Strong Relaxations for Continuous Nonlinear Programs Based on Decision Diagrams

Danial Davarnia[a,*]

[a]*Department of Industrial and Manufacturing Systems Engineering, Iowa State University, Ames, IA, USA*

Abstract

Decision Diagrams (DDs) have risen as a powerful tool to solve discrete optimization problems. The extension of this emerging concept to continuous problems, however, has remained a challenge. In this paper, we introduce a novel framework that utilizes DDs to model continuous nonlinear programs. This framework, when combined with the array of techniques available for discrete problems, illuminates a new pathway to solving mixed integer nonlinear programs with the help of DDs.

Keywords: Decision diagrams, Outer approximation, Mixed integer and nonlinear programs, Cutting planes.

1. Introduction

Since their reintroduction in 2006 [10], DDs have been primarily used to model discrete optimization problems. The integrality requirement is due to the graphical structure of DDs where variable values are represented through arcs. Because the number of arcs needs to be finite in DD representations, they can model explicitly only a finite set of variable values; see [4] for a detailed account on the structure of DDs.

Exploiting graphical properties of DDs has led to specialized relaxation and branch-and-bound techniques that mitigate the dimensionality growth of the

*Corresponding author
Email address: davarnia@iastate.edu (Danial Davarnia)

representation, making DDs a viable solution technique for a variety of combinatorial problems; see [5]. Areas of application include constraint programming [1], scheduling [7], and healthcare [3]. DDs have also been used to generate cutting planes to be embedded in classical branch-and-cut methods. [2] and [13] derive cutting planes for certain classes of linear integer programs. Expanding upon these techniques, [9] proposes an outer approximation framework to derive valid inequalities for integer nonlinear programs of general structure.

In this paper, we extend the DDs role in providing valid inequalities to families of continuous nonconvex programs. We refer the reader to [12, 8] for techniques to construct convex relaxations for nonconvex structures. An effective solution technique to solve nonconvex programs is to iteratively construct outer approximations, often linear, to enclose the feasible region of the original problem. We refer the reader to [6] for a survey on outer approximation methods and to [11] for a recent review of solvers using such techniques.

While various DD-based solution methods and modeling strategies have been proposed in the literature that outperform alternative approaches to solve discrete optimization problems, a successful utilization of DDs in modeling continuous problems has been lacking. In this paper, we undertake this task by introducing a novel framework that uses DDs to construct a specialized relaxation of continuous problems, coined as *arc-reduced* relaxation. This relaxation is then used to derive cutting planes that form a linear outer approximation for the original set. Our approach lifts the integrality barriers on the application domain of DDs by providing a modeling paradigm for general mixed integer nonlinear programs.

The remainder of this paper is organized as follows. Section 2 contains a brief background on DDs. In Section 3, we introduce a new relaxation concept for DDs, and apply it to continuous models. Then, we give an algorithm to construct such relaxed DDs. Further, we propose a framework that uses the constructed relaxed DDs as a basis to generate linear outer approximations. In Section 4, we present preliminary computational experiments on a nonconvex pricing model.

2. Background

In this section, we give a brief review on the basics of DDs for optimization as relevant to our study. A comprehensive review can be found in [4].

2.1. Definitions

A DD is a top-down directed acyclic multi-graph composed of $n+1$ *node layers* denoted by $\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_{n+1}$, where $\mathcal{U}_1$ contains a single *source* node s, and $\mathcal{U}_{n+1}$ contains a single *terminal* node t. The collection of arcs connecting nodes of layer $\mathcal{U}_i$ to $\mathcal{U}_{i+1}$ forms an *arc layer*, denoted by $\mathcal{A}_i$, for $i \in N := \{1, \ldots, n\}$. Each arc a of the DD is associated with a real value $l(a)$ called the *label* of the arc. A DD is compactly represented by $\mathcal{D} = (\mathcal{U}, \mathcal{A}, l(\cdot))$ indicating the set of node layers, arc layers and label mapping, respectively. The size of a DD is often measured by its *width*, denoted by $|\mathcal{D}|$, which records the maximum number of nodes in a node layer of DD.

DD $\mathcal{D}$ represents a set of points of the form $\boldsymbol{x} = (x_1, \ldots, x_n)$ with the following characterization. The label $l(a)$ of each arc $a \in \mathcal{A}_i$, for $i \in N$, represents a value assignment for x_i. Each node in layer $\mathcal{U}_i$ has a maximum outdegree equal to the number of distinct values in the domain of variable x_i. This definition implies that each arc-specified path from s to t, denoted by $\mathrm{P} = (a_1, \ldots, a_n) \in \mathcal{A}_1 \times \ldots \times \mathcal{A}_n$, encodes a value assignment to vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ such that $x_i = l(a_i)$ for all $i \in N$. The collection of points encoded by all paths of $\mathcal{D}$ is referred to as the solution set of $\mathcal{D}$ denoted by $\mathrm{Sol}(\mathcal{D})$.

Consider a set $\mathcal{P} \subseteq \mathbb{R}^n$ where the domain of each variable is a finite subset of $\mathbb{R}$. 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*. Constructing an exact DD is computationally prohibitive as the size of DD grows exponentially with the number of variables. To mitigate this difficulty, a *relaxed* DD $\overline{\mathcal{D}}$ is defined in such a way that $\mathcal{P} \subset \mathrm{Sol}(\overline{\mathcal{D}})$. Relaxed DDs are often constructed by suitably merging nodes at certain layers in such a way that

for any solution of $\mathcal{P}$ there exists an s-t path of $\overline{\mathcal{D}}$, but the converse would not hold necessarily. The complement of a relaxed DD is a *restricted* DD $\underline{\mathcal{D}}$ defined in such a way that $\mathcal{P} \supset \text{Sol}(\underline{\mathcal{D}})$. Restricted DDs are constructed by selecting a subset of arcs and nodes in such a way that each s-t path of $\underline{\mathcal{D}}$ encodes a solution of $\mathcal{P}$.

