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

# A graphical framework for global optimization of mixed-integer nonlinear programs

Danial Davarnia · Mohammadreza Kiaghadi

Received: date / Accepted: date

**Abstract** While mixed-integer linear programming and convex programming solvers have advanced significantly over the past several decades, solution technologies for general mixed-integer nonlinear programs (MINLPs) have yet to reach the same level of maturity. Various problem structures across different application domains remain challenging to model and solve using modern global solvers, primarily due to the lack of efficient parsers and convexification routines for their complex algebraic representations. In this paper, we introduce a novel graphical framework for globally solving MINLPs based on decision diagrams (DDs), which enable the modeling of complex problem structures that are intractable for conventional solution techniques. We describe the core components of this framework, including a graphical reformulation of MINLP constraints, convexification techniques derived from the constructed graphs, efficient cutting plane methods to generate linear outer approximations, and a spatial branch-and-bound scheme with convergence guarantees. In addition to providing a global solution method for tackling challenging MINLPs, our framework addresses a longstanding gap in the DD literature by developing a general-purpose DD-based approach for solving general MINLPs. To demonstrate its capabilities, we apply our framework to solve instances from one of the most difficult classes of unsolved test problems in the MINLP Library, which are otherwise inadmissible for state-of-the-art global solvers.

**Keywords** Mixed-Integer Nonlinear Programs · Global Solver · Decision Diagrams · Cutting Planes · Outer Approximation · Spatial Branch-and-Bound

## 1 Introduction

Optimization solvers play a crucial role in advancing mathematical optimization by bridging theoretical breakthroughs with computational power to solve real-world problems. While mixed-integer linear programming and convex programming solvers have made significant strides over the past few decades, solution techniques for general MINLPs still face substantial challenges. Some of these

This work was supported in part by the AFOSR YIP Grant FA9550-23-1-0183, and the NSF CAREER Grant CMMI-2338641.

D. Davarnia
Department of Industrial and Manufacturing Systems Engineering, Iowa State University, Ames, IA
E-mail: davarnia@iastate.edu ✉

M. Kiaghadi
Department of Industrial and Manufacturing Systems Engineering, Iowa State University, Ames, IA
E-mail: kiaghadi@iastate.edu

challenges arise from problem structures that fall outside the framework of current solvers, primarily due to the absence of appropriate parsers for specific algebraic representations found in various application domains. Examples include error functions in statistical models for portfolio optimization and risk management applications, hyperbolic functions in learning models for compressor power in artificial intelligence applications, cross-entropy functions in information theory and econometrics applications, and gamma functions in quantum mechanics applications; see Section 6 for a detailed discussion for such applications. Even when problems fall within the modeling capabilities of modern solvers, several classes with complex structures suffer from weak approximations and poor solution performance. As a result, there remains an ongoing need to develop global solution algorithms that mitigate such limitations of traditional techniques.

In this paper, we introduce a novel graphical framework to globally solve general MINLPs. The basis of this framework is formed by DDs, where the underlying problems are formulated through special-structured graphs. These graphs draw out data structures and variable interactions that often remain latent in the classical algebraic representation of constraints. This intrinsic feature enables DDs to model a broad array of functional forms, including nonconvex, nonsmooth, and even black-box types, that are intractable by standard solution techniques. Numerous computational studies suggest that DD-based algorithms can improve the solution time and quality compared to the outcome of modern solvers. Despite the success of DDs in various application areas, they have never been used to globally solve general MINLPs. As a result, the framework proposed in this paper marks the first solution technology for general MINLPs based on DDs that departs from traditional algebraic approaches to global optimization by capturing the graphical structure of the formulation.

### 1.1 Related Work

MINLPs are considered one of the most challenging classes of optimization problems, as they involve a combination of continuous and discrete variables along with nonlinear relationships in the constraints and/or objective function [8, 40]. As a result, globally solving MINLP formulations can be a daunting task, even for the most advanced optimization solvers. The prevalent framework to solve these problems globally is the *spatial branch-and-bound* [59], which relies on successive bound-reductions and convexification routines [7, 19]. The most common convexification routine is the *factorable decomposition*, where complicated terms are decomposed into simpler components with known convex relaxations [43, 38]. As the leading commercial global solver, BARON [52] employs these strategies to handle a wide range of MINLP structures. At its core, BARON integrates range-reduction methods with branch-and-bound techniques to guide the search towards a globally optimal solution [48]. To accelerate the search process, cutting planes and range-contraction methods have been proposed [50, 63]. Additional boosting techniques for special-structured problems include domain reduction for separable concave programs [58], consistency methods for mixed-binary problems [25], envelope construction for bilinear terms [31, 24, 37], and decomposition strategies for multilinear sets [5, 26, 42, 51]. The concept of convex extension has also been introduced to achieve tighter relaxations for lower semi-continuous functions [61] and fractional programs [60]. As a common class of MINLPs, nonconvex quadratically-constrained quadratic programs have attracted considerable research efforts, ranging from polyhedral approximations [55] to semi-definite relaxations [4]. Open-source solvers such as SCIP [2] and COUENNE [6] utilize constraint programming and polyhedral approximations, respectively, to globally solve MINLPs. Despite advancements in these global solvers, there remains a lack of capable solvers for handling MINLPs with irregular functional forms, such as hyperbolic trigonometric terms, as noted in [13].

While the aforementioned approaches target MINLPs with general structures, a significant body of literature focuses on a special class of MINLPs where the underlying functions are convex. For these problems, the convexity property allows the design of an *outer approximation* scheme that can

converge to an optimal solution without relying on spatial branch-and-bound techniques [27, 33, 45]. Outer approximation methods establish a refinement framework that recursively constructs and solves mixed-integer linear approximations of the problem; see [15] for a survey on such methods. Various solvers have been developed based on this framework, with BONMIN [14] being one of the most well-known. A recent review of the computational performance of these solvers can be found in [39]. However, when applied to nonconvex MINLPs, these solvers only provide local solutions with no guarantee of global optimality. Other local solvers commonly used to solve continuous relaxations of MINLPs include IPOPT [66], SNOTP [28], and KNITRO [17]. Most of these approaches use interior-point methods to find a locally-optimal solution; see [64, 66]. In contrast to these outer approximation and local solvers, our proposed solution method leverages outer approximation to find global optimal solutions for nonconvex MINLPs.

The core structure of our solution methodology in this paper is formed by DDs. DDs were introduced in [32] as an alternative modeling tool for certain classes of combinatorial problems. Later, [3] proposed the concept of relaxed DDs to mitigate the exponential growth in DD size when modeling large-scale discrete problems. Since then, significant efforts have been dedicated to enhancing DD performance in discrete optimization problems; see [34] for a tutorial on DDs, [11] for an introduction to DD modeling, and [18] for a recent survey. Thanks to their promising performance, DDs have been applied across a wide range of application areas, including healthcare [9], supply chain management [12], and transportation [54]. Other avenues of research in the DD community include cutting plane theory [23, 65], multi-objective and Lagrangian optimization [10], post-optimality analysis [57], sub-optimality and dominance detection [20], integrated search tree [30], two-stage stochastic programs [41, 53], and sequence alignment [35]. The novel perspective that DDs offer for modeling optimization problems has propelled DD-based solution methods into the spotlight in recent years. In this paper, we extend DDs further by leveraging their unique structural properties to develop a general-purpose global framework for solving complex MINLPs.

### 1.2 Contributions

**Contributions to the MINLP literature.** As discussed in Section 1.1, the global MINLP solution methods are based mainly on the algebraic representation of the constraints. Nonconvex nonlinear terms are typically decomposed into simpler forms and convexified individually, making these techniques highly dependent on the specific algebraic form and properties of the functions involved. For example, hyperbolic and trigonometric terms, which are common in applications ranging from artificial intelligence to energy systems, are often inadmissible in leading global solvers like BARON and SCIP due to the absence of suitable convexification routines for such structures.

In contrast, our DD-based framework offers significantly more flexibility with respect to the functional forms of the problem. The graphical nature of DDs enables direct evaluation and efficient relaxation of the underlying terms without decomposition during DD construction. As a result, our framework can be effectively applied to a wide range of MINLPs containing highly nonlinear, nonconvex, nonsmooth, and even black-box functions—many of which are intractable or poorly handled by modern global solvers. This paper presents a novel global solution method for general MINLPs that is rooted in the graphical structure of the problem, rather than its algebraic representation.

**Contributions to the DD literature.** Despite the successful application of DDs to various optimization problems over the past two decades, two significant limitations in their applicability have persisted: (i) DDs have primarily been applied to problems with special structures, and (ii) DDs were originally limited to modeling discrete programs. These limitations, recognized in [11], have posed a significant barrier to the universal adoption of DDs, highlighting the need for a general-purpose DD technology to solve MINLPs. In [23], the authors addressed challenge (i) by introducing

an outer approximation framework that tightens relaxations of integer nonlinear programs using DDs. To tackle challenge (ii), [22] proposed a novel DD-based methodology that enhances dual bounds for continuous programs. Subsequent works, including [53, 54], extended the scope of DDs through a DD-based Benders decomposition approach, which enabled their application to mixed-integer linear programs in energy systems and transportation.

While these efforts have expanded the applicability of DDs to new problem classes, they were not designed to obtain global optimal solutions for general MINLPs, lacking key elements such as algorithmic architectures and convergence mechanisms required for handling such broader problem structures. In this paper, we address this gap by introducing a novel branch-and-cut framework that leverages the structure of DDs at every stage of the solution process, from constructing efficient relaxations, to generating cutting planes and outer approximations, to performing branch-and-bound with convergence guarantees. As a result, this work establishes the first general-purpose DD-based solution method for globally solving general MINLPs.

The remainder of this paper is organized as follows. In Section 1, we introduce the structure of the MINLP under study and outline the main steps of our proposed solution framework. Section 3 provides background on DDs and details the algorithms used to construct DDs for different problem structures, representing relaxations of the MINLP. Additionally, we present strategies for calculating bounds for DDs and analyze the runtime complexity of the algorithms. In Section 4, we demonstrate how the constructed DDs can be used to generate linear outer approximations for the MINLP via various cut-generation methods employed within a separation oracle. Section 5 introduces a spatial branch-and-bound scheme designed to refine these outer approximations, with convergence guarantees to a global optimal solution for the MINLP. To evaluate the effectiveness of the framework, we present computational experiments on benchmark MINLP instances in Section 7. Concluding remarks are provided in Section 7.

**Notation.** We denote the vectors by bold letters. For any $k \in \mathbb{N}$, we define $[k] = \{1, 2, \dots, k\}$. Given a vector $\boldsymbol{x} \in \mathbb{R}^n$, we refer to a sub-vector of $vcx$ that includes variables with indices in $J \subseteq [n]$ as $\boldsymbol{x}_J$. We use calligraphic font to describe sets. Given a set $\mathcal{P} \subseteq \{(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^{n+m}\}$, we refer to the convex hull of $\mathcal{P}$ by $\text{conv}(\mathcal{P})$. We denote by $\text{proj}_{\boldsymbol{x}}(\mathcal{P})$ the projection of $\mathcal{P}$ onto the space of $\boldsymbol{x}$ variables. For a nested sequence $\{\mathcal{P}^j\}$ of sets $\mathcal{P}^j \subseteq \mathbb{R}^n$ for $j \in \mathbb{N}$, we denote by $\{\mathcal{P}^j\} \searrow \mathcal{P}$ the fact that this sequence converges (in the Hausdorff sense) to a set $\mathcal{P} \subseteq \mathbb{R}^n$. Given a closed interval $\mathcal{D} \subseteq \mathbb{R}$, we refer to its lower and upper bound as $\mathcal{D}\downarrow$ and $\mathcal{D}\uparrow$, respectively. To distinguish notation, we will represent the elements of a DD using 'typewriter' font. In particular, we define a DD as $\mathtt{D} = (\mathtt{U}, \mathtt{A}, \mathtt{l}(.))$. In this definition, the nodes of the DD are represented by $\mathtt{u} \in \mathtt{U}$ with state value $\mathtt{s}(\mathtt{u})$, and the arcs of DD are denoted by $\mathtt{a} \in \mathtt{A}$ with label $\mathtt{l}(\mathtt{a})$. We refer to the tail and the head nodes of an arc $\mathtt{a} \in \mathtt{A}$ as $\mathtt{t}(\mathtt{a})$ and $\mathtt{h}(\mathtt{a})$, respectively.

## 2 Problem Definition

Consider the MINLP

$$\zeta^* = \max \quad \boldsymbol{c}^\intercal \boldsymbol{x} \tag{2.1a}$$

$$\text{s.t.} \quad g^k(\boldsymbol{x}) \leq b_k, \qquad \forall k \in K \tag{2.1b}$$

$$x_i \in \mathcal{D}_i, \qquad \forall i \in I \cup C \tag{2.1c}$$

where $g^k(\boldsymbol{x}) : \prod_{i \in I \cup C} \mathcal{D}_i \to \mathbb{R}$ for $k \in K$ is a general mixed-integer nonlinear function that is well-defined and bounded over the domain of variables described in (2.1c). In the above model, $I$ and $C$ are the index sets for integer and continuous variables, respectively. Further, $\mathcal{D}_i := [\mathcal{D}_i\downarrow, \mathcal{D}_i\uparrow] \cap \mathbb{Z}$ represents the bounded domain for integer variable $x_i$ with $i \in I$, and $\mathcal{D}_i := [\mathcal{D}_i\downarrow, \mathcal{D}_i\uparrow]$ represents

the bounded domain interval for continuous variable $x_i$ for $i \in C$. This definition implies that for $i \in I$, $\mathcal{D}_i\downarrow, \mathcal{D}_i\uparrow \in \mathbb{Z}$. Define the feasible region of constraint $k \in K$ over the variables' domain as

$$\mathcal{G}^k = \left\{ \boldsymbol{x} \in \prod_{i \in I \cup C} \mathcal{D}_i \;\middle|\; g^k(\boldsymbol{x}) \leq b_k \right\}. \tag{2.2}$$

We outline the main steps of our solution method for globally solving (2.1a)–(2.1c) in Algorithm 1. Following this, we offer a high-level overview of the most critical components of the algorithm. Commonly used elements in global solvers, along with algorithmic settings, are not detailed here as they are thoroughly covered in the relevant literature; see Section 1 for examples.

This method utilizes a branch-and-bound (B&B) tree, where each node represents a specific restriction of the feasible region of (2.1a)–(2.1c) induced by partitioning the variable domains. The first node (root) of the B&B tree is created in line 1, where the original variable domains $\mathcal{D}_i$ for $i \in [n]$ are considered. In line 2, the function Stop_Flag checks whether a stopping criterion has been met, signaling the termination of the algorithm. These criteria might include factors such as a time limit, iteration count, remaining optimality gap, number of open nodes in the B&B tree, and more. If the stopping criteria are not met, the algorithm continues in line 3 by selecting the next open node in the B&B tree for processing. This node can be chosen using any well-known strategy, such as depth-first, breadth-first, or best-bound. Each node in the B&B tree is associated with a linear programming (LP) relaxation, $LP$, of the problem, defined by the variable domains corresponding to that node, along with any linear inequalities inherited from its parent node, if applicable. This LP model is solved in line 4 to obtain an optimal solution $\boldsymbol{x}^*$. If the problem is infeasible or unbounded, the primal bound $\underline{\zeta}$ or the dual bound $\overline{\zeta}$ will be updated accordingly in line 9. Subsequently, for each constraint $k \in K$, the loop in lines 5–8 is executed. First, it is checked whether the current solution $\boldsymbol{x}^*$ satisfies the constraint (2.1b) for the current $k$. If it does not, the oracle `Construct_DD` is called to build a DD that represents the solutions (or a relaxation of the solutions) to $\mathcal{G}^k$. Next, a linear outer approximation for the solutions of the constructed DD associated with $\mathcal{G}^k$ is generated by calling the oracle `Outer_Approx` to separate point $\boldsymbol{x}^*$ from $\text{conv}(\mathcal{G}^k)$. The constraints obtained from this outer approximation are then added to $LP$. Once these steps are completed for all constraints $k \in K$, the augmented LP model $LP$ is resolved. Depending on the optimal value obtained from this model, the bounds $\underline{\zeta}$ and $\overline{\zeta}$ are updated accordingly. In line 10, the function Prune_Node checks whether the current node can be pruned. The pruning rules applied in this function may include standard rules such as pruning by feasibility, pruning by infeasibility, and pruning by bound, as well as more advanced rules like pruning due to inconsistency [44]. If the node is not pruned, the oracle `Branch` is called to perform the branching operation. This operation includes identifying a variable to branch on and determining the branching value, which may utilize any well-known techniques such as most fractional rule, strong branching, and pseudocost branching; see [16, 44]. Following the branching operation, two new child nodes are created and added to the B&B tree. The bounds of the selected variable are updated for each node based on the branching value. Once this step is completed, the recursive steps of the algorithm are repeated until the stopping criteria in line 2 are met. At this point, the algorithm terminates and returns the calculated primal and dual bounds, which can be used to calculate the remaining optimality gap achieved by the algorithm.

As noted in the description of Algorithm 1, three key oracles, namely `Construct_DD`, `Outer_Approx`, and `Branch`, constitute the backbone of our solution framework. The following three sections are dedicated to explaining these oracles in detail.

