $w = 2000$ for $n = 1000$, subgradient iteration $s = 20$, and DD cut number $c = 1$. We generate 5 random instances for each problem size as follows. The bounds on integer variables are set as $[l_i, u_i] = [-1, 1]$. The objective coefficients c_i are generated randomly from a u.d.d. between $[-50, 50]$. The directional radius values r_i are randomly generated from a u.d.d. between $[10, 20]$.

Table 6.7 shows the performance of the DD-ECP algorithm in comparison with the solvers. The absolute gap closure achieved by each algorithm is outlined in Figure 6.4 for both problem sizes. Similarly to the previous instances, we observe a consistent dominance of the DD-ECP algorithm among all solvers. In contrast to the polynomial knapsack problem studied in Section 6.3, the computational results suggest that the gap improvement rate of the DD-ECP algorithm remains consistent across different problem sizes. This can attribute to the presence of a single nonlinear constraint in the model.

Table 6.7: Gap improvement achieved by the DD-ECP algorithm for the ellipsoid problem

Size	#	LB	DD UB	ECP #	cut #	Time	AECP	Δ_1	BNMN	Δ_2	DCPT	SBB	Δ_3
500	1	8293	8643.2	428	428	22.6	10988.7	87%	8869.3	39%	–	8900.3	42%
	2	8489	8746.0	422	422	25.7	10895.5	89%	8978.7	48%	–	9012.2	51%
	3	8773	9042.6	421	421	25.8	11192.2	89%	9269.8	46%	–	9300.2	49%
	4	8764	9057.7	417	417	26.2	11184.5	88%	9329.3	48%	–	9361.3	51%
	5	9056	9355.8	410	410	24.2	11388.6	87%	9580.2	43%	–	9614.2	46%
1000	1	14585	15202.1	240	240	29.6	21773.8	91%	16120.6	60%	–	16148.0	61%
	2	15424	16222.8	78	78	30.3	22258.6	88%	16886.3	45%	–	16914.3	46%
	3	13608	14386.6	84	84	31.4	21341.5	90%	15232.7	52%	–	15260.2	53%
	4	13101	13844.7	90	90	29.0	20753.0	90%	14594.8	50%	–	14621.7	51%
	5	13568	14251.6	86	86	29.0	21171.9	91%	15055.1	54%	–	15083.0	55%



(a) Gap closure for $n = 500$

(b) Gap closure for $n = 1000$

Fig. 6.4: DD-ECP performance for the ellipsoid problem

6.5 Empty ball problem

As our last experiment, we study another benchmark problem structure adapted from **ball_mk4_15** in the MINLPLib. This problem models a *rotated* ellipse that does not contain an integer point. The ellipsoid constraint is of the form $\sum_{(i,j)\in K}(x_i + x_j + 0.5)^2 \leq \frac{|K|}{4} - 1$ where K is a set of pair of variable indices. Even though this ellipsoid function is not separable, we can still apply the DD construction technique as discussed in Section 6.1.1. The goal is to test the ability of solvers to detect the infeasibility of this model. We fix the default time limit $t = 300$. We consider two problem sizes with $n = 500$ and $n = 1000$, and generate 5 random instances for each problem size as follows. The bounds on integer variables are set as $[l_i, u_i] = [-10, 10]$. The objective coefficients c_i are generated randomly from a u.d.d. between $[-10, 10]$. The set K of pair of variable indices is generated randomly with size $|K| = n$.

Table 6.8 shows the computational results. Symbol "INF" in column "LB" indicates that the problem is infeasible. This symbol in column "DD UB" indicates that the DD-ECP algorithm has concluded that the problem is infeasible. Column "Time" contains the time it takes for the DD-ECP algorithm to detect infeasiblity. This occurs at the construction stage during preprocessing. Next four columns include the dual bounds obtained by the solvers at termination. In words, none of the solvers were able to detect infeasibility in 300 seconds. This experiment shows another advantageous feature of the DD-ECP algorithm which is conducting an initial analysis on constraints' structure during the construction stage in preprocessing. This can tighten variables' domain by eliminating values that lead to infeasible solutions.