Constructing relaxed DDs is central to the viability of DD-based solution methods for practical applications. Relaxed DDs are used in two major ways to solve *suitably-structured* optimization problems. The most common way is to use relaxed DDs inside a specialized branch-and-bound method, where the objective function values are incorporated as weights of arcs. In this approach, the weight of the longest/shortest s-t path provides a dual bound for the optimization problem. These relaxed DDs are then successively refined through branch-and-bound to improve the dual bounds. While this approach can be used as a stand-alone solution method, it can only be applied to problems with special structures that allow for modeling the entire solution set. Examples are maximum independent sets, maximum cut and maximum 2-SAT problems [5]. The second way of using relaxed DDs is to generate cutting planes that are valid for the original optimization problem. When used in conjunction with classical branch-and-cut methods, this approach can be applied to a broader class of optimization problems as illustrated in [9]. We next elaborate on the technique developed in [9] as it provides the basis for our study in this paper.

2.2. Convex hull description

The solution set of DDs can be represented by a network formulation as given next; see [9] for a detailed derivation. For the sake of notational convenience, we use label value l_a for an arc $a \in \mathcal{A}$ as a shorthand for $l(a)$.

Proposition 1. *Consider a DD* $\mathcal{D} = (\mathcal{U}, \mathcal{A}, l(\cdot))$ *with solution set* $\text{Sol}(\mathcal{D})$.

Define $\mathcal{F}(\mathcal{D}) = \left\{(\boldsymbol{x};\boldsymbol{y}) \in \mathbb{R}^n \times \mathbb{R}_+^{|\mathcal{A}|} \middle| (1a), (1b)\right\}$ *where*

$$\sum_{a\in\delta^+(i)} y_a - \sum_{a\in\delta^-(i)} y_a = f_i, \qquad \forall i \in \mathcal{U} \tag{1a}$$

$$\sum_{a\in\mathcal{A}_k} l_a\, y_a = x_k, \qquad \forall k \in N, \tag{1b}$$

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 1, (1a) represents the flow-balance constraints for the network induced by $\mathcal{D}$ where $\boldsymbol{y}$ denotes the flow vector and $\boldsymbol{f}$ indicates the supply/demand vector. Constraint (1b) represents the relation between variables $\boldsymbol{x}$ and $\boldsymbol{y}$. 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 $\mathrm{Sol}(\mathcal{D})$. To obtain the description of the convex hull in the space of variables $\boldsymbol{x}$, a projected formulation is given in [9] as presented next. For this formulation, $\boldsymbol{\theta} \in \mathbb{R}^{|\boldsymbol{\mathcal{U}}|}$ and $\boldsymbol{\gamma} \in \mathbb{R}^n$ represent the dual variables corresponding to constraints (1a) and (1b), respectively.

Proposition 2. *Consider any point* $(\bar{\boldsymbol{\theta}};\bar{\boldsymbol{\gamma}})$ *of the cone described by*

$$\theta_{t(a)} - \theta_{h(a)} + l_a\gamma_k \le 0, \qquad \forall k \in N, a \in \mathcal{A}_k \tag{2a}$$

$$\theta_s = 0, \tag{2b}$$

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

$$\sum_{k\in N} x_k\bar{\gamma}_k \le \bar{\theta}_t, \tag{3}$$

is valid for $\mathrm{conv}(\mathrm{Sol}(\mathcal{D}))$. *Further, any facet-defining inequality of* $\mathrm{conv}(\mathrm{Sol}(\mathcal{D}))$ *is of the form* (3) *for some extreme ray* $(\bar{\boldsymbol{\theta}};\bar{\boldsymbol{\gamma}})$ *of* (2a)–(2b). □

The projection result of Proposition 2 leads to a natural cut-generating linear program (CGLP) that can be used to separate a given point from the convex hull of the solution set of a DD.

Proposition 3. *Consider a point $\bar{\boldsymbol{x}} \in \mathbb{R}^n$, and define*

$$\omega^* = \max \left\{ \sum_{k \in N} \bar{x}_k \gamma_k - \theta_t \;\middle|\; \begin{array}{l} \text{(2a), (2b)} \\ C(\boldsymbol{\theta}, \boldsymbol{\gamma}) \leq 0 \end{array} \right\}, \tag{4}$$

where the last constraint is a normalization constraint to enforce the problem to be bounded. Then, $\bar{\boldsymbol{x}} \in \text{conv}(\text{Sol}(\mathcal{D}))$ if and only if $\omega^* = 0$. Otherwise, $\bar{\boldsymbol{x}}$ can be separated from $\text{conv}(\text{Sol}(\mathcal{D}))$ via $\sum_{k \in N} x_k \gamma_k^* \leq \theta_t^*$ where $(\boldsymbol{\theta}^*; \boldsymbol{\gamma}^*)$ is an optimal solution of (4). □

Solving a CGLP for separation is computationally expensive when used inside branch-and-bound algorithms due to the large size of DDs constructed in practical applications. In [9], the authors present an iterative subgradient-based separation technique that converges to the optimal solution of the CGLP, but with a faster cut-generation rate to be used in practice.