## 3 DD Construction

In this section, we discuss the oracle `Construct_DD` in Algorithm 1 by outlining the steps involved in constructing DDs that represent relaxations of the set $\mathcal{G}^k$ for $k \in K$. To simplify notation, we

**Algorithm 1:** A high-level structure of the graphical global optimization framework

**Data:** MINLP of the form (2.1a)–(2.1c)
**Result:** A lower bound $\underline{\zeta}$ and upper bound $\overline{\zeta}$ for $\zeta^*$

```
create the root node of the B&B tree that includes the original domain of variables
while Stop_Flag = False do
    select an open node in the B&B tree
    solve an initial LP relaxation LP of the problem at this node to obtain an optimal solution x*
    forall k ∈ K do
        if g(x*) > b_k then
            call Construct_DD to construct a DD respreseting G^k
            call Outer_Approx to create a linear outer approximation of conv(G^k) based on its associated DD and add it to LP
    solve LP to update ζ and ζ̄, if possible
    if Prune_Node = True then
        prune the current node in the B&B tree
    else
        call Branch to perform branching and create children nodes to be added to the B&B tree
```

will omit the index $k$ whenever the results apply to any constraint, regardless of its specific index. We begin with a brief background on using DDs for optimization in Section 3.1. In Section 3.2, we present methods for constructing DDs when the functions in $\mathcal{G}^k$ are separable. The techniques for constructing DDs for general non-separable functions are discussed in Section 3.3. Section 3.4 includes algorithms for merging nodes at layers of a DD to obtain relaxed DDs of a desired size. In Section 3.5, we introduce a strategy for calculating state values for the nodes at the DD layers to ensure that the resulting DDs provide a valid relaxation for the underlying set. Taking all these components into account, we present the time complexity results for the proposed DD construction algorithms in Section 3.6. In the remainder of this paper, we assume, without loss of generality, a variable ordering in $I \cup C$ corresponding to the layers of the DD, denoted by $[n] = \{1, \dots, n\}$.

### 3.1 Background on DDs

In this section, we present basic definitions and results relevant to our DD analysis. A DD $\mathtt{D}$ is a directed acyclic graph denoted by the triple $(\mathtt{U}, \mathtt{A}, \mathtt{l}(.))$ where $\mathtt{U}$ is a node set, $\mathtt{A}$ is an arc set, and $\mathtt{l} : \mathtt{A} \to \mathbb{R}$ is an arc label mapping for the graph components. This DD is composed of $n \in \mathbb{N}$ arc layers $\mathtt{A}_1, \mathtt{A}_2, \dots, \mathtt{A}_n$, and $n+1$ node layers $\mathtt{U}_1, \mathtt{U}_2, \dots, \mathtt{U}_{n+1}$. The node layers $\mathtt{U}_1$ and $\mathtt{U}_{n+1}$ contain the root $\mathtt{r}$ and the terminal $\mathtt{t}$, respectively. In any arc layer $j \in [n]$, an arc $\mathtt{a} \in A_j$ is directed from the tail node $\mathtt{t}(\mathtt{a}) \in \mathtt{U}_j$ to the head node $\mathtt{h}(\mathtt{a}) \in \mathtt{U}_{j+1}$. The *width* of $\mathtt{D}$ is defined as the maximum number of nodes at any node layer $\mathtt{U}_j$. DDs have been traditionally used to model a bounded integer set $\mathcal{P} \subseteq \mathbb{Z}^n$ such that each $\mathtt{r}$-$\mathtt{t}$ arc-sequence (path) of the form $(\mathtt{a}_1, \dots, \mathtt{a}_n) \in \mathtt{A}_1 \times \dots \times \mathtt{A}_n$ encodes a point $\boldsymbol{x} \in \mathcal{P}$ where $\mathtt{l}(\mathtt{a}_j) = x_j$ for $j \in [n]$, that is $\boldsymbol{x}$ is an $n$-dimensional point in $\mathcal{P}$ whose $j$-th coordinate is equal to the label value $\mathtt{l}(\mathtt{a}_j)$ of the arc $\mathtt{a}_j$. For such a DD, we have $\mathcal{P} = \text{Sol}(\mathtt{D})$, where $\text{Sol}(\mathtt{D})$ represents the set of all $\mathtt{r}$-$\mathtt{t}$ paths.

As outlined above, DDs have traditionally been employed to model and solve discrete optimization problems. For instance, they have been extensively used to address combinatorial problems with special structures, such as stable set, set covering, and matching; see [11]. Recently, through

a series of works [22, 53, 54], the application of DD-based optimization has been extended to mixed-integer programs. This extension has enabled applications in new domains, ranging from energy systems to transportation, involving a combination of discrete and continuous variables. In this paper, we unify and expand upon the methods developed in these works, creating a general-purpose global solution framework for MINLPs, which integrates all essential components, from convexification to spatial branch-and-bound techniques.

The following result presents a technique known as *arc reduction*, which reduces the size of a decision diagram (DD) while preserving the convex hull of its solution set. Originally introduced in [22] for modeling continuous sets using DDs, this technique is extended here to handle mixed-integer sets that involve both discrete and continuous variables. Consider the following definitions for a DD $\mathtt{D} = (\mathtt{U}, \mathtt{A}, \mathtt{l}(.))$. For each pair $(\mathtt{u}, \mathtt{v})$ of connected nodes of $\mathtt{D}$ with $\mathtt{u} \in \mathtt{U}_i$ and $\mathtt{v} \in \mathtt{U}_{i+1}$ for some $i \in [n]$, define $l^{\max}(\mathtt{u}, \mathtt{v})$ to be the maximum label of all arcs connecting $\mathtt{u}$ and $\mathtt{v}$, i.e., $l^{\max}(\mathtt{u}, \mathtt{v}) = \max\{\mathtt{l}(\mathtt{a}) \,|\, \mathtt{a} \in \mathtt{A}, \mathtt{t}(\mathtt{a}) = \mathtt{u}, \mathtt{h}(\mathtt{a}) = \mathtt{v}\}$. Similarly, define $l^{\min}(\mathtt{u}, \mathtt{v}) = \min\{\mathtt{l}(\mathtt{a}) \,|\, \mathtt{a} \in \mathtt{A}, \mathtt{t}(\mathtt{a}) = \mathtt{u}, \mathtt{h}(\mathtt{a}) = \mathtt{v}\}$ to be the minimum label of all arcs connecting $\mathtt{u}$ and $\mathtt{v}$.

**Proposition 3.1** *Consider a DD* $\mathtt{D} = (\mathtt{U}, \mathtt{A}, \mathtt{l}(.))$. *Let* $\bar{\mathtt{D}} = (\bar{\mathtt{U}}, \bar{\mathtt{A}}, \bar{\mathtt{l}}(.))$ *be a DD obtained from* $\mathtt{D}$ *by removing every arc* $\mathtt{a} \in \mathtt{A}$ *such that* $l^{\min}(\mathtt{t}(\mathtt{a}), \mathtt{h}(\mathtt{a})) < \mathtt{l}(\mathtt{a}) < l^{\max}(\mathtt{t}(\mathtt{a}), \mathtt{h}(\mathtt{a}))$. *Then,* $\text{conv}(\text{Sol}(\bar{\mathtt{D}})) = \text{conv}(\text{Sol}(\mathtt{D}))$.

*Proof* We prove the result by showing $\text{conv}(\text{Sol}(\bar{\mathtt{D}})) \subseteq \text{conv}(\text{Sol}(\mathtt{D}))$ and $\text{conv}(\text{Sol}(\bar{\mathtt{D}})) \supseteq \text{conv}(\text{Sol}(\mathtt{D}))$. The forward inclusion is straightforward as the arcs in $\bar{\mathtt{D}}$ are a subset of the arcs in $\mathtt{D}$ by definition, which implies that the root-terminal paths in $\bar{\mathtt{D}}$ are a subset of the root-terminal paths in $\mathtt{D}$. Therefore, $\text{Sol}(\bar{\mathtt{D}}) \subseteq \text{Sol}(\mathtt{D})$, which yields $\text{conv}(\text{Sol}(\bar{\mathtt{D}})) \subseteq \text{conv}(\text{Sol}(\mathtt{D}))$.

For the reverse inclusion, consider a point $\boldsymbol{x} \in \text{conv}(\text{Sol}(\mathtt{D}))$. It follows that there exists a collection of $\mathtt{r}$-$\mathtt{t}$ paths of the form $\mathtt{P}^j = (\mathtt{a}_1^j, \ldots, \mathtt{a}_n^j)$ of $\mathtt{D}$ for $j \in [p]$ for some $p \in \mathbb{N}$, each encoding a point $\boldsymbol{x}^j = (\mathtt{l}(\mathtt{a}_1^j), \ldots, \mathtt{l}(\mathtt{a}_n^j))$, such that $\boldsymbol{x} = \sum_{j=1}^p \lambda_j \boldsymbol{x}^j$ with $\sum_{j=1}^p \lambda_j = 1$ and $\lambda_j \geq 0$ for all $j \in [p]$. Next, construct the sets $S_j$ for each $j \in [p]$ that consists of points $\hat{\boldsymbol{x}}^{j,k} \in \mathbb{R}^n$ for $k \in [q^j]$ for some $q^j \in \mathbb{N}$ such that $\hat{x}_i^{j,k} \in \left\{l^{\min}(\mathtt{t}(\mathtt{a}_i^j), \mathtt{h}(\mathtt{a}_i^j)), l^{\max}(\mathtt{t}(\mathtt{a}_i^j), \mathtt{h}(\mathtt{a}_i^j))\right\}$ for each $i \in [n]$. There are a maximum of $2^n$ such points. Each such point $\hat{\boldsymbol{x}}^{j,k}$ corresponds to an $\bar{\mathtt{r}}$-$\bar{\mathtt{t}}$ path of $\bar{\mathtt{D}}$ because all of its arcs are maintained for this DD by construction, implying that $\hat{\boldsymbol{x}}^{j,k} \in \text{Sol}(\bar{\mathtt{D}})$. As a result, we can write $\boldsymbol{x}^j = \sum_{k=1}^{q_j} \mu^{j,k} \hat{\boldsymbol{x}}^{j,k}$ for some $\boldsymbol{\mu}$ such that $\sum_{k=1}^{q_j} \mu^{j,k} = 1$ and $\mu^{j,k} \geq 0$ for all $k \in [q_j]$. Using this relation for all $j \in [p]$, we can write $\boldsymbol{x} = \sum_{j=1}^p \lambda_j \boldsymbol{x}^j = \sum_{j=1}^p \lambda_j (\sum_{k=1}^{q_j} \mu^{j,k} \hat{\boldsymbol{x}}^{j,k})$ where $\hat{\boldsymbol{x}}^{j,k} \in \text{Sol}(\bar{\mathtt{D}})$ for each $j \in [p]$ and $k \in [q_j]$ with $\lambda_j \mu^{j,k} \geq 0$ and $\sum_{j=1}^p \sum_{k=1}^{q_j} \lambda_j \mu^{j,k} = 1$. Therefore, $\boldsymbol{x} \in \text{conv}(\text{Sol}(\bar{\mathtt{D}}))$. □

Proposition 3.1 suggests that when multiple parallel arcs exist between two nodes in a DD, we can retain only the arcs with the minimum and maximum label values, removing all others. This operation preserves the convex hull of the DD's solution set. For the remainder of this paper, we will assume that this technique is applied wherever applicable.

### 3.2 Relaxed DD for Separable Constraints

In this section, we outline the steps for constructing a DD that represents a relaxation of the constraint set $\mathcal{G} = \left\{\boldsymbol{x} \in \prod_{i=1}^n \mathcal{D}_i \,\middle|\, g(\boldsymbol{x}) \leq b\right\}$ where the underlying function $g(\boldsymbol{x})$ is separable, i.e., $g(\boldsymbol{x}) = \sum_{i=1}^n g_i(x_i)$. These results unify the methods developed in [23] for discrete problems and [22] for continuous problems, extending them to represent the mixed-integer case for a general MINLP. The first procedure for constructing such DDs is provided in Algorithm 2, which is shown in Proposition 3.2 to yield a relaxation for the convex hull of the considered set. In this algorithm, we define $L_i$, for each $i \in [n]$, to be the index set for *sub-domain partitions* $\mathcal{D}_i^j := [\mathcal{D}_i^j \downarrow, \mathcal{D}_i^j \uparrow]$ for $j \in L_i$,

which collectively span the entire domain of variable $x_i$, i.e., $\bigcup_{j\in L_i} \mathcal{D}_i^j = \mathcal{D}_i$. If $x_i$ is integer, a sub-domain partition $\mathcal{D}_i^j$ may consist of integral numbers within an interval, i.e., $\mathcal{D}_i^j := [\mathcal{D}_i^j\downarrow, \mathcal{D}_i^j\uparrow] \cap \mathbb{Z}$. We illustrate the steps of the algorithm in Example 3.1.

**Algorithm 2:** Relaxed DD for a separable constraint

**Data:** Set $\mathcal{G} = \left\{\boldsymbol{x} \in \prod_{i=1}^n \mathcal{D}_i \,\middle|\, g(\boldsymbol{x}) \le b\right\}$ where $g(\boldsymbol{x}) = \sum_{i=1}^n g_i(x_i)$ is separable, and the sub-domain partitions $\mathcal{D}_i^j$ for $j \in L_i$ and $i \in [n]$

**Result:** A DD $\mathtt{D}$

1 create the root node $\mathtt{r}$ in the node layer $\mathtt{U}_1$ with state value $\mathtt{s}(\mathtt{r}) = 0$
2 create the terminal node $\mathtt{t}$ in the node layer $\mathtt{U}_{n+1}$
3 **forall** $i \in [n-1]$, $\mathtt{u} \in \mathtt{U}_i$, $j \in L_i$ **do**
4  calculate $\xi = \mathtt{s}(\mathtt{u}) + \eta$ where $\eta \le g_i(x_i)$ for all $x_i \in \mathcal{D}_i^j$
5  create a node $\mathtt{v}$ with state value $\mathtt{s}(\mathtt{v}) = \xi$ (if it does not already exist) in the node layer $\mathtt{U}_{i+1}$
6  add two arcs from $\mathtt{u}$ to $\mathtt{v}$ with label values $\mathcal{D}_i^j\downarrow$ and $\mathcal{D}_i^j\uparrow$
7 **forall** $\mathtt{u} \in \mathtt{U}_n$, $j \in L_n$ **do**
8  calculate $\xi^* = \mathtt{s}(\mathtt{u}) + \eta$ where $\eta \le g_n(x_n)$ for all $x_n \in \mathcal{D}_n^j$
9  **if** $\xi^* \le b$ **then**
10  add two arcs from $\mathtt{u}$ to the terminal node $\mathtt{t}$ with label values $\mathcal{D}_n^j\downarrow$ and $\mathcal{D}_n^j\uparrow$

**Proposition 3.2** *Consider $\mathcal{G} = \left\{\boldsymbol{x} \in \prod_{i=1}^n \mathcal{D}_i \,\middle|\, g(\boldsymbol{x}) \le b\right\}$, where $b \in \mathbb{R}$, and $g(\boldsymbol{x}) = \sum_{i=1}^n g_i(x_i)$ is separable. Let $\mathtt{D}$ be the DD constructed via Algorithm 2 for some sub-domain partitions $\mathcal{D}_i^j$ with $j \in L_i$ for $i \in [n]$. Then,* $\mathrm{conv}(\mathcal{G}) \subseteq \mathrm{conv}(\mathrm{Sol}(\mathtt{D}))$.

*Proof* It suffices to show that $\mathcal{G} \subseteq \mathrm{conv}(\mathrm{Sol}(\mathtt{D}))$ because the convex hull of a set is the smallest convex set that contains it. Pick $\bar{\boldsymbol{x}} \in \mathcal{G}$. It follows from the definition of $\mathcal{G}$ that $\sum_{i=1}^n g_i(\bar{x}_i) \le b$. For each $i \in [n]$, let $j_i^*$ be the index of a sub-domain partition $\mathcal{D}_i^{j_i^*}$ in $L_i$ such that $\bar{x}_i \in \mathcal{D}_i^{j_i^*}$. This index exists because $\bar{x}_i \in \mathcal{D}_i = \bigcup_{j\in L_i} \mathcal{D}_i^j$, where the inclusion follows from the fact that $\bar{\boldsymbol{x}} \in \mathcal{G}$, and the equality follows from the definition of sub-domain partitions.

Next, we show that $\mathtt{D}$ includes a node sequence $\{\mathtt{u}_1, \mathtt{u}_2, \ldots, \mathtt{u}_{n+1}\}$, where $\mathtt{u}_i \in \mathtt{U}_i$ for $i \in [n+1]$, such that each node $\mathtt{u}_i$ is connected to $\mathtt{u}_{i+1}$ via two arcs with labels $\mathcal{D}_i^{j_i^*}\downarrow$ and $\mathcal{D}_i^{j_i^*}\uparrow$ for each $i \in [n]$. We prove the result using induction on the node layer index $k \in [n]$ in the node sequence $\{\mathtt{u}_1, \mathtt{u}_2, \ldots, \mathtt{u}_{n+1}\}$. The induction base $k = 1$ follows from line 1 of Algorithm 2 as the root node $\mathtt{r}$ can be considered as $\mathtt{u}_1$. For the inductive hypothesis, assume that there exists a node sequence $\{\mathtt{u}_1, \mathtt{u}_2, \ldots, \mathtt{u}_k\}$ of $\mathtt{D}$ with $\mathtt{u}_t \in \mathtt{U}_t$ for $t \in [k]$ such that each node $\mathtt{u}_i$ is connected to $\mathtt{u}_{i+1}$ via two arcs with labels $\mathcal{D}_i^{j_i^*}\downarrow$ and $\mathcal{D}_i^{j_i^*}\uparrow$ for each $i \in [k-1]$. For the inductive step, we show that there exists a node sequence $\{\mathtt{u}_1, \mathtt{u}_2, \ldots, \mathtt{u}_{k+1}\}$ of $\mathtt{D}$ with $\mathtt{u}_t \in \mathtt{U}_t$ for $t \in [k+1]$ such that each node $\mathtt{u}_i$ is connected to $\mathtt{u}_{i+1}$ via two arcs with labels $\mathcal{D}_i^{j_i^*}\downarrow$ and $\mathcal{D}_i^{j_i^*}\uparrow$ for each $i \in [k]$. We consider two cases.