Table 6.8: Gap improvement achieved by the DD-ECP algorithm for the empty ball problem

Size	#	LB	DD UB	Time	AECP	BNMN	DCPT	SBB
500	1	INF	INF	9.0	22474.1	6184.5	6196.0	6196.7
	2	INF	INF	17.0	21098.9	6511.7	–	6527.8
	3	INF	INF	12.0	22245.8	6289.6	6302.0	6302.3
	4	INF	INF	12.2	20337.9	6341.0	–	6355.9
	5	INF	INF	13.0	22147.2	6959.0	6971.0	6971.6
1000	1	INF	INF	22.4	44940.8	13184.2	13193.0	13193.6
	2	INF	INF	20.0	45446.5	13353.5	13358.0	13358.4
	3	INF	INF	18.5	43033.7	12197.9	12209.0	12209.7
	4	INF	INF	19.8	43424.6	12977.2	–	12988.0
	5	INF	INF	19.4	45130.8	12302.1	–	12313.0

7 Conclusion

We use the DD representation of discrete optimization problems to derive valid inequalities for the convex hull of the feasible region described by nonlinear constraints. We introduce two methods to generate valid inequalities, one based on a linear program, and the other based on a subgradient algorithm. The subgradient algorithm has the practical advantage of generating inequalities fast through solving a sequence of longest path problems over the directed acyclic graph induced by the DD. We then design an outer approximation method that uses these DD inequalities to solve INLPs. Traditional outer approximation methods have been structurally designed for convex MINLPs, and fail to solve nonconvex models as the linearization cuts are not valid for nonconvex constraints. Substituting linearization cuts with DD cuts, we adopt the outer approximation methods for nonconvex INLPs. Thanks to several advantages of the DD inequalities, our proposed method gives stronger relaxations for both convex and nonconvex INLPs. Our computational experiments support this statement by showing a consistent gap improvement achieved by the DD cuts when compared to current solvers.

Acknowledgement

We thank Nick Sahinidis and Michael Bussieck for providing GAMS license for our experiments. We also thank the anonymous referees and the Associate Editor for their helpful comments that contributed to improving the paper.

References

1. Andersen HR, Hadžić T, Hooker JN, Tiedemann P (2007) A constraint store based on multivalued decision diagrams. In: Bessiére C (ed) Principles and Practice of Constraint Programming CP 2007, vol 4741, Springer, pp 118–132
2. Balas E (1979) Disjunctive programming. Annals of Discrete Mathematics 5:3–51
3. Balas E (1985) Disjunctive programming and a hierarchy of relaxations for discrete optimization problems. SIAM Journal of Discrete Mathematics 6:466–486
4. Behle M (2007) Binary decision diagrams and integer programming. PhD thesis, Max Planck Institute for Computer Science
5. Belotti P, Lee J, Liberti L, Margot F, Wächter A (2009) Branching and bounds tightening techniques for non-convex minlp. Optimization Methods and Software 24:597–634