We conclude this section by highlighting the main difference between the approach developed in the present paper and that of [9]. The results established in [9] including the construction of relaxed DDs, cut-generation algorithms and the outer-approximation framework apply to discrete programs. In the present paper, on the other hand, we develop a new concept to construct relaxed DDs for continuous programs. These DDs are then used through an adaptation of the cut-generation and outer-approximation framework of [9] to form a linear model that encloses the feasible region of the continuous program.

3. Continuous DDs

In this section, we present a framework to obtain DD-based relaxations for models with continuous variables. We first establish the foundation of a new class of DDs, and then present an algorithmic framework to make use of this class to form desired relaxations.

3.1. Arc-reduced DDs

Consider a DD $\mathcal{D} = (\mathcal{U}, \mathcal{A}, l(\cdot))$. For any pair (u, v) of nodes of $\mathcal{D}$, define $A(u, v)$ to be the set of all arcs of $\mathcal{D}$ whose tail node is u and head node is v.

Further, define $l^{\max}_{(u,v)}$ (resp. $l^{\min}_{(u,v)}$) to be the maximum (resp. minimum) label of the arcs in $A(u,v) \neq \emptyset$.

For a given $\mathcal{D}$, we refer to a *node sequence* as a set of nodes lying along a path from the source node to the terminal node, i.e., $\boldsymbol{u} = (u_1, u_2, \cdots, u_{n+1})$ where $u_i \in \mathcal{U}_i$ for $i \in N \cup \{n+1\}$. Let U be the collection of all node sequences of $\mathcal{D}$. For $\boldsymbol{u} \in U$, define $S_{\boldsymbol{u}} = \{x \in \mathbb{R}^n \,|\, l^{\min}_{(u_i,u_{i+1})} \leq x_i \leq l^{\max}_{(u_i,u_{i+1})}, \forall i \in N\}$. Define the collection of all such rectangular sets as $\mathcal{S} = \bigcup_{\boldsymbol{u} \in U} S_{\boldsymbol{u}}$.

Proposition 4. *Consider a DD $\mathcal{D} = (\mathcal{U}, \mathcal{A}, l(\cdot))$. Consider any point $(\bar{\boldsymbol{\theta}}; \bar{\boldsymbol{\gamma}})$ of the cone described by*

$$\theta_u - \theta_v + l^{max}_{(u,v)}\gamma_k \leq 0, \quad \forall k \in N, (u,v) \in \mathcal{U}_k \times \mathcal{U}_{k+1} \text{ s.t. } A(u,v) \neq \emptyset. \tag{5a}$$

$$\theta_u - \theta_v + l^{min}_{(u,v)}\gamma_k \leq 0, \quad \forall k \in N, (u,v) \in \mathcal{U}_k \times \mathcal{U}_{k+1} \text{ s.t. } A(u,v) \neq \emptyset. \tag{5b}$$

$$\theta_s = 0. \tag{5c}$$

Then, the inequality

$$\sum_{k \in N} x_k \bar{\gamma}_k \leq \bar{\theta}_t, \tag{6}$$

is valid for $\mathrm{conv}(\mathcal{S})$. *Further, any facet-defining inequality of* $\mathrm{conv}(\mathcal{S})$ *is of the form* (6) *for some extreme ray* $(\bar{\boldsymbol{\theta}}; \bar{\boldsymbol{\gamma}})$ *of* (5a)–(5c).

Proof. Note that $\mathrm{conv}(\mathcal{S})$ is a polytope as $\mathcal{S}$ is a finite union of polytopes defined by the Cartesian products of intervals. Construct a DD $\bar{\mathcal{D}} = (\mathcal{U}, \bar{\mathcal{A}}, l(\cdot))$ similarly to $\mathcal{D}$ with a difference that for each pair (u,v) of nodes of $\mathcal{D}$ such that $A(u,v) \neq \emptyset$, we only keep the arcs with the smallest and largest labels, i.e., $l^{\max}_{(u,v)}$ and $l^{\min}_{(u,v)}$, and remove other arcs. It is easy to verify that inequality (2a) can be rewritten as inequalities (5a) and (5b) for $\bar{\mathcal{D}}$. It follows from Proposition 2 that inequality (6) is valid for $\mathrm{conv}(\mathrm{Sol}(\bar{\mathcal{D}}))$ for any point $(\bar{\boldsymbol{\theta}}; \bar{\boldsymbol{\gamma}})$ of (5a)–(5c), and that $\mathrm{conv}(\mathrm{Sol}(\bar{\mathcal{D}}))$ is described by the set of inequalities (6) corresponding to extreme rays of (5a)–(5c). The proof will be complete if we show that $\mathrm{conv}(\mathrm{Sol}(\bar{\mathcal{D}})) = \mathrm{conv}(\mathcal{S})$. For the direct inclusion, we show that $\mathrm{Sol}(\bar{\mathcal{D}}) \subseteq \mathcal{S}$. Consider a point $\bar{x} \in \mathrm{Sol}(\bar{\mathcal{D}})$ encoded by an s-t path of $\bar{\mathcal{D}}$ passing through the node sequence $\bar{\boldsymbol{u}}$. It follows from the construction of $\bar{\mathcal{D}}$ that $\bar{x} \in S_{\bar{\boldsymbol{u}}} \subseteq \mathcal{S}$. For the reverse inclusion, it suffices