For the first case, assume that $k \le n-1$. Then, the for-loop in lines 3–6 of Algorithm 2 imply that node $\mathtt{u}_k$ is connected to another node in the node layer $\mathtt{U}_{k+1}$, which can be considered as $\mathtt{u}_{k+1}$, via two arcs with labels $\mathcal{D}_k^{j_k^*}\downarrow$ and $\mathcal{D}_k^{j_k^*}\uparrow$ as the conditions of the for-loop are satisfied as follows: $k \in [n-1]$ due to the assumption of the first case, $\mathtt{u}_k \in \mathtt{U}_k$ because of the inductive hypothesis, and $j_k^* \in L_k$ by construction.

For the second case, assume that $k = n$. It follows from lines 1–6 of Algorithm 2 that the state value of node $\mathtt{u}_{i+1}$ for $i \in [k-1]$ is calculated as $\mathtt{s}(\mathtt{u}_{i+1}) = \mathtt{s}(\mathtt{u}_i) + \eta_i$ where $\mathtt{s}(\mathtt{u}_1) = 0$ because of line

1 of the algorithm, and where $\eta_i$ is a lower bound for function $g_i(x_i)$ over the sub-domain partition $\mathcal{D}_i^{j_i^*}$ according to lines 4-5 of the algorithm, i.e., $\eta_i \leq g_i(\bar{x}_i)$ as $\bar{x}_i \in \mathcal{D}_i^{j_i^*}$. As a result, we have $\mathtt{s}(\mathtt{u}_k) = \sum_{i=1}^{k-1} \eta_i \leq \sum_{i=1}^{k-1} g_i(\bar{x}_i)$. Now consider lines 7-10 of the for-loop in Algorithm 2 for $\mathtt{u}_k \in \mathtt{U}_k$ and $j_k^* \in L_k$. We compute $\xi^* = \mathtt{s}(\mathtt{u}_k) + \eta_k$, where $\eta_k$ is a lower bound for function $g_i(x_i)$ over the sub-domain partition $\mathcal{D}_k^{j_k^*}$. Using a similar argument to that above, we conclude that $\eta_k \leq g_k(\bar{x}_k)$. Combining this result with that derived for $\mathtt{s}(\mathtt{u}_k)$, we obtain that $\xi^* = \sum_{i=1}^{k} \eta_i \leq \sum_{i=1}^{k} g_i(\bar{x}_i) \leq b$, where the last inequality follows from the fact that $\bar{\boldsymbol{x}} \in \mathcal{G}$. Therefore, lines 9–10 of Algorithm 2 imply that two arcs with label values $\mathcal{D}_k^{j_k^*}\downarrow$ and $\mathcal{D}_k^{j_k^*}\uparrow$ connect node $\mathtt{u}_k$ to the terminal node $\mathtt{t}$ which can be considered as $\mathtt{u}_{k+1}$, completing the desired node sequence.

Now consider the collection of points $\tilde{\boldsymbol{x}}^\kappa$ for $\kappa \in [2^n]$ encoded by all paths composed of the above-mentioned pairs of arcs with labels $\mathcal{D}_i^{j_i^*}\downarrow$ and $\mathcal{D}_i^{j_i^*}\uparrow$ between each two consecutive nodes $\mathtt{u}_i$ and $\mathtt{u}_{i+1}$ in the sequence $\{\mathtt{u}_1, \mathtt{u}_2, \ldots, \mathtt{u}_{n+1}\}$. Therefore, $\tilde{\boldsymbol{x}}^\kappa \in \mathrm{Sol}(\mathtt{D})$ for $\kappa \in [2^n]$. It is clear that these points form the vertices of an $n$-dimensional hyper-rectangle defined by $\prod_{i=1}^{n}[\mathcal{D}_i^{j_i^*}\downarrow, \mathcal{D}_i^{j_i^*}\uparrow]$. It follows from construction of the DD that $\bar{\boldsymbol{x}} \in \prod_{i=1}^{n}[\mathcal{D}_i^{j_i^*}\downarrow, \mathcal{D}_i^{j_i^*}\uparrow]$, i.e., $\bar{\boldsymbol{x}}$ is a point inside the above hyper-rectangle. As a result, $\bar{\boldsymbol{x}}$ can be represented as a convex combination of the vertices $\tilde{\boldsymbol{x}}^\kappa$ for $\kappa \in [2^n]$ of the hyper-rectangle, yielding $\bar{\boldsymbol{x}} \in \mathrm{conv}(\mathrm{Sol}(\mathtt{D}))$.

□

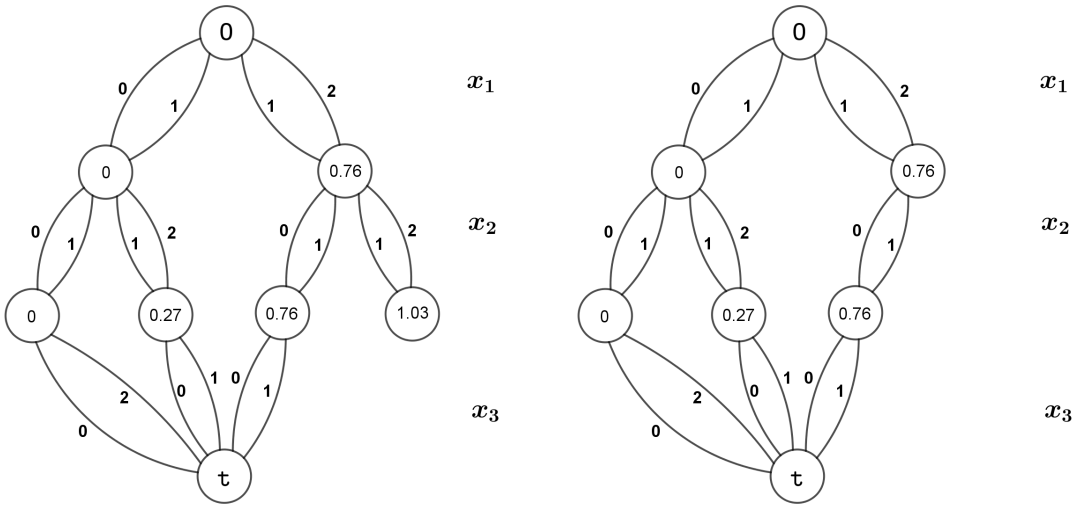
*Example 3.1* Consider a nonconvex MINLP set $\mathcal{G} = \left\{ \boldsymbol{x} \in \prod_{i=1}^{3} \mathcal{D}_i \,\middle|\, \tanh(x_1) + x_2 e^{-x_2} + ||x_3||_0 \leq 1 \right\}$ where $\mathcal{D}_1 = \mathcal{D}_2 = \mathcal{D}_3 = [0, 2]$, and $||x_3||_0 = 0$ if $x_3 = 0$ and $||x_3||_0 = 1$ otherwise. Following the definition of sets studied in this section, we have $g(\boldsymbol{x}) = g_1(x_1) + g_2(x_2) + g_3(x_3)$ where $g_1(x_1) = \tanh(x_1)$, $g_2(x_2) = x_2 e^{-x_2}$, and $g_3(x_3) = ||x_3||_0$.

To obtain a DD relaxation for this set, we apply Algorithm 2 with sub-domain partitions $\mathcal{D}_i^1 = [0, 1]$ and $\mathcal{D}_i^2 = [1, 2]$ for all $i = 1, 2, 3$. Following the steps of the algorithm, we obtain the DD $\mathtt{D}^1$ shown in Figure 3.1, where the number next to each arc represents the arc label, and the number inside each node shows the state value of that node. To illustrate the state value computation of these terms, consider the node with state value 0.76 at node layer 2, which we refer to as $\mathtt{u} \in \mathtt{U}_2$ with $\mathtt{s}(\mathtt{u}) = 0.76$. In line 4 of the algorithm, a lower bound $\eta$ for $g_2(x_2)$ is calculated over the sub-domain partitions $x_2 \in \mathcal{D}_2^1$ and $x_2 \in \mathcal{D}_2^2$. It is easy to verify that $\eta_1 = 0$ is a valid lower bound for $g_2(x_2)$ over the former domain, and $\eta_2 = 0.27 \leq g_2(2)$ is a valid lower bound for $g_2(x_2)$ over the latter domain. Therefore, according to line 5 and 6 of the algorithm, we create a node with state value $\xi = \mathtt{s}(\mathtt{u}) + \eta_1 = 0.76$ in layer 3, which is connected to $\mathtt{u}$ via two arcs with labels 0 and 1. Similarly, we create a node with state value $\xi = \mathtt{s}(\mathtt{u}) + \eta_2 = 1.03$ in layer 3, which is connected to $\mathtt{u}$ via two arcs with labels 1 and 2.

Now consider the node with state value 0 at node layer 3, which we refer to as $\mathtt{v} \in \mathtt{U}_3$ with $\mathtt{s}(\mathtt{v}) = 0$. In line 8 of Algorithm 2, a lower bound $\eta$ for $g_3(x_3)$ is calculated over the sub-domain partitions $x_3 \in \mathcal{D}_3^1$ and $x_3 \in \mathcal{D}_3^2$. For the former domain, we set $\eta = 0$, which is a valid lower bound for $g_3(x_3)$, yielding $\xi^* = \mathtt{s}(\mathtt{v}) + \eta = 0$. This satisfies the condition $\xi^* \leq 1$ in line 9 of the algorithm. Thus, two arcs with labels 0 and 1 are created to connect $\mathtt{v}$ to the terminal node. Similarly, we select $\eta = 1$ to be a valid lower bound for $g_3(x_3)$ over $x_3 \in \mathcal{D}_3^2$, which yields $\xi^* = \mathtt{s}(\mathtt{v}) + \eta = 1$. This satisfies the condition $\xi^* \leq 1$ in line 9 of the algorithm, leading to creation of two arcs with labels 1 and 2 to connect $\mathtt{v}$ to the terminal node. Since the arcs with label value 1 are the middle arcs among those between $\mathtt{v}$ and $\mathtt{t}$, we can use the result of Proposition 3.1 to remove them from the DD as illustrated in Figure 3.1a.

Because there is a node at layer 3 that is not connected to the terminal node, we can reduce the size of the DD by removing that node and its incoming arcs as they do not lead to a feasible path, which yields the so-called *reduced DD* in Figure 3.1b. It is easy to verify that the convex hull of the solutions of $\mathtt{D}^1$ is equal to the convex hull of the union of hyper-rectangles $\{[0, 1] \times [0, 1] \times [0, 2], [0, 1] \times [1, 2] \times [0, 1], [1, 2] \times [0, 1] \times [0, 1]\}$. This yields the polyhedral convex hull $\mathrm{conv}(\mathrm{Sol}(\mathtt{D}^1)) =$

$\big\{(x_1, x_2, x_3) \in [0,2]^3 \,\big|\, x_1 + x_2 + x_3 \leq 4,\ x_1 + x_2 \leq 3,\ x_1 + x_3 \leq 3,\ x_2 + x_3 \leq 3\big\}$. This set provides a relaxation for the convex hull of the original set $\mathcal{G}$ with strict inclusion, i.e., $\text{conv}(\mathcal{G}) \subset \text{conv}(\text{Sol}(\mathtt{D}^1))$. ■




(a) Original DD representation

(b) Reduced DD representation

Fig. 3.1: Relaxed DD for the set in Example 3.1.

According to Algorithm 2, the number of nodes at layer $k+1$ for $k \in [n-1]$ of the DD obtained from this algorithm is bounded by $|\mathtt{U}_k||L_k|$, where $|\mathtt{U}_k|$ is the number of nodes at layer $k$, and $L_k$ is the number of sub-domain partitions for variable $x_k$. As a result, the size of this DD grows exponentially as the number of layers increases. To control this growth rate, we can use two approaches as outlined below.

The first approach involves controlling the size of the DD by adjusting the number of sub-domain partitions for variables at certain layers. For instance, assume that there is an imposed width limit of $\omega$ at layer $k+1$, for some $k \in [n]$ in the DD. To satisfy this width limit at layer $k+1$, we can set the number of sub-domain partitions at layer $k$ to be no greater than $\frac{\omega}{|\mathtt{U}_k|}$, ensuring that the number of nodes at layer $k+1$ does not exceed $\omega$.

The second approach to controlling the size of the DD involves creating a "relaxed DD" by *merging* nodes at layers where the size exceeds the width limit $\omega$. In this process, multiple nodes in a layer are merged into a single node in such a way that all feasible paths of the original DD are preserved. This merging process consists of creating a new node that replaces the merged nodes while inheriting their incoming arcs; see Section 3.4 for detailed steps. For the DDs constructed using Algorithm 2, setting the state value of the new node as the minimum of state values of the merged nodes ensures that all feasible paths of the original DD are preserved, as shown next.

**Proposition 3.3** *Consider* $\mathcal{G} = \big\{\boldsymbol{x} \in \prod_{i=1}^n \mathcal{D}_i \,\big|\, g(\boldsymbol{x}) \leq b\big\}$ *where* $g(\boldsymbol{x}) = \sum_{i=1}^n g_i(x_i)$ *is separable. Let* $\mathtt{D} = (\mathtt{U}, \mathtt{A}, \mathtt{l}(.))$ *be a DD constructed via Algorithm 2 for sub-domain partitions* $\mathcal{D}_i^j$ *with* $j \in L_i$ *for* $i \in [n]$. *Similarly, let* $\bar{\mathtt{D}} = (\bar{\mathtt{U}}, \bar{\mathtt{A}}, \bar{\mathtt{l}}(.))$ *be the DD constructed via Algorithm 2 for the same sub-domain partitions and the same lower bound calculation rules for computing state values of DD nodes with the following additional operation. At a certain layer* $k \in [n-1]$, *a collection of nodes* $\{\mathtt{v}_{p_1}, \mathtt{v}_{p_2}, \ldots, \mathtt{v}_{p_t}\}$ *at node layer*

*$k+1$, for some $t \in \mathbb{N}$, are merged into a node $\tilde{\mathtt{v}}$ with state value $\mathtt{s}(\tilde{\mathtt{v}}) = \min_{j=1,\dots,t}\{\mathtt{s}(\mathtt{v}_{p_j})\}$. Then,* $\mathrm{Sol}(\mathtt{D}) \subseteq \mathrm{Sol}(\bar{\mathtt{D}})$.

*Proof* Consider a solution $\hat{\boldsymbol{x}} \in \mathrm{Sol}(\mathtt{D})$ encoded by the arc sequence $\{\mathtt{a}_1, \mathtt{a}_2, \dots, \mathtt{a}_n\}$ of $\mathtt{D}$, i.e., $\hat{x}_i = \mathtt{l}(\mathtt{a}_i)$ for $i \in [n]$. We show that $\hat{\boldsymbol{x}} \in \mathrm{Sol}(\bar{\mathtt{D}})$, i.e., there exists an arc sequence $\{\bar{\mathtt{a}}_1, \bar{\mathtt{a}}_2, \dots, \bar{\mathtt{a}}_n\}$ in $\bar{\mathtt{D}}$ such that $\bar{\mathtt{l}}(\bar{\mathtt{a}}_i) = \mathtt{l}(\mathtt{a}_i)$ for $i \in [n]$. Let $\mathtt{u}_i$ and $\bar{\mathtt{u}}_i$ be the tail nodes of arcs $\mathtt{a}_i$ and $\bar{\mathtt{a}}_i$ in $\mathtt{D}$ and $\bar{\mathtt{D}}$, respectively. We consider two cases.

For the first case, assume that $\mathtt{u}_{k+1} \notin \{\mathtt{v}_{p_1}, \mathtt{v}_{p_2}, \dots, \mathtt{v}_{p_t}\}$. Then, it is easy to verify that there exists an arc sequence $\{\bar{\mathtt{a}}_1, \bar{\mathtt{a}}_2, \dots, \bar{\mathtt{a}}_n\}$ in $\bar{\mathtt{D}}$ such that $\bar{\mathtt{l}}(\bar{\mathtt{a}}_i) = \mathtt{l}(\mathtt{a}_i)$ for $i \in [n]$, because the steps of Algorithm 2 executed to create this arc sequence is precisely the same as those executed to create arcs in $\{\mathtt{a}_1, \mathtt{a}_2, \dots, \mathtt{a}_n\}$ of $\mathtt{D}$ by definition.

For the second case, assume that $\mathtt{u}_{k+1} \in \{\mathtt{v}_{p_1}, \mathtt{v}_{p_2}, \dots, \mathtt{v}_{p_t}\}$. For each $i \in [n]$, it follows from lines 6 and 10 of Algorithm 2 that arc $\mathtt{a}_i$ of $\mathtt{D}$ is created with the label value equal to the lower bound $\mathcal{D}_i^{j_i^*}\downarrow$ or upper bound $\mathcal{D}_i^{j_i^*}\uparrow$ of the sub-domain partition $\mathcal{D}_i^{j_i^*}$ for some $j_i^* \in L_i$. Further, according to lines 4–5 of Algorithm 2, the state value of the node $\mathtt{u}_{i+1}$, for $i \in [n-1]$, is calculated as $\mathtt{s}(\mathtt{u}_{i+1}) = \mathtt{s}(\mathtt{u}_i) + \eta_i$ with $\eta_i \leq g_i(x_i)$ for all $x_i \in \mathcal{D}_i^{j_i^*}$. In this relation, we set $\mathtt{s}(\mathtt{u}_1) = 0$, which represents the state value of the root node $\mathtt{r}$ as the tail node of arc $\mathtt{a}_1$. Using this relation recursively combined with line 8 of Algorithm 2, we obtain that $\xi^* = \sum_{i=1}^n \eta_i$, where $\eta_n \leq g_n(x_n)$ for all $x_n \in \mathcal{D}_n^{j_n^*}$. Since arc $\mathtt{a}_n$ is in the considered arc sequence, we must have that $\xi^* \leq b$ due to line 9 of Algorithm 2.