6. Belotti P, Kirches C, Leyffer S, Linderoth J, Luedtke J, Mahajan A (2013) Mixed-integer nonlinear optimization. Acta Numberica 22:1–131
7. Bergman D, Cire AA (2018) Discrete nonlinear optimization by state-space decompositions. Management Science
8. Bergman D, Cire AA, van Hoeve WJ, Hooker J (2013) Optimization bounds from binary decision diagrams. INFORMS Journal on Computing 26:253–268
9. Bergman D, Cire AA, van Hoeve WJ, Hooker J (2016) Decision Diagrams for Optimization. Springer International Publishing
10. Bergman D, Cire AA, van Hoeve WJ, Hooker J (2016) Discrete optimization with decision diagrams. INFORMS Journal on Computing 28:47–66
11. Bertsekas DP (1999) Nonlinear Programming. Athena Scientific
12. Bixby RE (2012) A brief history of linear and mixed integer programming computation. Documenta Mathematica pp 107–121
13. Bonami P, Kilinç M, Linderoth J (2012) Algorithms and software for convex mixed integer nonlinear programs. In: Lee J, Leyffer S (eds) Mixed Integer Nonlinear Programming. The IMA Volumes in Mathematics and its Applications, vol 154, Springer
14. Bonami P, Linderoth JT, Lodi A (2012) Disjunctive cuts for mixed integer nonlinear programming problems
15. Boukouvalaa F, Misener R, Floudas CA (2016) Global optimization advances in mixed-integer nonlinear programming, MINLP, and constrained derivative-free optimization, CDFO. European Journal of Operations Research 252:701–727
16. Burer S, Letchford AN (2012) Non-convex mixed-integer nonlinear programming: A survey. Surveys in Operations Research and Management Science 17:97–106
17. Ciré AA, van Hoeve WJ (2013) Multivalued decision diagrams for sequencing problems. Operations Research 61:1411–1428
18. Conforti M, Cornuéjols G, Zambelli G (2014) Integer Programming. Springer
19. Dean J (1976) Pricing policies for new products. Harvard Business Review 54:141–153
20. Dunning I, Huchette J, Lubin M (2017) Jump: A modeling language for mathematical optimization. SIAM Review 59(2):295–320, DOI 10.1137/15M1020575
21. Duran MA, Grossmann I (1986) An outer-approximation algorithm for a class of mixed-integer nonlinear programs. Mathematical Programming 36:307–339
22. Eppstein D (2005) Quasiconvex programming. In: Goodman JE, Pach J, Welzl E (eds) Combinatorial and Computational Geometry, vol 52, MSRI Publications, pp 287–331
23. Garey MR, Johnson DS (1979) Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company
24. Geoffrion A (1985) Generalized benders decomposition. Journal of Optimization Theory and Applications 10:237–260
25. Hadžić T, Hooker JN (2006) Discrete global optimization with binary decision diagrams. In: Workshop on Global Optimization: Integrating Convexity, Optimization, Logic Programming, and Computational Algebraic Geormetry (GICOLAG)
26. Hemmecke R, Köppe M, Lee J, Weismantel R (2009) Nonlinear integer programming. In: Jünger M, Liebling TM, Naddef D, Nemhauser GL, Pulleyblank WR, Reinelt G, Rinaldi G, Wolsey LA (eds) 50 Years of Integer Programming 1958-2008, Springer, pp 561–618
27. Holdershaw J, Gendall P, Garland R (1997) The widespread use of odd pricing in the retail sector. Marketing Bulletin 8:53–58
28. Kronqvist J, Lundell A, Westerlund T (2016) The extended supporting hyperplane algorithm for convex mixed-integer nonlinear programming. Journal of Global Optimization 64:249–272
29. Lee CY (1959) Representation of switching circuits by binary-decision programs. Bell Systems Technical Journal 38:985–999
30. Nagle T, Hogan J, Zale J (2011) The Strategy and Tactics of Pricing: A Guide to Growing More Profitably. Prentice Hall

31. Quesada I, Grossmann IE (1992) An LP/NLP based branchandbound algorithm for convex MINLP optimization problems. Computers and Chemical Engineering 16:937–947
32. St-Aubin R, Hoey J, Boutilier C (2000) Approximation policy construction using decision diagrams. In: Proceedings of Conference on Neural Information Processing Systems, Nantes, France, pp 1089–1095
33. Tawarmalani M, Sahinidis NV (2002) Convexification and global optimization in continuous and mixed-integer nonlinear programming: Theory, algorithms, software, and applications. Kluwer Academic Publishers
34. Tjandraatmadja C, van Hoeve WJ (2018) Target cuts from relaxed decision diagrams. INFORMS Journal on Computing
35. Vigerske S (2019) A library of mixed-integer and continuous nonlinear programming instances. `https://minlplib.org`
36. Wegener I (2000) Branching programs and binary decision diagrams: theory and applications. Society for Industrial and Applied Mathematics
37. Westerlund T, Pettersson F (1995) A cutting plane method for solving convex MINLP problems. Computers and Chemical Engineering 19:s131–s136