to show that $\text{conv}(\text{Sol}(\bar{\mathcal{D}})) \supseteq S_{\boldsymbol{u}}$ for all $\boldsymbol{u} \in U$. Consider $S_{\bar{\boldsymbol{u}}}$ for some $\bar{\boldsymbol{u}} \in U$. It is easy to verify that $S_{\bar{\boldsymbol{u}}}$ is equal to the convex hull of the collection of points encoded by all s-t paths of $\bar{\mathcal{D}}$ passed through the node sequence $\bar{\boldsymbol{u}}$, i.e., points $(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n)$ with $\hat{x}_i \in \{l^{\min}_{(\bar{u}_i, \bar{u}_{i+1})}, l^{\max}_{(\bar{u}_i, \bar{u}_{i+1})}\}$ for $i \in N$ are extreme points of $S_{\bar{\boldsymbol{u}}}$. Since such a collection of points is a subset of $\text{Sol}(\bar{\mathcal{D}})$, we conclude that $S_{\bar{\boldsymbol{u}}} \subseteq \text{conv}(\text{Sol}(\bar{\mathcal{D}}))$. □

It follows from Proposition 4 that in a given DD, only the arcs with the smallest and largest label values between two nodes matter for the convex hull description. This property, according to Proposition 4, also applies to a (virtual) DD where there are infintely many arcs between two nodes spanning label values within a continuous interval. In words, the arcs with labels associated with the lower and upper bound of the interval are sufficient to describe the convex hull.

We refer to the DD obtained by removing all arcs between two nodes except those with the smallest and largest labels as an *arc-reduced* DD. An arc-reduced DD is not a relaxed DD, but in fact a restricted DD as it contains a subset of the s-t paths of the original DD. The unique property of an arc-reduced DD, however, is that it contains the points on the *boundary* of the solution set of the original DD, and hence preserves same convex hull. We next show how this key property can be used to construct relaxations for models that contain continuous variables.

Corollary 5. *Consider a compact set $\mathcal{P} \subseteq \mathbb{R}^n$, and a DD $\mathcal{D} = (\mathcal{U}, \mathcal{A}, l(\cdot))$. Assume that for any $\boldsymbol{x} \in \mathcal{P}$, there exists a node sequence $\boldsymbol{u}$ of $\mathcal{D}$ such that $\boldsymbol{x} \in S_{\boldsymbol{u}}$. Then,* $\text{conv}(\mathcal{P}) \subseteq \text{conv}(\text{Sol}(\mathcal{D}))$.

Proof. It holds that $\mathcal{P} \subseteq \bigcup_{\boldsymbol{u} \in U} S_{\boldsymbol{u}} = \mathcal{S}$, since for any $\boldsymbol{x} \in \mathcal{P}$ we have $\boldsymbol{x} \in S_{\boldsymbol{u}}$ for some $\boldsymbol{u}$. As a result, $\text{conv}(\mathcal{P}) \subseteq \text{conv}(\mathcal{S}) = \text{conv}(\text{Sol}(\mathcal{D}))$, where the equality follows from Proposition 4. □

Note here that the DD described in Corollary 5 does not provide a relaxation of $\mathcal{P}$ in the traditional sense, as it does not necessarily contain all points of $\mathcal{P}$.

In fact, the convex hull of the solutions encoded by the DD is a relaxation of the convex hull of $\mathcal{P}$.

From a practical point of view, the main question is how to build a DD that satisfies the conditions of Corollary 5 for a given set $\mathcal{P}$. As a general rule, the construction of such a DD consists of two steps: (i) Building a (virtual) relaxed DD in such a way that it contains an s-t path for every point of $\mathcal{P}$, and (ii) Performing arc-reduction procedure on the relaxed DD as described above. While these steps can be applied to problems of general structures, the construction of the desired relaxed DD can be challenging. We next illustrate these steps in an outer approximation framework which facilitates these tasks.

3.2. Outer approximation

Let J be the index set of the constraints of the following problem

$$\max \quad \boldsymbol{c}^\intercal \boldsymbol{x} \tag{7a}$$

$$\text{s.t.} \quad g^j(\boldsymbol{x}) \leq 0, \qquad \forall j \in J \tag{7b}$$

$$\boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}], \tag{7c}$$

where $g^j(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ for $j \in J$ is a general function.

Following the outer approximation technique proposed in [9], we are interested in building DD relaxations for each individual constraint in (7b). In particular, define the set of points satisfying constraint j over variable domains as $\mathcal{G}^j := \{\boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}] \,|\, g^j(\boldsymbol{x}) \leq 0\}$. Let $\mathcal{D}_j$ be a DD such that $\text{conv}(\text{Sol}(\mathcal{D}_j)) \supseteq \text{conv}(\mathcal{G}^j)$; the construction of such DDs will be discussed in Section 3.3. For a countable set $\mathcal{K} \subseteq \mathbb{R}^n$, we define

$$\max \quad \boldsymbol{c}^\intercal \boldsymbol{x} \tag{8a}$$

$$\text{s.t.} \quad h_j(\bar{\boldsymbol{x}}, \boldsymbol{x}) \leq 0, \qquad \forall \bar{\boldsymbol{x}} \in \mathcal{K}, \quad j \in J \tag{8b}$$

$$\boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}], \tag{8c}$$

where the inequality (8b) is obtained via the CGLP of Proposition 3 to separate point $\bar{\boldsymbol{x}} \in \mathcal{K}$ from $\text{conv}(\text{Sol}(\mathcal{D}_j))$. If the point cannot be separated from $\text{conv}(\text{Sol}(\mathcal{D}_j))$, no inequality will be added to the model for constraint j.