Next, we identify the arcs in the sequence $\{\bar{\mathtt{a}}_1, \bar{\mathtt{a}}_2, \dots, \bar{\mathtt{a}}_n\}$ as follows. Starting from the root node $\bar{\mathtt{r}}$ of $\bar{\mathtt{D}}$, for each $i \in [n-1]$, we select $\bar{\mathtt{a}}_i$ to be the arc that represents the lower or upper bound value of $\mathcal{D}_i^{j_i^*}$ that matches the one represented by $\mathtt{a}_i$. Such arcs exist because the sub-domain partitions are the same for $\mathtt{D}$ and $\bar{\mathtt{D}}$ by assumption. We compute the state values for the tail nodes of these arcs. It follows from the assumption that, $\bar{\eta}_i = \eta_i$ for each $i \in [n]$, where $\bar{\eta}_i$ is the lower bound for $g_i(x_i)$ over the sub-domain partition $\mathcal{D}_i^{j_i^*}$ since the same rule for computing a lower bound of functions over each sub-domain partition is used for both $\mathtt{D}$ and $\bar{\mathtt{D}}$. Further, Algorithm 2 implies that $\mathtt{s}(\bar{\mathtt{u}}_{i+1}) = \mathtt{s}(\bar{\mathtt{u}}_i) + \bar{\eta}_i$ for each $i \in [k-1]$. Using the previous argument together with the fact that $\mathtt{s}(\bar{u}_1) = 0$ due to line 1 of the algorithm, we obtain that $\mathtt{s}(\bar{\mathtt{u}}_k) = \mathtt{s}(\mathtt{u}_k)$. Due to the additional merging operation employed at layer $k$ of $\bar{\mathtt{D}}$, we have $\mathtt{s}(\bar{\mathtt{u}}_{k+1}) = \mathtt{s}(\tilde{\mathtt{v}}_{k+1}) \leq \mathtt{s}(\mathtt{u}_{k+1}) = \mathtt{s}(\mathtt{u}_k) + \eta_k = \mathtt{s}(\bar{\mathtt{u}}_k) + \bar{\eta}_k$, where the first equality holds because the tail node $\bar{\mathtt{u}}_{k+1}$ of arc $\bar{\mathtt{a}}_{k+1}$ is the merged node by assumption of this case, the inequality follows from the state value calculation imposed by the merging rule, the second equality holds because of recursive formulation that was previously shown, and the last equality follows from the facts that $\mathtt{s}(\bar{\mathtt{u}}_k) = \mathtt{s}(\mathtt{u}_k)$ and $\bar{\eta}_k = \eta_k$ as demonstrated above. Using this relation along with the recursive formula $\mathtt{s}(\bar{\mathtt{u}}_{i+1}) = \mathtt{s}(\bar{\mathtt{u}}_i) + \bar{\eta}_i$ for $i = \{k, k+1, \dots, n-1\}$, we obtain $\mathtt{s}(\bar{\mathtt{u}}_n) \leq \sum_{i=1}^{n-1} \bar{\eta}_i$. Therefore, it follows from line 8 of Algorithm 2 that $\bar{\xi}^* \leq \sum_{i=1}^n \bar{\eta}_i = \sum_{i=1}^n \eta_i \leq b$, where $\bar{\xi}^*$ is an equivalent term to $\xi^*$ used for $\bar{\mathtt{D}}$, the equality follows from the fact that $\bar{\eta}_i = \eta_i$ for $i \in [n]$ as derived above, and the last inequality holds because $\hat{\boldsymbol{x}} \in \mathrm{Sol}(\mathtt{D})$. We conclude that the condition in line 9 of Algorithm 2 is satisfied, implying that two arcs with labels equal to the bounds in $\mathcal{D}_n^{j_n^*}$ are created to connect $\bar{\mathtt{u}}_n$ to the terminal node $\bar{\mathtt{t}}$ of $\bar{\mathtt{D}}$. Choosing $\bar{\mathtt{a}}_n$ to be the arc with $\bar{\mathtt{l}}(\bar{\mathtt{a}}_n) = \mathtt{l}(\mathtt{a}_n)$, we complete the desired arc sequence.

□

Using the result of Proposition 3.3, we can control the size of a relaxed DD by merging multiple node collections at different node layers, as outlined in Algorithm 3. In this algorithm, the merging operation is performed by the *merging oracle* $\mathtt{Merge}(\omega, \mathtt{V})$, where $\omega$ is the width limit, and $\mathtt{V}$ is the set of nodes at a particular layer of the DD. This oracle employs a *merging policy* that identifies one or more subsets of $\mathtt{V}$ whose nodes should be merged according to the merging operation described in Proposition 3.3 to ensure that the width limit is satisfied. Various merging policies can be implemented within $\mathtt{Merge}(\omega, \mathtt{V})$. In Section 3.4, we present two general and effective policies that can be utilized within this oracle.

**Algorithm 3:** Relaxed DD for a separable constraint with merging operation

**Data:** Set $\mathcal{G} = \left\{ \boldsymbol{x} \in \prod_{i=1}^{n} \mathcal{D}_i \,\middle|\, g(\boldsymbol{x}) \leq b \right\}$ where $g(\boldsymbol{x}) = \sum_{i=1}^{n} g_i(x_i)$ is separable, the sub-domain partitions $\mathcal{D}_i^j$ with $j \in L_i$ for $i \in [n]$, a width limit $\omega$, and a merging oracle $\texttt{Merge}(\omega, \texttt{V})$

**Result:** A DD $\texttt{D}$

1 create the root node $\texttt{r}$ in the node layer $\texttt{U}_1$ with state value $\texttt{s}(\texttt{r}) = 0$
2 create the terminal node $\texttt{t}$ in the node layer $\texttt{U}_{n+1}$
3 **forall** $i \in [n-1]$ **do**
4  **forall** $\texttt{u} \in \texttt{U}_i$, $j \in L_i$ **do**
5  calculate $\xi = \texttt{s}(\texttt{u}) + \eta$ where $\eta \leq g_i(x_i)$ for all $x_i \in \mathcal{D}_i^j$
6  create a node $\texttt{v}$ with state value $\texttt{s}(\texttt{v}) = \xi$ (if it does not already exist) in the node layer $\texttt{U}_{i+1}$
7  add two arcs from $\texttt{u}$ to $\texttt{v}$ with label values $\mathcal{D}_i^j\downarrow$ and $\mathcal{D}_i^j\uparrow$
8  merge nodes at this layer to satisfy the prescribed width limit by invoking the merging oracle $\texttt{Merge}(\omega, \texttt{U}_{i+1})$
9 **forall** $\texttt{u} \in \texttt{U}_n$, $j \in L_n$ **do**
10  calculate $\xi^* = \texttt{s}(\texttt{u}) + \eta$ where $\eta \leq g_n(x_n)$ for all $x_n \in \mathcal{D}_n^j$
11  **if** $\xi^* \leq b$ **then**
12  add two arcs from $\texttt{u}$ to the terminal node $\texttt{t}$ with label values $\mathcal{D}_n^j\downarrow$ and $\mathcal{D}_n^j\uparrow$

**Theorem 3.1** *Consider* $\mathcal{G} = \left\{ \boldsymbol{x} \in \prod_{i=1}^{n} \mathcal{D}_i \,\middle|\, g(\boldsymbol{x}) \leq b \right\}$, *where* $g(\boldsymbol{x}) = \sum_{i=1}^{n} g_i(x_i)$ *is separable. Let* $\texttt{D}$ *be the DD constructed via Algorithm 3 for some sub-domain partitions* $\mathcal{D}_i^j$ *with* $j \in L_i$ *for* $i \in [n]$, *width limit* $\omega$, *and merging oracle* $\texttt{Merge}(\omega, \texttt{V})$. *Then,* $\text{conv}(\mathcal{G}) \subseteq \text{conv}(\text{Sol}(\texttt{D}))$.

*Proof* We prove the result by decomposing the merging operations used in line 8 of the algorithm as follows. Starting from lines 1–2 of Algorithm 3, for each $i \in [n-1]$, consider the node layer $\texttt{U}_{i+1}$ after the for-loop 4-7 is completed. Assume that the merging oracle $\texttt{Merge}(\omega, \texttt{U}_{i+1})$ invoked in line 8 of the algorithm consists of the merging operations $O_1^i, O_2^i, \ldots, O_{p_i}^i$ for some $p_i \in \{0, 1, \ldots\}$, where $O_j^i$ indicates the operation for merging nodes in a subset $V_j^i$ of $\texttt{U}_{i+1}$ into a node $\tilde{\texttt{v}}_j^i$ with the state value $\texttt{s}(\tilde{\texttt{v}}_j^i) = \min_{\texttt{v} \in V_j^i} \{\texttt{s}(\texttt{v})\}$. In this definition, $p_i = 0$ means that no merging operation is used in layer $i$. Note also that in this definition, the sets $V_j^i$ are mutually exclusive for each $i$, i.e., $\bigcap_{j=1}^{p_i} V_j^i = \emptyset$, because a node cannot be merged into multiple distinct nodes.

Define $\texttt{D}_0^0$ to be the DD constructed by Algorithm 2 for the same sub-domain partitions and the same lower bound calculation rules for computing state values that were used for $\texttt{D}$. Define $\texttt{D}_1^1$ to be the DD constructed from Algorithm 2 for the same sub-domain partitions and the same lower bound calculation rules for computing state values that were used for $\texttt{D}_0^0$ with the following additional operation: At layer $i = 1$, the nodes in subset $V_1^1$ of $\texttt{U}_2$ are merged into node $\tilde{\texttt{v}}_1^1$ according to the merging operation $O_1^1$. Then, it follows from Proposition 3.3 that $\text{Sol}(\texttt{D}_0^0) \subseteq \text{Sol}(\texttt{D}_1^1)$. We extend this process as follows: We define $\texttt{D}_j^i$ to be the DD constructed from Algorithm 2 for the same sub-domain partitions and the same lower bound calculation rules for computing state values that were used for $\texttt{D}_{j-1}^i$ if $j > 1$, or $\texttt{D}_{p_i}^{i-1}$ if $j = 1$, with the following additional operation: At layer $i$, the nodes in subset $V_j^i$ of $\texttt{U}_{i+1}$ are merged into node $\tilde{\texttt{v}}_j^i$ according to the merging operation $O_j^i$. In other words, each new DD has one more merging operation applied at some layer of it compared to the DD constructed in the previous iteration. Using arguments similar to those in the proof of Proposition 3.3, we can show that $\text{Sol}(\texttt{D}_{j-1}^i) \subseteq \text{Sol}(\texttt{D}_j^i)$ if $j > 1$, and $\text{Sol}(\texttt{D}_{p_i}^{i-1}) \subseteq \text{Sol}(\texttt{D}_j^i)$ if $j = 1$. Considering this relation for the sequence of the constructed DDs, we obtain that $\text{Sol}(\texttt{D}_0^0) \subseteq \text{Sol}(\texttt{D}_{p_{n-1}}^{n-1})$. It follows by construction that $\texttt{D}_{p_{n-1}}^{n-1}$ is the same as the DD $\texttt{D}$ that is constructed via Algorithm 3 according to the proposition

statement. Further, Proposition 3.2 implies that $\text{conv}(\mathcal{G}) \subseteq \text{conv}(\text{Sol}(\mathtt{D}_0^0))$. Combining the above three statements, we obtain that $\text{conv}(\mathcal{G}) \subseteq \text{conv}(\text{Sol}(\mathtt{D}))$.

□

The following example demonstrates the steps of Algorithm 3 construct relaxed DDs with a specified size.

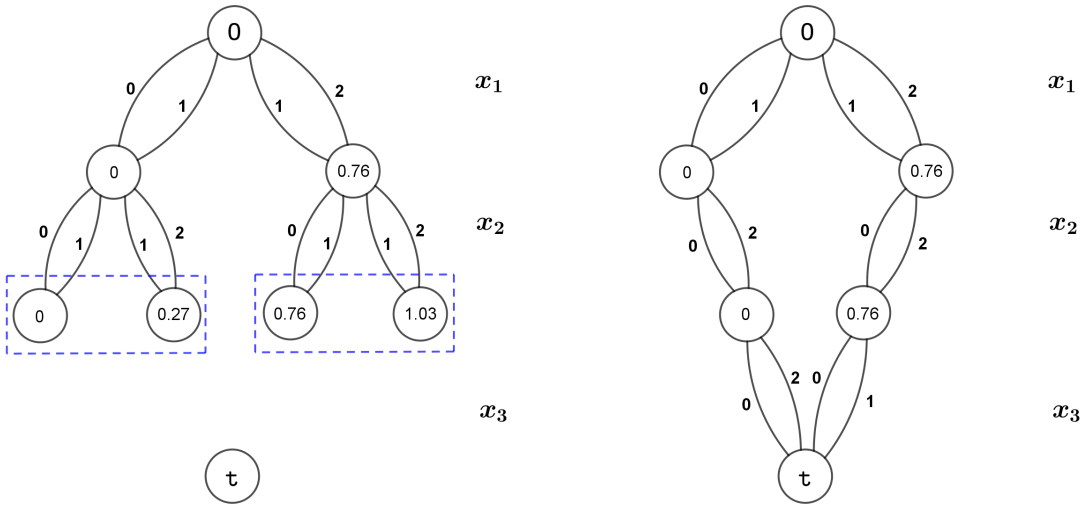
*Example 3.2* Consider the MINLP set $\mathcal{G}$ studied in Example 3.1. Assume that a width limit $\omega = 2$ is imposed. Because the DD constructed via Algorithm 2 in Figure 3.1 does not satisfy this width limit, we use Algorithm 3 to apply a merging operation at node layers whose size exceed the width limit. The process of constructing the DD through merging nodes is shown in Figure 3.2. Following the top-down construction steps, the first and second node layers satisfy the width limit. After the nodes in the third node layer are created through lines 4–7 of the algorithm, we observe that the number of nodes in this layer exceeds the width limit, which calls for the merging oracle $\mathtt{Merge}(\omega, \mathtt{U}_3)$ in line 8. According to this oracle, we merge nodes with state values 0 and 0.27, as well as the nodes with state values 0.76 and 1.03, as depicted in dashed boxes in Figure 3.2a. This merging operation selects the minimum state value for each pair of merged nodes and updates the incoming arcs accordingly, as shown in the second layer of Figure 3.2b. During this process, the arcs with labels 1 are removed according to Proposition 3.1 as they are the middle arcs in a group of parallel arcs. Since the number of nodes in this layer satisfies the width limit, the algorithm proceeds to the last iteration to create the arcs that are connected to the terminal node. The resulting DD, which we refer to as $\mathtt{D}^2$, is shown in Figure 3.4b.

It is easy to verify that the convex hull of solutions of $\mathtt{D}^2$ is equal to the convex hull of the union of hyper-rectangles $\big\{[0,1]\times[0,2]\times[0,2], [1,2]\times[0,2]\times[0,1]\big\}$. This yields the polyhedral convex hull $\text{conv}(\text{Sol}(\mathtt{D}^2)) = \big\{(x_1,x_2,x_3) \in [0,2]^3 \,\big|\, x_1 + x_3 \leq 3\big\}$. This set provides a relaxation for the convex hull of the original set $\mathcal{G}$ with strict inclusion, i.e., $\text{conv}(\mathcal{G}) \subset \text{conv}(\text{Sol}(\mathtt{D}^2))$. Furthermore, this set provides a relaxation for the convex hull of the solutions of the DD constructed in Example 3.1, i.e., $\text{conv}(\text{Sol}(\mathtt{D}^1)) \subset \text{conv}(\mathcal{G}) \subset \text{conv}(\text{Sol}(\mathtt{D}^2))$. This verifies the result of Proposition 3.3, demonstrating that applying a merging operation to the DD associated with sets that contain separable functions leads to a relaxation of the initial DD. ■

### 3.3 Relaxed DD for Non-Separable Constraints

In this section, we consider a more general case for constructing DDs corresponding to a constraint of a MINLP, where the functions involved in the constraint are not separable. In the separable case discussed in Section 3.2, the state values calculated for a node layer of the DD depend only on the state values of the nodes in the previous layer and the labels of the arcs in that layer, thereby satisfying a Markovian property. This property simplifies the construction of the DD. However, in the non-separable case, the state values calculated for a node layer can be impacted by the arc labels from any of the preceding layers, due to the interactions between variables in the non-separable terms. As a result, constructing a relaxed DD in this context is more challenging. In what follows, we illustrate the methods for constructing such relaxed DDs.

**Definition 3.1** Consider a DD $\mathtt{D} = (\mathtt{U}, \mathtt{A}, \mathtt{l}(.))$ with the variable ordering $1, \ldots, n$. Given a layer $i \in [n]$ and a node $\mathtt{v} \in \mathtt{U}_i$, we define $\mathtt{A}_j(\mathtt{v})$ to be the set of arcs in arc layer $j \in [i-1]$ that lie on a path from the root node to node $\mathtt{v}$. Further, we define $\mathcal{D}_j(\mathtt{v}) = [\mathcal{D}_j(\mathtt{v})\downarrow, \mathcal{D}_j(\mathtt{v})\uparrow]$ for $j \in [i-1]$ to be *the sub-domain of variable $x_j$ relative to node* $\mathtt{v}$. In this definition, the domain lower bound is calculated as $\mathcal{D}_j(\mathtt{v})\downarrow = \min_{\mathtt{a}\in\mathtt{A}_j(\mathtt{v})}\{\mathtt{l}(\mathtt{a})\}$ and the domain upper bound is calculated as $\mathcal{D}_j(\mathtt{v})\uparrow = \max_{\mathtt{a}\in\mathtt{A}_j(\mathtt{v})}\{\mathtt{l}(\mathtt{a})\}$.