It follows from the definition of $\mathcal{D}_j$ that the feasible region of (8), for any $\mathcal{K} \subseteq \mathbb{R}^n$, is a linear outer approximation for that of (7b). As a result, the bound obtained from the former problem gives a dual bound for the latter problem. This outer approximation model can be refined through an iterative algorithm given next.

Algorithm 1 Outer approximation for (7)

Initialize $\mathcal{K} \leftarrow \emptyset$, $i \leftarrow 1$
Solve (8) for $\mathcal{K}$
if (8) is infeasible **then**
    Return that (7) is infeasible
else
    Find an optimal solution $\bar{\boldsymbol{x}}^i$ of (8)
end if
$\mathcal{K} \leftarrow \mathcal{K} \cup \{\bar{\boldsymbol{x}}^i\}$, update constraints (8b)
if no new constraint is added to (8b) **then**
    Return $\boldsymbol{c}^\intercal \bar{\boldsymbol{x}}^i$ as a dual bound for (7)
else
    $i \leftarrow i + 1$, go to 2
end if

Under the assumption that the modules used to solve LPs in Algorithm 1 return an extreme point as an optimal solution, we can obtain the following convergence result.

Proposition 6. *Algorithm 1 converges to* $p^* = \max\{\boldsymbol{c}^\intercal \boldsymbol{x} \mid \boldsymbol{x} \in \bigcap_{j \in J} \mathrm{conv}(\mathrm{Sol}(\mathcal{D}_j)), \boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}]\}$ *or proves that the problem is infeasible in a finite number of iterations.*

Proof. The proof for the convergence value p^* is straightforward, as for any optimal solution of the current iteration $\bar{\boldsymbol{x}}$ that does not belong to $\mathrm{conv}(\mathrm{Sol}(\mathcal{D}_j))$ for some $j \in J$, the CGLP of Proposition 3 generates a cutting plane to separate the point and add it to $\mathcal{K}$. In this case, the algorithm continues until no such point is added either due to the infeasibility of the augmented LP model, or

the fact that the optimal solution belongs to the convex hull of the solution set of all DDs. To show the finite convergence, note that for any point $\bar{\boldsymbol{x}} \in \mathcal{K}$ and index $j \in J$, there are finitely many outcomes in terms of cutting planes generated from the CGLP of Proposition 3 as the corresponding LP is bounded. Since the total number of inequalities that can be added to (8) are finite, their combination leads to a finite set of LPs, each having a finite set of extreme points as optimal solution. As a result, the number of points in $\mathcal{K}$ is finite. □

3.3. Construction of DDs

In this section, we discuss the construction of $\mathcal{D}_j$ corresponding to $\mathcal{G}^j$ as the basis for the outer approximation method described above. We present the results for the case where the underlying function $g^j(\boldsymbol{x})$ is separable, i.e., it can be written as $\sum_{i \in N} g_i^j(x_i) + g_0^j$ where $g_i^j : \mathbb{R} \to \mathbb{R}$ for $i \in N$ and $g_0^j \in \mathbb{R}$. The extension to the non-separable case, where there are factorable multivariate terms, follows similarly through the use of a backtracking technique as demonstrated in [9].

The construction technique is inspired by the dynamic programming formulation of a knapsack constraint. Accordingly, each node u of the DD is assigned a *state value* $s_u \in \mathbb{R}$ which represents an under-estimator for the resource consumption level at that node. For any $i \in N$, select a collection $\{(l_k, u_k)\}_{k \in L_i}$ of lower and upper bounds of sub-intervals that cover the domain of variable x_i, i.e., $\bigcup_{k \in L_i} [l_k, u_k] = [l_i, u_i]$. The construction method is given in Algorithm 2.

In Algorithm 2, while s^* can be chosen to be any valid lower bound of the underlying univariate function on the specified interval, it is preferable to set it equal to the minimum value of the function over the interval. Computing the minimum value is a simple task for functions with special properties such as being convex, concave, monotone, polynomial, etc.

Algorithm 2 combines the classical approach to build relaxed DDs by merging nodes at layers of a DD together with the arc-reduction method described in Section 3.1. As a result, the output is a DD whose convex hull of the solution set provides a linear outer approximation for $\mathcal{G}^j$. We refer to such DD as an

Algorithm 2 Constructing arc-reduced relaxed DDs for separable constraint $\mathcal{G}^j$

Create the source node with state value g_0^j, and the terminal node with state value 0
for all $i = 1, \ldots, n-1$ **do**
  for all $u \in \mathcal{U}_i$ **do**
    for all $k \in L_i$ **do**
      Create a node $v \in \mathcal{U}_{i+1}$ with state value $s_v = s_u + s^*$ for some $s^* \leq \min\{g_i^j(x_i) \,|\, l_k \leq x_i \leq u_k\}$
      Create two arcs directed from node u to node v with label values l_k and u_k
    end for
  end for
end for
for all $u \in \mathcal{U}_n$ **do**
  for all $k \in L_n$ **do**
    Compute $s_v = s_u + s^*$ for some $s^* \leq \min\{g_n^j(x_n) \,|\, l_k \leq x_n \leq u_k\}$
    if $s_v \leq 0$ **then**
      Create two arcs directed from node u to the terminal node with label values l_k and u_k
    end if
  end for
end for

arc-reduced relaxed DD.

Proposition 7. *For any $j \in J$, the output $\mathcal{D}^j$ of Algorithm 2 is an arc-reduced relaxed DD for $\mathcal{G}^j$, i.e.,* $\mathrm{conv}(\mathcal{G}^j) \subseteq \mathrm{conv}(\mathrm{Sol}(\mathcal{D}^j))$.

Proof. Consider a point $\bar{\boldsymbol{x}} \in \mathcal{G}^j$. It follows from definition that $\bar{x}_i \in [l_i, u_i]$ for all $i \in N$. Hence, for each $i \in N$, there exists a sub-interval $[l_{k_i}, u_{k_i}]$ for some $k_i \in L_i$ that contains $\bar{x}_i$. Starting from the source node, let $(v_1, v_2, \ldots, v_n)$

be the sequence of nodes at layers of the DD created through steps 2 – 9 of Algorithm 2 and connected via arcs representing intervals $[l_{k_i}, u_{k_i}]$ for all $i \in N \setminus \{n\}$. It follows from the construction that $s_{v_1} = g_0^j$ and that $s_{v_{i+1}} = s_{v_i} + s_i^* \leq s_{v_i} + g_i^j(\bar{x}_i)$ for all $i \in N \setminus \{n\}$, where the inequality holds because $s_i^* \leq \min\{g_i^j(x_i) \,|\, l_{k_i} \leq x_i \leq u_{k_i}\} \leq g_i^j(\bar{x}_i)$ as $\bar{x}_i \in [l_{k_i}, u_{k_i}]$. This recursive equation implies that $s_{v_n} \leq \sum_{i=1}^{n-1} g_i^j(\bar{x}_i) + g_0^j$. Using a similar argument, it follows from steps 10 – 16 of Algorithm 2 that $s_{v_n} + s_n^* \leq \sum_{i \in N} g_i^j(\bar{x}_i) + g_0^j \leq 0$, where the last inequality follows from the assumption that $\bar{\boldsymbol{x}} \in \mathcal{G}^j$. Therefore, arcs representing interval $[l_{k_n}, u_{k_n}]$ connect v_n to the terminal node $t = v_{n+1}$. We conclude that the DD contains a node sequence $\boldsymbol{v} = (v_1, v_2, \ldots, v_{n+1})$ such that $\bar{x} \in S_{\boldsymbol{v}}$. Corollary 5 completes the proof. □

The next example gives a geometric interpretation of the arc-reduced relaxations.

Example 1. Consider a set $\mathcal{G}^j$ for some $j \in J$. Assume that the univariate function for variable x_i for some $i \in N$ is of the form $g_i^j(x_i) = \frac{1}{15}x_i^3 - 0.8x_i^2 + 2.4x_i + 2$. Assume also that the domain of variable x_i is $[0, 8]$ in $\mathcal{G}^j$. Define L_i as the collection of sub-intervals of unit length covering $[0, 8]$. The graph of the nonconvex function $g_i^j(x_i)$ is shown in Figure 1. According to Algorithm 2, the construction of DD at layer i involves finding an under-estimator for $g_i^j(x_i)$ over each interval in L_i. The resulting state values, when plotted over the underlying intervals, yields a *Riemann*-type under-approximation of the function curve as depicted in Figure 1. The arc-reduction procedure can be interpreted as removing the continuous intervals on the Riemann bars and replacing them with the break points at the ends of each interval as shown in Figure 2. The set of these remaining points are represented by paths of the DD, after being projected on the space of $\boldsymbol{x}$ variables corresponding to the level set induced by $\mathcal{G}^j$.

It is clear from the construction of arc-reduced relaxed DDs that as the length of intervals in L_i decreases, the approximations of the functions included