(a) Merging operation at node layer 3

(b) Relaxed DD representation

Fig. 3.2: Relaxed DD for the set in Example 3.4.

Determining the variable sub-domains relative to a node requires accounting for all paths in the DD that pass through that node. This can become computationally intensive, especially for large-scale DDs. The following proposition offers an efficient method for calculating these bounds by leveraging the top-down construction process of DDs.

**Proposition 3.4** *Consider a DD* $\mathtt{D} = (\mathtt{U}, \mathtt{A}, \mathtt{l}(.))$ *with the variable ordering* $1, \dots, n$. *Given a node layer* $i \in [n]$ *and a node* $\mathtt{v} \in \mathtt{U}_i$, *we can calculate* $\mathcal{D}_j(\mathtt{v}) = [\mathcal{D}_j(\mathtt{v})\downarrow, \mathcal{D}_j(\mathtt{v})\uparrow]$ *for* $j \in [i-1]$ *as follows:*

*(i) If* $j = i - 1$*:*

$$\mathcal{D}_j(\mathtt{v})\downarrow = \min_{\mathtt{a} \in \delta^-(\mathtt{v})} \{\mathtt{l}(\mathtt{a})\} \tag{3.1a}$$

$$\mathcal{D}_j(\mathtt{v})\uparrow = \max_{\mathtt{a} \in \delta^-(\mathtt{v})} \{\mathtt{l}(\mathtt{a})\}. \tag{3.1b}$$

*(ii) If* $j < i - 1$*:*

$$\mathcal{D}_j(\mathtt{v})\downarrow = \min_{\mathtt{a} \in \delta^-(\mathtt{v})} \{\mathcal{D}_j(\mathtt{t}(\mathtt{a}))\downarrow\} \tag{3.2a}$$

$$\mathcal{D}_j(\mathtt{v})\uparrow = \max_{\mathtt{a} \in \delta^-(\mathtt{v})} \{\mathcal{D}_j(\mathtt{t}(\mathtt{a}))\uparrow\}. \tag{3.2b}$$

*Proof* We prove the results for the lower bound equations (3.1a) and (3.2a) as the proof arguments for the upper bound equations are similar.

(i) Assume that $j = i - 1$. In this case, node $\mathtt{v} \in \mathtt{U}_i$ must be the head node of each arc $\mathtt{a} \in \mathtt{A}_j(\mathtt{v})$ that is on a path from the root node of $\mathtt{D}$ to node $\mathtt{v}$. That is, $\mathtt{A}_j(\mathtt{v}) \subseteq \delta^-(\mathtt{v})$. On the other hand, using the DD structure that each node at layer $i \in \{2, \dots, n-1\}$ is connected to at least one node in the previous layer, we obtain that each arc $\mathtt{a} \in \delta^-(\mathtt{v})$ must be on a path from the root node to $\mathtt{v}$, *i.e.*, $\mathtt{A}_j(\mathtt{v}) \supseteq \delta^-(\mathtt{v})$. Therefore, $\mathtt{A}_j(\mathtt{v}) = \delta^-(\mathtt{v})$. As a result, by definition of $\mathcal{D}_j(\mathtt{v})\downarrow$, we have $\mathcal{D}_j(\mathtt{v})\downarrow = \min_{\mathtt{a} \in \mathtt{A}_j(\mathtt{v})}\{\mathtt{l}(\mathtt{a})\} = \min_{\mathtt{a} \in \delta^-(\mathtt{v})} \{\mathtt{l}(\mathtt{a})\}$.

(ii) Assume that $j < i-1$. For the forward direction, consider arc $\mathtt{a} \in \mathtt{A}_j(\mathtt{v})$, *i.e.*, $\mathtt{a}$ is on a path from the root node of $\mathtt{D}$ to node $\mathtt{v}$. This path passes through some node $\mathtt{u}$ in the node layer $i-1$ that is connected to node $\mathtt{v}$ via arc $\dot{\mathtt{a}}$, *i.e.*, $\mathtt{u} = \mathtt{t}(\dot{\mathtt{a}})$ and $\dot{\mathtt{a}} \in \delta^-(\mathtt{v})$. As a result, $\mathtt{a} \in \mathtt{A}_j(\mathtt{u})$. Using this relation, we obtain that $\mathtt{A}_j(\mathtt{v}) \subseteq \bigcup_{\dot{\mathtt{a}} \in \delta^-(\mathtt{v})} \mathtt{A}_j(\mathtt{t}(\dot{\mathtt{a}}))$. Therefore, $\mathcal{D}_j(\mathtt{v})\downarrow = \min_{\mathtt{a} \in \mathtt{A}_j(\mathtt{v})}\{\mathtt{l}(\mathtt{a})\} \geq \min_{\dot{\mathtt{a}} \in \delta^-(\mathtt{v})} \left\{ \min_{\hat{\mathtt{a}} \in \mathtt{A}_j(\mathtt{t}(\dot{\mathtt{a}}))}\{\mathtt{l}(\hat{\mathtt{a}})\} \right\} = \min_{\dot{\mathtt{a}} \in \delta^-(\mathtt{v})} \left\{ \mathcal{D}_j(\mathtt{t}(\dot{\mathtt{a}}))\downarrow \right\}$, where the first equality follows from the definition of $\mathcal{D}_j(\mathtt{v})\downarrow$, the inequality follows from the previously derived inclusion argument, and the last equality follows from the definition of $\mathcal{D}_j(\mathtt{t}(\dot{\mathtt{a}}))\downarrow$. By design, node $\mathtt{v}$ is connected to a node $\mathtt{u}$ in the previous node layer $\mathtt{U}_{i-1}$. For the reverse direction, assume that $\min_{\mathtt{a} \in \delta^-(\mathtt{v})} \left\{ \mathcal{D}_j(\mathtt{t}(\mathtt{a}))\downarrow \right\}$ is achieved by arc $\mathtt{a}^* \in \delta^-(\mathtt{v})$, *i.e,.* $\mathcal{D}_j(\mathtt{t}(\mathtt{a}^*))\downarrow = \mathtt{l}(\bar{\mathtt{a}})$ for some $\bar{\mathtt{a}} \in \mathtt{A}_j(\mathtt{t}(\mathtt{a}^*))$. This means that $\bar{\mathtt{a}}$ is on a path from the root node of $\mathtt{D}$ to node $\mathtt{t}(\mathtt{a}^*)$, which is the tail node of arc $\mathtt{a}^*$ that connects this node to $\mathtt{v}$. As a result, $\bar{\mathtt{a}}$ is on a path from the root node of $\mathtt{D}$ to node $\mathtt{v}$, *i.e.*, $\bar{\mathtt{a}} \in \mathtt{A}_j(\mathtt{v})$. We can write that $\mathcal{D}_j(\mathtt{v})\downarrow = \min_{\mathtt{a} \in \mathtt{A}_j(\mathtt{v})}\{\mathtt{l}(\mathtt{a})\} \leq \mathtt{l}(\bar{\mathtt{a}}) = \mathcal{D}_j(\mathtt{t}(\mathtt{a}^*))\downarrow = \min_{\mathtt{a} \in \delta^-(\mathtt{v})} \left\{ \mathcal{D}_j(\mathtt{t}(\mathtt{a}))\downarrow \right\}$, where the first equality follows from the definition of $\mathcal{D}_j(\mathtt{v})\downarrow$, the inequality holds because $\bar{\mathtt{a}} \in \mathtt{A}_j(\mathtt{v})$, and the second and third equalities follow from the assumptions. Combining both directions, we obtain the equation 3.2a.

□

The recursive relations (3.1a)–(3.1b) and (3.2a)–(3.2b) provide an efficient method to compute the values of $\mathcal{D}_j(\mathtt{v})$ during the construction of DD layers by updating the lower and upper bound values of its interval as each new node is created. The next corollary gives the complexity of these calculations. The proof is omitted, as it directly follows from the application of the recursive formulas across consecutive DD layers.

**Corollary 3.1** *Consider a DD* $\mathtt{D} = (\mathtt{U}, \mathtt{A}, \mathtt{l}(.))$ *with the variable ordering* $1, \ldots, n$. *Given a node layer* $i \in [n]$ *and an arc layer* $j \in [i-1]$, *the sub-domain of variable* $x_j$ *relative to all nodes in* $\mathtt{U}_i$ *can be calculated in* $\mathcal{O}(\sum_{k=j}^{i-1} |\mathtt{A}_k|)$ *using the recursive relations* (3.1a)–(3.1b) *and* (3.2a)–(3.2b). □

Corollary 3.1 shows that the sub-domains of a particular variable in layer $j$ relative to all nodes in layer $i$, for any $j < i$, can be computed in linear time in the number of arcs between layers $j$ and $i$. This method makes the calculation of these interval values simple and fast during the DD construction process.

Next, we introduce an algorithm to build relaxed DDs for non-separable functions. Consider $\mathcal{G} = \left\{ \boldsymbol{x} \in \prod_{i=1}^n \mathcal{D}_i \,\middle|\, g(\boldsymbol{x}) \leq b \right\}$, where $g(\boldsymbol{x}) = \sum_{j=1}^q g_j(\boldsymbol{x}_{H_j})$ such that $g_j(\boldsymbol{x}_{H_j}) : \mathbb{R}^{|H_j|} \to \mathbb{R}$ is a non-separable function that contains variables with indices in $H_j \subseteq [n]$. For each $j \in [q]$, define $H_j^{\max} = \max_{k \in H_j}\{k\}$, which represents the last DD layer that involves a variable in the non-separable function $g_j(\boldsymbol{x}_{H_j})$.

Similarly to Section 3.2, we will first present an algorithm to construct a DD-based relaxation for the solutions of $\mathcal{G}$. Then, we will discuss how merging operations can be incorporated to produce relaxed DDs of the desired size.

**Proposition 3.5** *Consider* $\mathcal{G} = \left\{ \boldsymbol{x} \in \prod_{i=1}^n \mathcal{D}_i \,\middle|\, g(\boldsymbol{x}) \leq b \right\}$, *where* $g(\boldsymbol{x}) = \sum_{j=1}^q g_j(\boldsymbol{x}_{H_j})$ *such that* $g_j(\boldsymbol{x}_{H_j}) : \mathbb{R}^{|H_j|} \to \mathbb{R}$ *is a non-separable function that contains variables with indices in* $H_j \subseteq [n]$. *Let* $\mathtt{D}$ *be the DD constructed via Algorithm 4 for some sub-domain partitions* $\mathcal{D}_i^j$ *with* $j \in L_i$ *for* $i \in [n]$. *Then,* $\mathrm{conv}(\mathcal{G}) \subseteq \mathrm{conv}(\mathrm{Sol}(\mathtt{D}))$.

*Proof* We show that $\mathcal{G} \subseteq \mathrm{conv}(\mathrm{Sol}(\mathtt{D}))$. Pick $\bar{\boldsymbol{x}} \in \mathcal{G}$. It follows from the definition of $\mathcal{G}$ that $\sum_{k=1}^q g_k(\bar{\boldsymbol{x}}_{H_k}) \leq b$. For each $i \in [n]$, let $j_i^*$ be the index of a sub-domain partition $\mathcal{D}_i^{j_i^*}$ in $L_i$ such that $\bar{x}_i \in \mathcal{D}_i^{j_i^*}$. This index exists because $\bar{x}_i \in \mathcal{D}_i = \bigcup_{j \in L_i} \mathcal{D}_i^j$, where the inclusion follows from the fact that $\bar{\boldsymbol{x}} \in \mathcal{G}$, and the equality follows from the definition of sub-domain partitions.

Next, we show that $\mathtt{D}$ includes a node sequence $\{\mathtt{u}_1, \mathtt{u}_2, \ldots, \mathtt{u}_{n+1}\}$, where $\mathtt{u}_i \in \mathtt{U}_i$ for $i \in [n+1]$, such that each node $\mathtt{u}_i$ is connected to $\mathtt{u}_{i+1}$ via two arcs with labels $\mathcal{D}_i^{j_i^*}\downarrow$ and $\mathcal{D}_i^{j_i^*}\uparrow$ for each $i \in [n]$.

**Algorithm 4:** Relaxed DD for a non-separable constraint

**Data:** Set $\mathcal{G} = \left\{ \boldsymbol{x} \in \prod_{i=1}^{n} \mathcal{D}_i \,\middle|\, g(\boldsymbol{x}) \leq b \right\}$, where $g(\boldsymbol{x}) = \sum_{k=1}^{q} g_k(\boldsymbol{x}_{H_k})$ where $g_k(\boldsymbol{x}_{H_k}) : \mathbb{R}^{|H_k|} \to \mathbb{R}$ is a non-separable function that contains variables with indices in $H_k$, and the sub-domain partitions $\mathcal{D}_i^j$ with $j \in L_i$ for $i \in [n]$

**Result:** A DD $\mathtt{D}$

1 create the root node $\mathtt{r}$ in the node layer $\mathtt{U}_1$ with state value $\mathtt{s}(\mathtt{r}) = 0$
2 create the terminal node $\mathtt{t}$ in the node layer $\mathtt{U}_{n+1}$
3 **forall** $i \in [n]$, $\mathtt{u} \in \mathtt{U}_i$, $j \in L_i$ **do**
4  **forall** $k \in [q]$ **do**
5  **if** $i = H_k^{max}$ **then**
6  calculate $\eta_k$ such that $\eta_k \leq g_k(\boldsymbol{x}_{H_k})$ for all $x_i \in \mathcal{D}_i^j$ and $x_l \in \mathcal{D}_l(\mathtt{u})$ for $l \in H_k \setminus \{i\}$
7  **else**
8  calculate $\eta_k = 0$
9  calculate $\xi = \mathtt{s}(\mathtt{u}) + \sum_{k=1}^{q} \eta_k$
10  **if** $i < n$ **then**
11  create a node $\mathtt{v}$ with state value $\mathtt{s}(\mathtt{v}) = \xi$ (if it does not already exist) in the node layer $\mathtt{U}_{i+1}$
12  add two arcs from $\mathtt{u}$ to $\mathtt{v}$ with label values $\mathcal{D}_i^j\downarrow$ and $\mathcal{D}_i^j\uparrow$
13  update $\mathcal{D}_k(\mathtt{v})$ for all $k$ such that $k \in H_l$ for some $l > i$
14  **else if** $\xi \leq b$ **then**
15  add two arcs from $\mathtt{u}$ to the terminal node $\mathtt{t}$ with label values $\mathcal{D}_i^j\downarrow$ and $\mathcal{D}_i^j\uparrow$

Using an argument similar to that in the proof of Proposition 3.2 together with the instructions in lines 10–12 of Algorithm 4, we conclude that a node sequence $\{\mathtt{u}_1, \mathtt{u}_2, \ldots, \mathtt{u}_n\}$ that satisfies the above conditions exists. We prove this sequence is completed by two arcs directed from node $\mathtt{u}_n$ to the terminal node $\mathtt{t}$ with labels $\mathcal{D}_n^{j_n^*}\downarrow$ and $\mathcal{D}_n^{j_n^*}\uparrow$. To this end, we need to show that the condition in line 14 of Algorithm 4 is satisfied.

We use induction on layer $i \in [n]$ to prove $\mathtt{s}(\mathtt{u}_i) \leq \sum_{k \in [q]: H_k^{\max} < i} g_k(\bar{\boldsymbol{x}}_{H_k})$. In words, we show that the state value of each node $\mathtt{u}_i$ in the previously picked sequence is no greater than the summation of non-separable function terms $g_k(\bar{\boldsymbol{x}}_{H_k})$ whose variables have already been visited in the previous layers of the DD. The induction base for $i = 1$ follows from line 1 of Algorithm 4 because $\mathtt{s}(\mathtt{u}_1) = \mathtt{s}(\mathtt{r}) = 0$, and the fact that $H_k^{\max} > 0$ for all $k \in [q]$, implying that $\sum_{k \in [q]: H_k^{\max} < i} g_k(\bar{\boldsymbol{x}}_{H_k}) = 0$ by default. For the inductive hypothesis, assume that $\mathtt{s}(\mathtt{u}_{\hat{i}}) \leq \sum_{k \in [q]: H_k^{\max} < \hat{i}} g_k(\bar{\boldsymbol{x}}_{H_k})$ for $\hat{i} \in [n-1]$. For the inductive step, we show that $\mathtt{s}(\mathtt{u}_{\hat{i}+1}) \leq \sum_{k \in [q]: H_k^{\max} < \hat{i}+1} g_k(\bar{\boldsymbol{x}}_{H_k})$. It follows from lines 9–11 of Algorithm 4 that $\mathtt{s}(\mathtt{u}_{\hat{i}+1}) = \mathtt{s}(\mathtt{u}_{\hat{i}}) + \sum_{k=1}^{q} \eta_k$, where $\eta_k$ is a lower bound for $g_k(\boldsymbol{x}_{H_k})$ for all $x_{\hat{i}} \in \mathcal{D}_{\hat{i}}^{j_{\hat{i}}^*}$ and $x_l \in \mathcal{D}_l(\mathtt{u}_{\hat{i}})$ for all $l \in H_k \setminus \{\hat{i}\}$. Furthermore, we have that $\sum_{k \in [q]: H_k^{\max} < \hat{i}+1} g_k(\bar{\boldsymbol{x}}_{H_k}) = \sum_{k \in [q]: H_k^{\max} < \hat{i}} g_k(\bar{\boldsymbol{x}}_{H_k}) + \sum_{k \in [q]: H_k^{\max} = \hat{i}} g_k(\bar{\boldsymbol{x}}_{H_k})$. Therefore, to prove the relation in the inductive step, it suffices to show that $\sum_{k=1}^{q} \eta_k \leq \sum_{k \in [q]: H_k^{\max} = \hat{i}} g_k(\bar{\boldsymbol{x}}_{H_k})$, because $\mathtt{s}(\mathtt{u}_{\hat{i}}) \leq \sum_{k \in [q]: H_k^{\max} < \hat{i}} g_k(\bar{\boldsymbol{x}}_{H_k})$ by the inductive hypothesis. We can rewrite the right-hand-side $\sum_{k \in [q]: H_k^{\max} = \hat{i}} g_k(\bar{\boldsymbol{x}}_{H_k})$ of the above inequality as $\sum_{k=1}^{q} \phi_k$ where $\phi_k = g_k(\bar{\boldsymbol{x}}_{H_k})$ if $H_k^{\max} = \hat{i}$, and $\phi_k = 0$ otherwise. As a result, it is sufficient to show that $\eta_k \leq \phi_k$ for each $k \in [q]$. There are two cases.