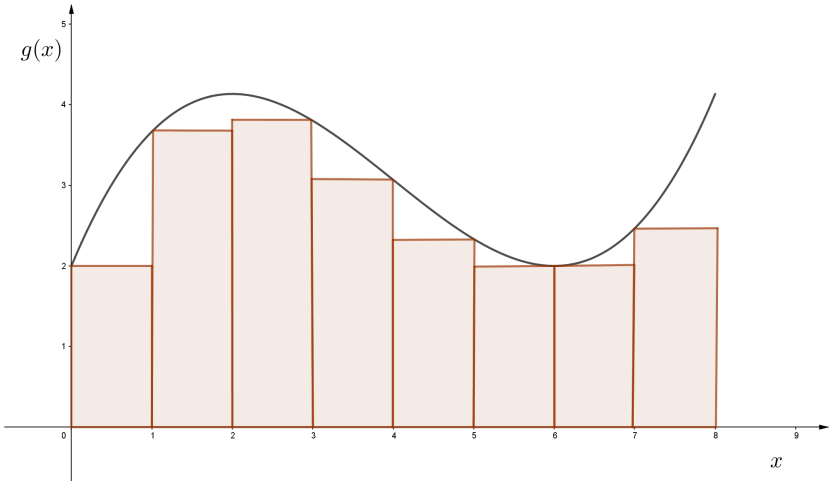

Figure 1: Riemann approximation based on DD

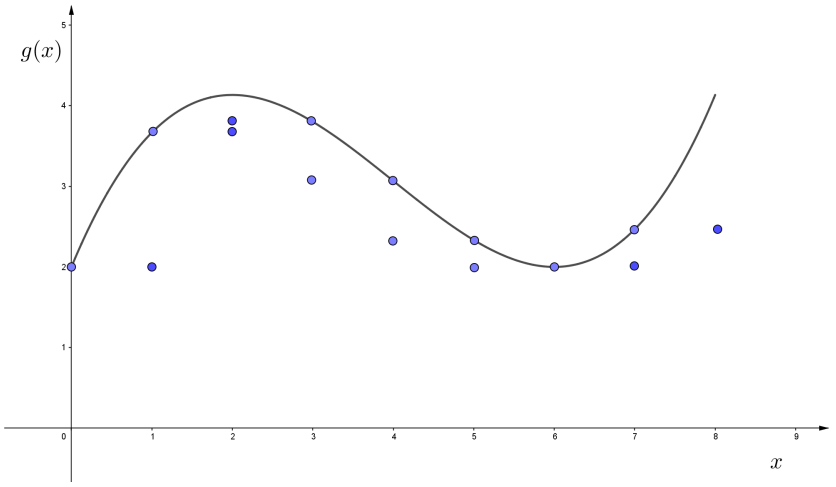

Figure 2: Arc-reduced relaxed DD

in $\mathcal{G}^j$ become closer to the original function curve, leading to a tighter linear relaxation of the set. This observation can be exploited through suitable branch-and-reduce type algorithms to successively refine relaxations and improve the bound quality.

We conclude this section by noting that the arc-reduction technique proposed here for continuous variables can be used in conjunction with the relaxation

techniques developed in [9] for discrete models to build DDs for constraints with both discrete and continuous variables. This expansion allows for applying DDs to new problem classes that could not be modeled by traditional DD techniques due to integrality restrictions.

4. Computational Results

In this section, we conduct a computational study to show the potential of the arc-reduction technique in constructing strong relaxations for continuous nonconvex programs. These results are obtained on a Windows 10.1 (64-bit) operating system, 16 GB RAM, 2.20 GHz Core i7 CPU. Our algorithms are written in Julia v1.1 via JuMP v0.19 and the integer linear optimization models are solved with CPLEX v12.7.1. The results are compared with those obtained from the state-of-the-art solvers via GAMS v24.8.5 modeling language.

The problem class we study is a pricing problem in marketing applications benchmarked in [9]. The formulation is as follows.

$$\min \quad \sum_{i=1}^{n} c_i x_i \tag{9a}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} a_i^j x_i e^{-x_i^{k_i^j}} \geq b_j, \qquad \forall j \in J \tag{9b}$$

$$\boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}], \tag{9c}$$

where x_i represents the price of product i, taking values in $[l_i, u_i]$ for $i \in N$. The objective function minimizes the total weighted pricing of products to assure competitiveness of the firm in the market. Constraints (9b) prevent an excessive loss by imposing a lower bound on the profit margin. The profit function is computed as a weighted sum of each product's profit, which is calculated as the price of the product multiplied with the demand function $e^{-x_i^{k_i}}$, where k_i is the price sensitivity factor of product i.

The key property of problem (9) is its nonconvex structure that makes it challenging for solvers, and hence provides a suitable case study for testing

our method's capabilities. We perform the experiments on benchmark problem instances studied in [9] categorized into three classes based on the problem size: $n = 50$ (small), $n = 200$ (medium), and $n = 500$ (large). The constraint number is set to $|J| = 5$, and variables' bounds are defined as $[\boldsymbol{l}, \boldsymbol{u}] = [0, 10]$. For each category, we consider 5 randomly generated instances with the following characteristics. The objective coefficients c_i belong to a uniform discrete distribution (u.d.d.) between $[0, 20]$, constraint coefficients $a_i^j \sim$ u.d.d $[0, 100]$, the right-hand-side values $b_j \sim$ u.d.d $[10n, 20n]$, and price sensitivity factor $k_i^j \in \{1, 2, 3\}$.

We evaluate the performance of our method by comparing the bounds obtained from the outer approximation algorithm of Section 3.2 with those obtained from the state-of-the-art global solvers ANTIGONE, BARON, COUENNE and SCIP. For the outer approximation algorithm, we use implementation settings similar to those given in [9]. In summary, for each problem instance, we first construct DDs for each constraint using Algorithm 2. The number of sub-intervals over variable domains for each category is considered as follows: $|L_i| = 80$ for small problems, $|L_i| = 50$ for medium problems, and $|L_i| = 30$ for large problems. Next, we run an outer approximation model based on Algorithm 1, starting from box relaxations where all nonlinear constraints are removed. The optimal solution of the resulting LP relaxation is then separated with respect to the DD corresponding to each constraint. For the separation method, we use the subgradient algorithm developed in [9], as the CGLP is too expensive and not viable for practical implementation of this scale. If the point is separated, the corresponding linear cut will be added to the outer approximation and the model is resolved. We set a time limit of 300 seconds, and report the best dual bound obtained at termination. Other parameters are set as follows. The maximum number of cuts that can be added to the outer approximation model at each iteration is set to 2. The maximum number of iterations per each subgradient search to find a violated inequality is set to 20. In all of our experiments, we apply the outer approximation algorithm at the root node, and report the best dual bound at termination. In contrast, the solvers

we employ in our experiments use their default arsenal of branch-and-cut tools including preprocessing, cutting planes, branching, etc.

Table 1 shows the performance of our outer approximation method as compared to global solvers for different problem sizes. Column "#" represents instance number in a category. Column "UB" contains the best primal bound obtained across all solvers within the time limit. The dual bound obtained from our DD-based algorithm at termination is reported in column "DD LB". Columns "OA #" and "cut #" show the number of iterations in the outer approximation method and the total number of DD cuts added to the model, respectively. The total 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 bounds 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 our algorithm with respect to each solver in the order above. For instance, this value is computed as $\frac{\text{DD LB} - \text{ATGN}}{\text{UB} - \text{ATGN}}$ for ANTIGONE.