For the first case, assume that $H_k^{\max} \neq \hat{i}$. Then, it follows from line 7–8 of Algorithm 4 that $\eta_k = 0$. Additionally, the definition of $\phi_k$ implies that $\phi_k = 0$, which proves the desired inequality.

For the second case, assume that $H_k^{\max} = \hat{\imath}$. It follows from the arguments in the first paragraph of the proof that $\bar{x}_i \in \mathcal{D}_i^{j_i^*}$ for all $i \in [n]$. Further, we have established that $\{\mathtt{u}_1, \mathtt{u}_2, \ldots, \mathtt{u}_{\hat{\imath}}\}$ is a node sequence such that each node $\mathtt{u}_l$ is connected to $\mathtt{u}_{l+1}$ via two arcs $\mathtt{a}_l^1$ and $\mathtt{a}_l^2$ with labels $\mathcal{D}_l^{j_l^*}\downarrow$ and $\mathcal{D}_l^{j_l^*}\uparrow$ for $l = 1, \ldots, \hat{\imath}-1$. By definition of $H_k^{\max}$, we must have that $H_k \setminus \{\hat{\imath}\} \subseteq \{1, \ldots, \hat{\imath}-1\}$. Therefore, for each $l \in H_k \setminus \{\hat{\imath}\}$, the arcs $\mathtt{a}_l^1$ and $\mathtt{a}_l^2$ must be in $\mathtt{A}_l(\mathtt{u}_{\hat{\imath}})$ because they are on some paths from $\mathtt{r}$ to $\mathtt{u}_{\hat{\imath}}$. We can write that $\mathcal{D}_l(\mathtt{u}_{\hat{\imath}})\downarrow \leq \mathcal{D}_l^{j_l^*}\downarrow \leq \bar{x}_l$, where the first inequality follows from the definition of $\mathcal{D}_l(\mathtt{u}_{\hat{\imath}})$ in Definition 3.1, and the second inequality follows from the fact that $\bar{x}_l \in \mathcal{D}_l^{j_l^*}$. Similarly, we can write that $\mathcal{D}_l(\mathtt{u}_{\hat{\imath}})\uparrow \geq \mathcal{D}_l^{j_l^*}\uparrow \geq \bar{x}_l$. This yields $\bar{x}_l \in \mathcal{D}_l(\mathtt{u}_{\hat{\imath}})$. Additionally, we have argued that $\bar{x}_{\hat{\imath}} \in \mathcal{D}_{\hat{\imath}}^{j_{\hat{\imath}}^*}$. Therefore, it follows from the definition of $\eta_k$ that $\eta_k \leq g_k(\bar{\boldsymbol{x}}_{H_k}) = \phi_k$, obtaining the desired inequality.

For the next step of the proof, we show that the condition in line 14 of Algorithm 4 is satisfied, i.e., $\xi^* = \mathtt{s}(\mathtt{u}_n) + \sum_{k=1}^q \eta_k^* \leq b$, where we use $^*$ to distinguish the values of $\xi$ and $\eta_k$ calculated at the last layer $i = n$ from those calculated in the previous part for layers $i < n$. It follows from the above induction result that $\mathtt{s}(\mathtt{u}_n) \leq \sum_{k\in[q]:H_k^{\max}<n} g_k(\bar{\boldsymbol{x}}_{H_k})$. Using an argument similar to that used in the above induction steps, we can show that, for each $k \in [q]$, we have $\eta_k^* \leq g_k(\bar{\boldsymbol{x}}_{H_k})$ if $H_k^{\max} = n$, and $\eta_k^* = 0$ otherwise. Aggregating the above two inequalities, we obtain that $\mathtt{s}(\mathtt{u}_n) + \sum_{k=1}^q \eta_k^* \leq \sum_{k\in[q]:H_k^{\max}<n} g_k(\bar{\boldsymbol{x}}_{H_k}) + \sum_{k\in[q]:H_k^{\max}=n} g_k(\bar{\boldsymbol{x}}_{H_k})$. The right-hand-side of this inequality can be rewritten as $\sum_{k\in[q]} g_k(\bar{\boldsymbol{x}}_{H_k})$ because $H_k^{\max} \leq n$ for all $k \in [q]$ by definition. On the other hand, the definition of $\mathcal{G}$ implies that $\sum_{k=1}^q g_k(\bar{\boldsymbol{x}}_{H_k}) \leq b$. Combining the above inequalities, we conclude that $\mathtt{s}(\mathtt{u}_n) + \sum_{k=1}^q \eta_k^* = \xi^* \leq b$, which satisfies the condition in line 14 of Algorithm 4. Therefore, line 15 of Algorithm 4 implies that two arcs with label values $\mathcal{D}_n^{j_n^*}\downarrow$ and $\mathcal{D}_n^{j_n^*}\uparrow$ connect node $\mathtt{u}_n$ to the terminal node $\mathtt{t}$ which can be considered as $\mathtt{u}_{n+1}$, completing the desired node sequence.

Now consider the collection of points $\tilde{\boldsymbol{x}}^\kappa$ for $\kappa \in [2^n]$ encoded by all paths composed of the above-mentioned pair of arcs with labels $\mathcal{D}_i^{j_i^*}\downarrow$ and $\mathcal{D}_i^{j_i^*}\uparrow$ between each two consecutive nodes $\mathtt{u}_i$ and $\mathtt{u}_{i+1}$ in the sequence $\{\mathtt{u}_1, \mathtt{u}_2, \ldots, \mathtt{u}_{n+1}\}$. Therefore, $\tilde{\boldsymbol{x}}^\kappa \in \mathrm{Sol}(\mathtt{D})$ for $\kappa \in [2^n]$. It is clear that these points form the vertices of an $n$-dimensional hyper-rectangle defined by $\prod_{i=1}^n [\mathcal{D}_i^{j_i^*}\downarrow, \mathcal{D}_i^{j_i^*}\uparrow]$. By construction, we have that $\bar{\boldsymbol{x}} \in \prod_{i=1}^n [\mathcal{D}_i^{j_i^*}\downarrow, \mathcal{D}_i^{j_i^*}\uparrow]$, i.e., $\bar{\boldsymbol{x}}$ is a point inside the above hyper-rectangle. As a result, $\bar{\boldsymbol{x}}$ can be represented as a convex combination of the vertices $\tilde{\boldsymbol{x}}^\kappa$ for $\kappa \in [2^n]$ of the hyper-rectangle, yielding $\bar{\boldsymbol{x}} \in \mathrm{conv}(\mathrm{Sol}(\mathtt{D}))$.

□

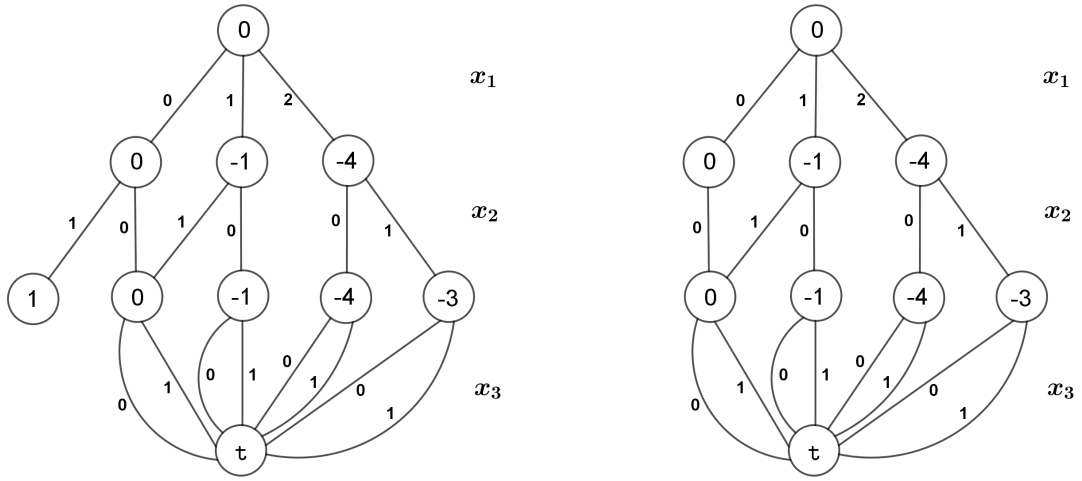
The following example illustrates the steps of Algorithm 5 for an MINLP set with a non-separable term.

*Example 3.3* Consider a nonconvex MINLP set $\mathcal{G} = \left\{ \boldsymbol{x} \in \prod_{i=1}^3 \mathcal{D}_i \,\middle|\, -x_1^2 + x_2 - x_1 x_3 \leq -1 \right\}$, where $\mathcal{D}_1 = \{0, 1, 2\}$, $\mathcal{D}_2 = \{0, 1\}$, and $\mathcal{D}_3 = [0, 1]$. Following the definition of sets studied in this section, we have $g(\boldsymbol{x}) = g_1(x_1) + g_2(x_2) + g_3(x_1, x_3)$ where $g_1(x_1) = -x_1^2$, $g_2(x_2) = x_2$, and $g_3(x_1, x_3) = -x_1 x_3$. Further, we obtain $H_1^{\max} = 1$, $H_2^{\max} = 2$, and $H_3^{\max} = 3$. The feasible region of $\mathcal{G}$ can be represented as the following union of hyper-rectangles: $\mathcal{G} = \big\{\{1\} \times \{0\} \times [0, 1], \{1\} \times \{1\} \times [0, 1], \{2\} \times \{0\} \times [0, 1], \{2\} \times \{1\} \times [0, 1]\big\}$. Therefore, we can obtain the convex hull of this set as $\mathrm{conv}(\mathcal{G}) = [1, 2] \times [0, 1] \times [0, 1]$.

To obtain a relaxation for this set based on DDs, we apply Algorithm 4 with sub-domain partitions $\mathcal{D}_1^1 = [0, 0]$, $\mathcal{D}_1^2 = [1, 1]$, $\mathcal{D}_1^3 = [2, 2]$ for $x_1$ as a discrete variable, $\mathcal{D}_2^1 = [0, 0]$, $\mathcal{D}_2^2 = [1, 1]$ for $x_2$ as a binary variable, and $\mathcal{D}_3^1 = [0, 1]$ for $x_3$ as a continuous variable. Following the steps of the algorithm, we obtain the DD $\mathtt{D}^3$ presented in Figure 3.3, where the numbers next to each arc represent the arc label, and the numbers inside each node show the state value of the node. To

illustrate the state value computation of non-separable terms, consider the node with state value 1 at node layer 3, which we refer to as $\mathtt{u} \in \mathtt{U}_3$ with $\mathtt{s}(\mathtt{u}) = 1$. In line 6 of the algorithm, a lower bound $\eta_3$ for $g_3(x_1, x_3)$ must be calculated over the domain $x_3 \in \mathcal{D}_3^1$ and $x_1 \in \mathcal{D}_1(\mathtt{u})$. Since there is one path from the root node to $\mathtt{u}$ with arc label 0 for variable $x_1$, we obtain $\mathcal{D}_1(\mathtt{u}) = [0, 0]$. Therefore, $\eta_3$ can be set to 0. Similarly, we set $\eta_1 = \eta_2 = 0$ in line 8 of the algorithm because $H_1^{\max} \neq 3$ and $H_2^{\max} \neq 3$. Therefore, we obtain $\xi = \mathtt{s}(\mathtt{u}) + \sum_{i=1}^{3} \eta_3 = 1$. Since $\xi \nleq -1$, the condition in line 14 of the algorithm is not satisfied, implying that node $\mathtt{u}$ is not connected to the terminal node.

Now consider the node with state value 0 at node layer 3, which we refer to as $\mathtt{v} \in \mathtt{U}_3$ with $\mathtt{s}(\mathtt{v}) = 0$. In line 6 of the algorithm, a lower bound $\eta_3$ for $g_3(x_1, x_3)$ must be calculated over the domain $x_3 \in \mathcal{D}_3^1$ and $x_1 \in \mathcal{D}_1(\mathtt{v})$. Since there are two paths from the root node to $\mathtt{v}$ with arc labels 0 and 1 for variable $x_1$, we obtain $\mathcal{D}_1(\mathtt{v}) = [0, 1]$. Therefore, $\eta_3$ can be set to $-1$. Similarly, we set $\eta_1 = \eta_2 = 0$ in line 8 of the algorithm because $H_1^{\max} \neq 3$ and $H_2^{\max} \neq 3$. Therefore, we obtain $\xi = \mathtt{s}(\mathtt{v}) + \sum_{i=1}^{3} \eta_3 = -1$. Since $\xi \leq -1$, the condition in line 14 of the algorithm is satisfied, implying that node $\mathtt{v}$ is connected to the terminal node by two arcs with labels 0 and 1. The calculation for other nodes can be carried out similarly, which yields the DD in Figure 3.3a. Since there is a node at layer 3 that is not connected to the terminal node, we can reduce the size of the DD by removing that node and its incoming arc as it does not lead to a feasible path, which yields a reduced DD in Figure 3.3b. It is easy to verify that $\text{conv}(\text{Sol}(\mathtt{D}^3)) = \big\{(x_1, x_2, x_3) \in [0, 2] \times [0, 1] \times [0, 1] \,\big|\, x_2 - x_1 \leq 0\big\}$. This set provides a relaxation for the convex hull of the original set $\mathcal{G}$ with strict inclusion, i.e., $\text{conv}(\mathcal{G}) \subset \text{conv}(\text{Sol}(\mathtt{D}^3))$. ■




(a) Original DD representation

(b) Reduced DD representation

Fig. 3.3: Relaxed DD for the set in Example 3.3.

Similarly to the case with separable functions discussed in Section 3.2, we can control the size of the DD constructed via Algorithms 4 by adjusting the number of sub-domain partitions and employing a merging operation. In particular, Algorithm 5 incorporates a merging oracle after the nodes in each DD layer are created to ensure that the prescribed width limit is satisfied. The correctness of this algorithm in providing a relaxation for the convex hull of the solution set described by a non-separable function is established in Theorem 3.2. This method is further illustrated in Example 3.4.

**Algorithm 5:** Relaxed DD for a non-separable constraint

**Data:** Set $\mathcal{G} = \left\{ \boldsymbol{x} \in \prod_{i=1}^{n} \mathcal{D}_i \,\middle|\, g(\boldsymbol{x}) \leq b \right\}$ with $g(\boldsymbol{x}) = \sum_{k=1}^{q} g_k(\boldsymbol{x}_{H_k})$ where $g_k(\boldsymbol{x}_{H_k}) : \mathbb{R}^{|H_k|} \to \mathbb{R}$ is a non-separable function that contains variables with indices in $H_k$, the sub-domain partitions $\mathcal{D}_i^j$ with $j \in L_i$ for $i \in [n]$, a width limit $\omega$, and a merging oracle $\texttt{Merge}(\omega, \texttt{V})$

**Result:** A DD $\texttt{D}$

1 create the root node $\texttt{r}$ in the node layer $\texttt{U}_1$ with state value $\texttt{s}(\texttt{r}) = 0$
2 create the terminal node $\texttt{t}$ in the node layer $\texttt{U}_{n+1}$
3 **forall** $i \in [n]$ **do**
4  **forall** $\texttt{u} \in \texttt{U}_i$, $j \in L_i$ **do**
5  **forall** $k \in [q]$ **do**
6  **if** $i = H_k^{max}$ **then**
7  calculate $\eta_k$ such that $\eta_k \leq g_k(\boldsymbol{x}_{H_k})$ for all $x_i \in \mathcal{D}_i^j$ and $x_l \in \mathcal{D}_l(\texttt{u})$ for $l \in H_k \setminus \{i\}$
8  **else**
9  calculate $\eta_k = 0$
10  calculate $\xi = \texttt{s}(\texttt{u}) + \sum_{k=1}^{q} \eta_k$
11  **if** $i < n$ **then**
12  create a node $\texttt{v}$ with state value $\texttt{s}(\texttt{v}) = \xi$ (if it does not already exist) in the node layer $\texttt{U}_{i+1}$
13  add two arcs from $\texttt{u}$ to $\texttt{v}$ with label values $\mathcal{D}_i^j\!\downarrow$ and $\mathcal{D}_i^j\!\uparrow$
14  update $\mathcal{D}_k(\texttt{v})$ for all $k$ such that $k \in H_l$ for some $l > i$
15  **else if** $\xi \leq b$ **then**
16  add two arcs from $\texttt{u}$ to the terminal node $\texttt{t}$ with label values $\mathcal{D}_i^j\!\downarrow$ and $\mathcal{D}_i^j\!\uparrow$
17  **if** $i < n$ **then**
18  merge nodes at this layer to satisfy the prescribed width limit by invoking the merging oracle $\texttt{Merge}(\omega, \texttt{U}_{i+1})$

**Theorem 3.2** *Consider* $\mathcal{G} = \left\{ \boldsymbol{x} \in \prod_{i=1}^{n} \mathcal{D}_i \,\middle|\, g(\boldsymbol{x}) \leq b \right\}$ *with* $g(\boldsymbol{x}) = \sum_{j=1}^{q} g_j(\boldsymbol{x}_{H_j})$ *where* $g_j(\boldsymbol{x}_{H_j}) : \mathbb{R}^{|H_j|} \to \mathbb{R}$ *is a non-separable function that contains variables with indices in* $H_j \subseteq [n]$. *Let* $\texttt{D}$ *be the DD constructed via Algorithm 5 for some sub-domain partitions* $\mathcal{D}_i^j$ *with* $j \in L_i$ *for* $i \in [n]$, *width limit* $\omega$, *and merging oracle* $\texttt{Merge}(\omega, \texttt{V})$. *Then,* $\text{conv}(\mathcal{G}) \subseteq \text{conv}(\text{Sol}(\texttt{D}))$.

*Proof* Most of the proof steps are similar to those in Proposition 3.5, with an additional step to account for the impact of the merging operation on calculating the state values of each node. Throughout this proof, we reference the proof of Proposition 3.5 for the parts that follow from similar arguments provided therein. We show that $\mathcal{G} \subseteq \text{conv}(\text{Sol}(\texttt{D}))$. Pick $\bar{\boldsymbol{x}} \in \mathcal{G}$. It follows from the definition of $\mathcal{G}$ that $\sum_{k=1}^{q} g_k(\bar{\boldsymbol{x}}_{H_k}) \leq b$. For each $i \in [n]$, let $j_i^*$ be the index of a sub-domain partition $\mathcal{D}_i^{j_i^*}$ in $L_i$ such that $\bar{x}_i \in \mathcal{D}_i^{j_i^*}$. This index exists because $\bar{x}_i \in \mathcal{D}_i = \bigcup_{j \in L_i} \mathcal{D}_i^j$ where the inclusion follows from the fact that $\bar{\boldsymbol{x}} \in \mathcal{G}$, and the equality follows from the definition of sub-domain partitions.

Next, we show that $\texttt{D}$ includes a node sequence $\{\texttt{u}_1, \texttt{u}_2, \ldots, \texttt{u}_{n+1}\}$, where $\texttt{u}_i \in \texttt{U}_i$ for $i \in [n+1]$, such that each node $\texttt{u}_i$ is connected to $\texttt{u}_{i+1}$ via two arcs with labels $\mathcal{D}_i^{j_i^*}\!\downarrow$ and $\mathcal{D}_i^{j_i^*}\!\uparrow$ for each $i \in [n]$. Using an argument similar to that in the proof of Proposition 3.5 together with the instructions in lines 11–13 of Algorithm 5 and the fact that the incoming arcs of a node are preserved through

the merging operation, we conclude that a node sequence $\{\mathtt{u}_1, \mathtt{u}_2, \ldots, \mathtt{u}_n\}$ that satisfies the above conditions exists. We prove this sequence is completed by two arcs directed from node $\mathtt{u}_n$ to the terminal node $\mathtt{t}$ with labels $\mathcal{D}_n^{j_n^*}\downarrow$ and $\mathcal{D}_n^{j_n^*}\uparrow$. To this end, we need to show that the condition in line 15 of Algorithm 5 is satisfied.

To achieve this goal, we use induction on layer $i \in [n]$ to prove $\mathtt{s}(\mathtt{u}_i) \leq \sum_{k\in[q]:H_k^{\max}<i} g_k(\bar{\boldsymbol{x}}_{H_k})$. In words, we show that the state value of each node $\mathtt{u}_i$ in the previously picked sequence is no greater than the summation of non-separable function terms $g_k(\bar{\boldsymbol{x}}_{H_k})$ whose variable have already been visited in the previous layers of the DD. The induction base for $i = 1$ follows from line 1 of Algorithm 5 due to $\mathtt{s}(\mathtt{u}_1) = \mathtt{s}(\mathtt{r}) = 0$ and the fact that $H_k^{\max} > 0$ for all $k \in [q]$, implying that $\sum_{k\in[q]:H_k^{\max}<i} g_k(\bar{\boldsymbol{x}}_{H_k}) = 0$ by default. For the inductive hypothesis, assume that $\mathtt{s}(\mathtt{u}_{\hat{i}}) \leq \sum_{k\in[q]:H_k^{\max}<\hat{i}} g_k(\bar{\boldsymbol{x}}_{H_k})$ for $\hat{i} \in [n-1]$. For the inductive step, we show that $\mathtt{s}(\mathtt{u}_{\hat{i}+1}) \leq \sum_{k\in[q]:H_k^{\max}<\hat{i}+1} g_k(\bar{\boldsymbol{x}}_{H_k})$. To calculate $\mathtt{s}(\mathtt{u}_{\hat{i}+1})$, we consider two cases for $\mathtt{u}_{\hat{i}+1}$.

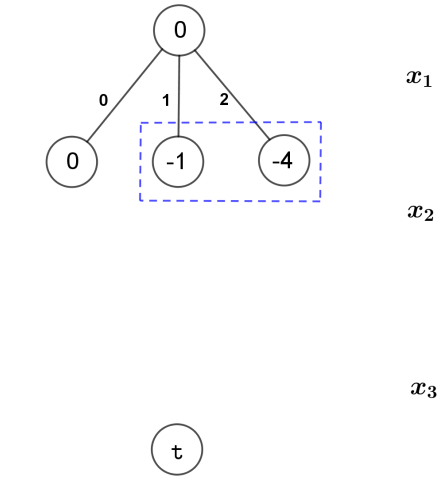
For the first case, assume that $\mathtt{u}_{\hat{i}+1}$ is not a merged node created through the merging oracle $\mathtt{Merge}(\omega, \mathtt{V})$. Therefore, this node must have been created through lines 11–13 of Algorithm 5. This case falls into the settings for Proposition 3.5. Consequently, we can use arguments similar to those in the proof of Proposition 3.5 to show that $\mathtt{s}(\mathtt{u}_{\hat{i}+1}) \leq \sum_{k\in[q]:H_k^{\max}<\hat{i}+1} g_k(\bar{\boldsymbol{x}}_{H_k})$.

For the second case, assume that $\mathtt{u}_{\hat{i}+1}$ is a merged node created through the merging oracle $\mathtt{Merge}(\omega, \mathtt{V})$. Therefore, before reaching line 18 of Algorithm 5, there must have been a node $\mathtt{v}$ in layer $\hat{i}+1$ created in lines 11–13 of the algorithm, with arcs connected to $\mathtt{u}_{\hat{i}}$ with label values $\mathcal{D}_{\hat{i}}^{j_{\hat{i}}^*}\downarrow$ and $\mathcal{D}_{\hat{i}}^{j_{\hat{i}}^*}\uparrow$, which is subsequently merged into node $\mathtt{u}_{\hat{i}+1}$ after executing the merging oracle $\mathtt{Merge}(\omega, \mathtt{V})$ in line 18 of the algorithm. As a result, we can use an argument similar to that of the first case above to show that $\mathtt{s}(\mathtt{v}) \leq \sum_{k\in[q]:H_k^{\max}<\hat{i}+1} g_k(\bar{\boldsymbol{x}}_{H_k})$. Since $\mathtt{v}$ is merged into $\mathtt{u}_{\hat{i}+1}$, it follows from the merging rule that $\mathtt{s}(\mathtt{u}_{\hat{i}+1}) \leq \mathtt{s}(\mathtt{v})$, yielding $\mathtt{s}(\mathtt{u}_{\hat{i}+1}) \leq \sum_{k\in[q]:H_k^{\max}<\hat{i}+1} g_k(\bar{\boldsymbol{x}}_{H_k})$.

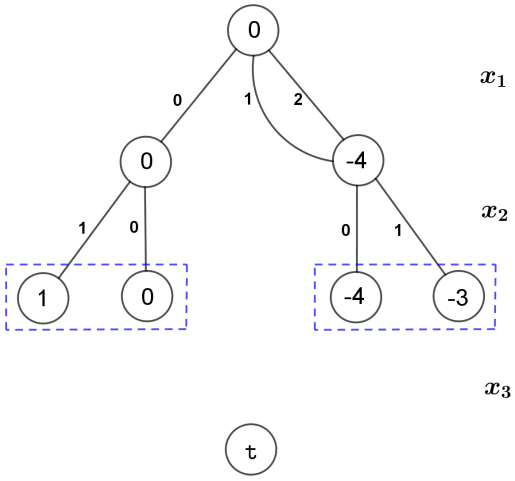
The remainder of the proof follows from arguments similar to those in the proof of Proposition 3.5.

□

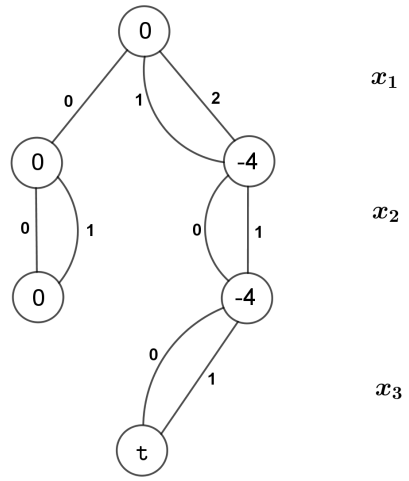
*Example 3.4* Consider the MINLP set $\mathcal{G}$ studied in Example 3.3. Assume that a width limit $\omega = 2$ is imposed. Since the DD constructed via Algorithm 4 in Figure 3.3 does not satisfy this width limit, we employ Algorithm 5 to apply a merging operation at layers whose size exceeds the width limit. The process of constructing the DD through merging nodes is shown in Figure 3.4. The first node layer contains the root node with state value 0. After the nodes in the second node layer are created through lines 3–14 of the algorithm, we observe that the number of nodes in this layer exceed the width limit, thereby calling the merging oracle $\mathtt{Merge}(\omega, \mathtt{U}_2)$ in line 18. According to this oracle, we merge nodes with state values $-1$ and $-4$, as shown in Figure 3.4a. The algorithm proceeds with creating the next layer which contains four nodes as shown in Figure 3.4b. Similarly to the previous layer, the merging oracle $\mathtt{Merge}(\omega, \mathtt{U}_3)$ is called to merge nodes with state values 1 and 0, as well as the nodes with state values $-4$ and $-3$ as depicted in dashed boxes in Figure 3.4b. Since the number of nodes in this layer satisfies the width limit, the algorithm continues to the last iteration to create the arcs that are connected to the terminal node. Through a calculation similar to that in Example 3.3, we conclude that the node with state value 0 in the node layer 3 cannot be connected to the terminal node as it does not satisfy the condition in line 15 of Algorithm 5. In contrast, the node with state value $-4$ is connected to the terminal node via two arcs with labels 0 and 1. The resulting DD, which we refer to as $\mathtt{D}^4$, is shown in Figure 3.4c. To reduce the size of the DD, we can remove the nodes that are not connected to the terminal node together with their incoming arcs to obtain a reduced DD shown in Figure 3.4d. It is easy to verify that $\text{conv}(\text{Sol}(\mathtt{D}^4)) = [0,2] \times [0,1] \times [0,1]$. This set matches the convex hull of the original set $\mathcal{G}$, i.e., $\text{conv}(\mathcal{G}) = \text{conv}(\text{Sol}(\mathtt{D}^4))$. ■



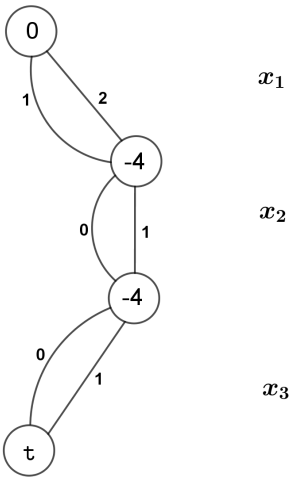

(a) Merging operation at node layer 2




(b) Merging operations at node layer 3




(c) Relaxed DD representation




(d) Reduced DD representation

Fig. 3.4: Relaxed DD for the set in Example 3.4.

Theorem 3.2 implies that applying a merging operation at the layers of a DD that models non-separable functions can effectively reduce the size of the DD while still providing a relaxation for the underlying set. This result is analogous to Theorem 3.1 for separable functions. However, there is an interesting and notable difference in how the merging operation impacts the relaxations produced for the separable and non-separable cases. In the separable case, the merging operation exhibits a *sequential relaxation* property. This means that the operation can be decomposed into a sequence of steps, with each step providing a relaxation for the set obtained from the previous step. This property is formalized in Proposition 3.3, which is used to establish Theorem 3.1. In contrast, the sequential relaxation property does not apply to the non-separable case. Specifically, merging nodes at the layers of a DD that uses node backtracking, as in Algorithm 4, while still providing a relaxation for the original set, does not necessarily provide a relaxation for the solution set of the DD itself.

This result may initially seem counterintuitive because merging operations are traditionally associated with weakening the relaxation, as they often lead to underestimating the state value of some nodes in the DD. However, when dealing with functions that include non-separable terms, the impact of merging operations extends beyond just the state values of individual nodes. Merging nodes at a given layer of a DD not only directly affects the state values of those nodes but also indirectly alters the structure of the DD by regrouping the arcs that can reach specific nodes in subsequent layers. This restructuring can significantly influence how state values are computed in later layers, potentially leading to stronger relaxations rather than weaker ones. As a result, the DD obtained after applying merging operations may not necessarily provide a relaxation of the original DD. This phenomenon can be illustrated by comparing the DDs from the separable and non-separable cases in Examples 3.1 and 3.2 for the former, and Examples 3.3 and 3.4 for the latter. For the separable case, as discussed in Examples 3.1 and 3.2, we have $\text{conv}(\mathtt{D}^1) \subset \text{conv}(\mathtt{D}^2)$, where $\mathtt{D}^2$ is obtained by applying the merging operation at some layers of $\mathtt{D}^1$. However, in the non-separable case discussed in Examples 3.3 and 3.4, the relationship between the DDs changes. In particular, we have $\text{conv}(\mathtt{D}^3) \nsubseteq \text{conv}(\mathtt{D}^4)$, even though $\mathtt{D}^3$ is derived by merges some nodes of $\mathtt{D}^4$. Instead, the opposite inclusion holds, i.e., $\text{conv}(\mathtt{D}^4) \subset \text{conv}(\mathtt{D}^3)$. This inversion occurs because the merging operation regrouped the arcs in layer 1 in a manner that strengthened the relaxation of the bilinear term $-x_1x_3$ in layer 3, leading to a tighter overall relaxation. This example highlights the nuanced effects of merging operations in non-separable DDs and demonstrates that, contrary to what one might expect, these operations can sometimes result in stronger relaxations rather than weaker ones.

The foundation of our DD solution framework lies in constructing outer approximations for the MINLP by convexifying the feasible regions defined by its individual constraints. However, a valuable feature that can enhance both the flexibility and strength of our method is the ability to convexify the feasible region defined by an intersection of multiple constraints simultaneously. The following remark outlines two approaches that can be employed to achieve this feature.

*Remark 3.1* Consider sets $\mathcal{G}^k$ defined in (2.2) for $k \in K$. The first approach for representing the intersection of multiple constraints via DDs involves the following steps: (i) Begin by constructing a separate DD $\mathtt{D}^k$ corresponding to each constraint $\mathcal{G}^k$ for $k \in K$ using the appropriate algorithms such as Algorithms 2–5, depending on whether the constraints are separable or non-separable; and (ii) intersect the resulting DDs using the well-known "conjoining" technique, which yields a DD $\bar{\mathtt{D}}$ whose feasible solutions consist of the solutions feasible to all individual DDs, i.e., $\text{Sol}(\bar{\mathtt{D}}) = \bigcap_{k\in K} \text{Sol}(\mathtt{D}^k)$; see [11] for a detailed exposition to the conjoining technique.

The second approach involves directly constructing a DD $\tilde{\mathtt{D}}$ that represents the feasible region defined by the intersected constraints, i.e., $\text{Sol}(\tilde{\mathtt{D}}) = \bigcap_{k\in K} \mathcal{G}^k$. The construction procedure follows a similar process to that outlined in Algorithms 2–5, with a key difference: for each node in the DD, instead of a single state value, there will be a vector of state values. Each component of this vector corresponds to the state values calculated according to the algorithms for each individual constraint $\mathcal{G}^k$. In the final step of these algorithms, a node at layer $n$ is connected to the terminal node if and only if the last if-condition is satisfied for all components of the state value vector.

### 3.4 Merging Policies

In Algorithms 3 and 5, the merging oracle $\texttt{Merge}(\omega, \mathtt{U})$ plays a critical role by providing a policy by merging subsets of nodes $\mathtt{U}$ in a DD layer to satisfy the prescribed DD width $\omega$. As noted in Example 3.4 and its subsequent discussions, this merging policy significantly impacts the quality of the relaxation represented by a DD. Various general-purpose and problem-specific merging policies can be developed to achieve specific properties for the relaxed DD set. In this section, we introduce two of the most effective and versatile merging oracles that can be applied to any MINLP structures.