As observed in the table, the bound obtained by our method achieves a significant gap improvement upon those obtained by the solvers for the small problem sizes. For the medium size problems, while SCIP has the best performance among all solvers, our method demonstrates a considerable gap improvement compared to the other three solvers. Even though the bound quality diminishes for all solvers as we switch to the large problem size category, our method still manages to uniformly outperform all solvers. The decline in the gap improvement rate of our method for larger problem sizes is attributed to the fact that, with the increase in the size of the DDs, the number of sub-intervals over the domain of variables decreases to keep the construction of DDs within a reasonable time. As a result, the corresponding approximations becomes weaker, leading to smaller gap improvements.

It is worth noting that the computational study presented here is intended to showcase the potential of our developments. While the results show promise for our technique of building relaxations through the use of DDs, comprehensive

computational studies over various problem classes are encouraged to identify practical strengths and weaknesses of the approach in handling different problem structures.

Size	#	UB	DD LB	OA #	cut #	Time	ATGN	Δ_1	BARN	Δ_2	COUN	Δ_3	SCIP	Δ_4
50	1	1584.2	1393.3	198	380	44.8	996.3	68%	1302.2	32%	1277.5	38%	1250.1	43%
	2	1813.3	1663.7	300	563	41.2	1037.4	81%	1354.2	67%	1437.6	60%	1368.3	66%
	3	1881.4	1684.2	264	526	29.7	1089.1	75%	1448.1	54%	1479.1	51%	1407.9	58%
	4	2381.5	2218.8	322	638	23.2	1139.7	87%	1509.1	81%	1729.2	75%	1584.7	80%
	5	1785.7	1639.4	375	735	26	808.6	85%	1183.3	76%	1273.7	71%	1216.5	74%
200	1	5861.3	4165.4	264	526	50.6	3186.2	37%	3883.2	14%	3205.3	36%	4404.2	−14%
	2	6237.9	4466.1	289	576	46.6	3361.3	38%	4205.8	13%	3370.3	38%	4658.1	−11%
	3	7998.5	5262.2	270	538	45.1	3517.1	39%	4336.3	25%	3532.5	39%	5389.4	−5%
	4	6957.1	4575.1	321	629	48.6	3348.3	34%	4215.8	13%	3407.4	33%	4864.1	−12%
	5	8259.6	5809.8	428	852	40.1	3937.1	43%	4904.9	27%	3971.7	43%	5785.3	1%
500	1	24391.3	15569.4	31	58	190.9	12444.3	26%	15103.2	5%	12467.7	26%	12302.1	27%
	2	19778.6	12453.3	17	32	234.7	9213.6	31%	11428.8	12%	9219.5	31%	12449.3	0%
	3	16019.2	9375.9	11	22	206.2	8515.5	11%	9232.8	2%	8525.2	11%	8435.7	12%
	4	21272.5	13307.3	15	28	221.6	11028.7	22%	13286.2	0%	11040.6	22%	10918.8	23%
	5	21377.2	13004.2	8	14	233.1	9337.0	30%	11393.5	16%	9370.5	30%	12797.7	2%

Table 1: Size comparison for the pricing problem

Acknowledgement

We thank Willem-Jan van Hoeve and Christian Tjaandramatja for helpful discussions. We also thank the area editor and the anonymous referee for suggestions that helped improve the paper.

References

[1] Andersen, H. R., Hadžić, T., Hooker, J. N., 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] Behle, M., 2007. Binary decision diagrams and integer programming. Ph.D. thesis, Max Planck Institute for Computer Science.

[3] Bergman, D., Cire, A. A., 2018. Discrete nonlinear optimization by state-space decompositions. Management Science 28, 47–66.

[4] Bergman, D., Cire, A. A., van Hoeve, W.-J., Hooker, J., 2016. Decision Diagrams for Optimization. Springer International Publishing.

[5] Bergman, D., Cire, A. A., van Hoeve, W.-J., Hooker, J., 2016. Discrete optimization with decision diagrams. INFORMS Journal on Computing 28, 47–66.

[6] 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.

[7] Ciré, A. A., van Hoeve, W. J., 2013. Multivalued decision diagrams for sequencing problems. Operations Research 61, 1411–1428.

[8] Davarnia, D., Richard, J.-P. P., Tawarmalani, M., 2017. Simultaneous convexification of bilinear functions over polytopes with application to network interdiction. SIAM Journal on Optimization 27, 1801–1833.

[9] Davarnia, D., van Hoeve, W.-J., 2020. Outer approximation for integer nonlinear programs via decision diagrams. Mathematical Programming. URL https://doi.org/10.1007/s10107-020-01475-4

[10] Hadžić, T., Hooker, J. N., 2006. Discrete global optimization with binary decision diagrams. In: Workshop on Global Optimization: Integrating Convexity, Optimization, Logic Programming, and Computational Algebraic Geormetry (GICOLAG).

[11] Kronqvist, J., Bernal, D., Lundell, A., Grossmann, I., 2017. A review and comparison of solvers for convex MINLP. Optimization and Engineering 20, 397–455.

[12] Tawarmalani, M., Sahinidis, N. V., 2002. Convexification and global optimization in continuous and mixed-integer nonlinear programming: Theory, algorithms, software, and applications. Kluwer Academic Publishers.

[13] Tjandraatmadja, C., van Hoeve, W.-J., 2019. Target cuts from relaxed decision diagrams. INFORMS Journal on Computing 31, 285–301.