The first merging oracle, denoted by $\texttt{Merge}^f(.)$, is described in Algorithm 6. This algorithm takes as inputs a width limit $\omega$ and a set of nodes $\texttt{U}_i = \{\texttt{u}_1, \texttt{u}_2, \ldots, \texttt{u}_\kappa\}$ in a DD layer $i > 1$, where $\kappa = |\texttt{U}_i|$. The merging policy used in this algorithm merges the $\kappa - \omega + 1$ nodes with the lowest state values to ensure that the width limit is satisfied. This approach aims to group nodes that are more likely to be part of feasible paths in the DD due to their minimal state value contribution at this layer. The following result shows that the time complexity of Algorithm 6 depends on the width limit and the number of sub-domain partitions, both of which can be controlled by the user.

**Proposition 3.6** *Consider a node set $\texttt{U}_i$ created in layer $i > 1$ of a DD $\texttt{D}$ through Algorithm 3 or Algorithm 5. Let $\omega$ be the width limit and $L_{i-1}$ be the index set of sub-domain partitions at layer $i-1$. Then, Algorithm 6 implements the merging policy defined by $\texttt{Merge}^f(\omega, \texttt{U}_i)$ with a time complexity of $\mathcal{O}(\tau \log(\tau))$, where $\tau = \omega |L_{i-1}|$.*

*Proof* The runtime complexity of Algorithm 6 is $\mathcal{O}(|\texttt{U}_i| \log(|\texttt{U}_i|) + |\mathcal{A}_{i-1}|)$ since $\mathcal{O}(|\texttt{U}_i| \log(|\texttt{U}_i|)$ is the complexity of sorting elements in $\texttt{U}_i$ in line 2 of the algorithm, and $|\mathcal{A}_{i-1}|$ is an upper bound for the number of incoming arcs that are updated in line 5 of the algorithm. Since the node set $\texttt{U}_i$ is created through Algorithm 3 or Algorithm 5, the number of nodes in this layer can be bounded by the prescribed width limit and the number of sub-domain partitions in that layer. In particular, we have $|\texttt{U}_i| \leq \omega |L_{i-1}|$. This is because of the for-loops in line 4 of Algorithms 3 and 5, which imply that for each node $\texttt{v} \in \texttt{U}_{i-1}$ and each sub-domain partition $j \in L_{i-1}$, one new node can be created in layer $i$ as shown in line 6 of Algorithms 3 and line 12 of Algorithm 5. Because the previous DD layers satisfy the width limit, we have that $|\texttt{U}_{i-1}| \leq \omega$. Similarly, the arcs in layer $\mathcal{A}_{i-1}$ are created within the same for-loops in line 7 of Algorithms 3 and line 13 of Algorithm 5, yielding $|\mathcal{A}_{i-1}| \leq 2\omega |L_{i-1}|$. Therefore, we obtain the time complexity $\mathcal{O}(\tau \log(\tau))$ for the algorithm. □

**Algorithm 6:** Merging oracle $\texttt{Merge}^f(.)$

**Data:** Set $\texttt{U}_i$ in a DD layer $i$, a width limit $\omega$
**Result:** A modified set $\texttt{U}_i$ that satisfies the width limit
1 **if** $|\texttt{U}_i| > \omega$ **then**
2  sort the node indices in $\texttt{U}_i$ based on their state values, i.e., $\{\texttt{u}_{j_1}, \texttt{u}_{j_2}, \ldots, \texttt{u}_{j_\kappa}\}$ such that $\texttt{s}(\texttt{u}_{j_1}) \leq \texttt{s}(\texttt{u}_{j_2}) \leq \ldots, \texttt{s}(\texttt{u}_{j_\kappa})$
3  create a node $\texttt{v}$ in $\texttt{U}_i$ and assign its state value $\texttt{s}(\texttt{v}) = \min\left\{\texttt{s}(\texttt{u}_{j_1}), \texttt{s}(\texttt{u}_{j_2}), \ldots, \texttt{s}(\texttt{u}_{j_{\kappa-\omega+1}})\right\}$
4  **forall** $k \in [\kappa - \omega + 1]$ **do**
5  disconnect the incoming arcs of $\texttt{u}_{j_k}$ from it and connect them to $\texttt{v}$
6  delete node $\texttt{u}_{j_k}$ from $\texttt{U}_i$

The second merging oracle, denoted by $\texttt{Merge}^g(.)$, is given in Algorithm 7. Similar to Algorithm 6, the inputs are the node sets $\texttt{U}_i$ and a width limit $\omega$. This merging oracle divides the entire range of state values of nodes in $\texttt{U}_i$ into $\omega$ sub-ranges, then merges all nodes within each sub-range. The rationale behind this approach is to group nodes with similar state values as merging candidates, thereby reducing the variation in new paths created by merging, hence yielding a tighter relaxation. The next result shows the time complexity of Algorithm 6. The proof is omitted as it follows from arguments similar to those presented in Proposition 3.6.

**Proposition 3.7** *Consider a node set $\texttt{U}_i$ created in layer $i > 1$ of a DD $\texttt{D}$ through Algorithm 3 or Algorithm 5. Let $\omega$ be the width limit and $L_{i-1}$ be the index set of sub-domain partitions at layer $i-1$. Then, Algorithm 7 implements the merging policy defined by $\texttt{Merge}^g(\omega, \texttt{U}_i)$ with a time complexity of $\mathcal{O}(\tau \log(\tau))$, where $\tau = \omega |L_{i-1}|$.*

**Algorithm 7:** Merging oracle $\texttt{Merge}^g(.)$

**Data:** Set $\texttt{U}_i$ in a DD layer $i$, a width limit $\omega$
**Result:** A modified set $\texttt{U}_i$ that satisfies the width limit

**if** $|\texttt{U}_i| > \omega$ **then**
  sort the node indices in $\texttt{U}_i$ based on their state values, i.e., $\texttt{U}_s = \{\texttt{u}_{j_1}, \texttt{u}_{j_2}, \ldots, \texttt{u}_{j_\kappa}\}$ such that $\texttt{s}(\texttt{u}_{j_1}) \leq \texttt{s}(\texttt{u}_{j_2}) \leq \ldots, \texttt{s}(\texttt{u}_{j_\kappa})$
  calculate the merging interval length $\gamma = \big(\texttt{s}(\texttt{u}_{j_\kappa}) - \texttt{s}(\texttt{u}_{j_1})\big)/\omega$
  **forall** $k \in [\omega]$ **do**
    create a node $\texttt{v}_k$ in $\texttt{U}_i$ and assign its state value $\texttt{s}(\texttt{v}) = \min_{l\in\pi}\{\texttt{s}(\texttt{u}_l)\}$, where $\pi$ is the set of indices of nodes in $\texttt{U}_i$ whose state value lie within $\big[\texttt{s}(\texttt{u}_{j_1}) + (k-1)\gamma,\ \texttt{s}(\texttt{u}_{j_1}) + k\gamma\big]$
    **forall** $l \in \pi$ **do**
      disconnect the incoming arcs of $\texttt{u}_l$ from it and connect them to $\texttt{v}_k$
      delete node $\texttt{u}_l$ from $\texttt{U}_i$

### 3.5 Lower Bound Calculation Rules

A key step in constructing DDs using Algorithms 2–5 is calculating a lower bound for constraint terms, which is used for determining the state values of each node. In this section, we discuss the methods that can be employed to calculate these bounds.

The first approach leverages *factorable decomposition*, a widely used technique for obtaining convex relaxations of MINLPs in existing global solvers. Consider a factorable function $g(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$, i.e., it can be decomposed into simpler terms with known convex hull representations over box domains. The goal is to find a lower bound $\eta$ for $g(\boldsymbol{x})$ over the box domain described by $x_i \in \mathcal{D}_i$ for all $i \in [n]$. Let $\mathcal{S}$ be the convex relaxation (possibly in a higher dimension) for the set $\big\{(\boldsymbol{x}, z) \in \prod_{i=1}^n \mathcal{D}_i \times \mathbb{R} \,\big|\, z = g(\boldsymbol{x})\big\}$ obtained by applying the factorable decomposition technique. Since $\mathcal{S}$ may be defined in a higher-dimensional space, we represent its solutions as a vector $(\boldsymbol{x}, z, \boldsymbol{y}) \in \mathbb{R}^{n+1+p}$, where components $\boldsymbol{y} \in \mathbb{R}^p$ are auxiliary variables introduced during the factorable decomposition process. The desired lower bound can then be computed as $\eta = \min\big\{z \big| (\boldsymbol{x}, z, \boldsymbol{y}) \in \mathcal{S}\big\}$. This results in a convex program that can be solved using various existing convex optimization methods.

Although convex programs can be solved in polynomial time, repeatedly invoking a convex solver for a large number of nodes in DD layers can be computationally intensive. Moreover, there are several applications, such as those discussed in Section 6, where the underlying nonlinear functions are not factorable, making them unsuitable for the factorable decomposition approach outlined above. Therefore, it is critical to develop an alternative method that can efficiently find lower bounds for a broader class of nonlinear functions when constructing large DDs. Our proposed method addresses this by leveraging the monotone properties of the underlying function, as described below. In the following definition, given a vector $\boldsymbol{x} \in \mathbb{R}^n$, we denote by $\boldsymbol{x}_{-i}$ a replica of the vector $\boldsymbol{x}$ with component $i \in [n]$ is removed.

**Definition 3.2** Consider a function $g(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ and a box domain described by $x_i \in \mathcal{D}_i$ for all $i \in [n]$. For each $i \in [n]$, we denote by $g(x_i, \bar{\boldsymbol{x}}_{-i}) : \mathbb{R} \to \mathbb{R}$ the univariate restriction of $g(\boldsymbol{x})$ in the space of $x_i$ where variables $x_j$ are fixed at $\bar{x}_j$ for all $j \in [n] \setminus \{i\}$. We say that $g(x_i, \bar{\boldsymbol{x}}_{-i})$ is *monotonically non-decreasing* (resp. *non-increasing*) over $\mathcal{D}_i$ if $g(\hat{\boldsymbol{x}}) \leq g(\tilde{\boldsymbol{x}})$ for any pair of points $\hat{\boldsymbol{x}}, \tilde{\boldsymbol{x}} \in \mathbb{R}^n$ such that $\hat{x}_j = \tilde{x}_j = \bar{x}_j$ for $j \in [n] \setminus \{i\}$, $\hat{x}_i, \tilde{x}_i \in \mathcal{D}_i$, and $\hat{x}_i \leq \tilde{x}_i$ (resp. $\hat{x}_i \geq \tilde{x}_i$). Further, we say that $g(\boldsymbol{x})$ is *monotone* over $\prod_{i=1}^n \mathcal{D}_i$ if its univariate restriction $g(x_i, \bar{\boldsymbol{x}}_{-i})$ is monotonically non-decreasing or non-increasing over $\mathcal{D}_i$ for each $i \in [n]$ and all fixed values $\bar{x}_j \in \mathcal{D}_j$ for $j \in [n] \setminus \{i\}$.

Next, we demonstrate how the monotone property of functions can be used to efficiently find a lower bound by evaluating the function at a specific point within its domain.

**Proposition 3.8** *Consider a monotone function $g(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ over a box domain described by $x_i \in \mathcal{D}_i$ for all $i \in [n]$. Then, the minimum value of $g(\boldsymbol{x})$ over the above box domain can be calculated as $\eta = g(\tilde{\boldsymbol{x}})$ where $\tilde{x}_i = \mathcal{D}_i\downarrow$ if $g(x_i, \bar{\boldsymbol{x}}_{-i})$ is monotonically non-decreasing, and $\tilde{x}_i = \mathcal{D}_i\uparrow$ if $g(x_i, \bar{\boldsymbol{x}}_{-i})$ is monotonically non-increasing, for each $i \in [n]$ and all fixed values $\bar{x}_j \in \mathcal{D}_j$ for $j \in [n] \setminus \{i\}$.*

*Proof* Assume by contradiction that $g(\tilde{\boldsymbol{x}})$ is not the minimum value of $g(\boldsymbol{x})$ over the box domain $\prod_{i=1}^n \mathcal{D}_i$. Then, there exists a point $\hat{\boldsymbol{x}} \in \prod_{i=1}^n \mathcal{D}_i$ such that $g(\hat{\boldsymbol{x}}) < g(\tilde{\boldsymbol{x}})$. Since $\hat{\boldsymbol{x}} \neq \tilde{\boldsymbol{x}}$, there must exist indexes $i_1, i_2, \ldots, i_p$ for some $1 \leq p \leq n$ such that $\hat{x}_{i_j} \neq \tilde{x}_{i_j}$ for each $j \in [p]$. For each such $j$, construct $\hat{\boldsymbol{x}}^j$ to be the point where $\hat{x}^j_k = \hat{x}^{j-1}_k$ for $k \neq i_j$, and $\hat{x}^j_{i_j} = \mathcal{D}_{i_j}\downarrow$ if $g(x_{i_j}, \hat{\boldsymbol{x}}_{-i_j})$ is monotonically non-decreasing over $\mathcal{D}_{i_j}$, and $\hat{x}^j_{i_j} = \mathcal{D}_{i_j}\uparrow$ if $g(x_{i_j}, \hat{\boldsymbol{x}}_{-i_j})$ is monotonically non-increasing over $\mathcal{D}_{i_j}$. In this definition, we set $\hat{\boldsymbol{x}}^0 = \hat{\boldsymbol{x}}$. Further, the description of $\tilde{\boldsymbol{x}}$ in the proposition statement implies that $\hat{\boldsymbol{x}}^p = \tilde{\boldsymbol{x}}$. It follows from Definition 3.2 that $g(\hat{\boldsymbol{x}}) = g(\hat{\boldsymbol{x}}^0) \geq g(\hat{\boldsymbol{x}}^1) \geq \ldots \geq g(\hat{\boldsymbol{x}}^p) = g(\tilde{\boldsymbol{x}})$. This is a contradiction to the initial assumption that $g(\hat{\boldsymbol{x}}) < g(\tilde{\boldsymbol{x}})$. □

In practice, many nonlinear functions satisfy the monotone property as defined in Definition 3.2, enabling the use of a fast method, as outlined in Proposition 3.8, to calculate lower bounds when constructing DDs for those functions. For example, consider a general polynomial function commonly used in MINLP models, defined as $g(\boldsymbol{x}) = \prod_{i=1}^n x_i^{\alpha_i}$ with $\alpha_i \in \mathbb{R}$ for $i \in [n]$. It is easy to verify that $g(\boldsymbol{x})$ is monotone over each orthant. As another advantage, the monotone property facilitates the calculation of lower bounds for non-factorable functions that are not suitable for factorable decomposition, as demonstrated in the following example.

*Example 3.5* Consider the $\ell_p$-norm function $g(\boldsymbol{x}) = ||\boldsymbol{x}||_p = \left(\sum_{i=1}^n x_i^p\right)^{1/p}$, for $p \in (0, \infty)$. This function can be convexified using factorable decomposition method over a box domain $\prod_{i=1}^n \mathcal{D}_i$ in the positive orthant. Let $\mathcal{S}$ be the convex relaxation described by convexifying individual constraints of the following decomposed formulation of the model

$$
\begin{aligned}
& z = y_0^{1/p} && \\
& y_0 = \sum_{i=1}^n y_i && \\
& y_i = x_i^p && \forall i \in [n] \\
& x_i \in \mathcal{D}_i && \forall i \in [n].
\end{aligned}
$$

According to the previous arguments, minimizing $z$ over $\mathcal{S}$ provides a lower bound for $g(\boldsymbol{x})$ over its box domain. Alternatively, we can use Proposition 3.8 to calculate this lower bound because $g(\boldsymbol{x})$ is monotone over its imposed domain. An advantage of the latter approach is that it can be executed directly in the space of the original variables, eliminating the need to introduce auxiliary variables and additional constraints to handle decoupled terms, as required by the factorable decomposition approach.

Now consider the $\ell_0$-norm function $h(\boldsymbol{x}) = ||\boldsymbol{x}||_0 = \sum_{i=1}^n \mathbb{I}(x_i)$ over the above box domain $\prod_{i=1}^n \mathcal{D}_i$, where $\mathbb{I}(x_i) = 0$ if $x_i = 0$, and $\mathbb{I}(x_i) = 1$ otherwise. This function is not factorable, making it not unsuitable for application of factorable decomposition. In contrast, it is easy to verify that $h(\boldsymbol{x})$ is monotone over its box domain. Therefore, we can use Proposition 3.8 to calculate its lower bound. ■

Despite the wide range of functions that satisfy the monotone property, some functions do not exhibit this property due to the presence of variables in multiple positions, breaking the monotonic

patterns of their univariate restrictions. For instance, consider the function $g(x_1, x_2) = \frac{x_2^4\, e^{-x_2}}{\arctan(x_1)+1}$ over the positive orthant. It is clear that the univariate restriction of $g(x_1, x_2)$ in the space of $x_1$, i.e., $g(x_1, \bar{x}_{-1}) = \frac{\bar{x}_2^4\, e^{-\bar{x}_2}}{\arctan(x_1)+1}$, is monotonically non-increasing over this domain. However, the univariate restriction of $g(x_1, x_2)$ in the space of $x_2$, i.e., $g(x_2, \bar{x}_{-2}) = \frac{x_2^4\, e^{-x_2}}{\arctan(\bar{x}_1)+1}$, is not monotone. Consequently, the method of Proposition 3.8 cannot be used to find a lower bound for $g(x_1, x_2)$. To address such function structures, we employ a technique referred to as *re-indexing*, which is outlined next.

**Definition 3.3** Consider a function $g(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$. Define the *re-indexed* function $g^{\text{rx}}(\boldsymbol{y})$ of $g(\boldsymbol{x})$ by substituting the variables $\boldsymbol{x}$ with variables $\boldsymbol{y}$ such that each $y_i$ variable appears only once in the function's expression. If variable $y_j$ substitutes variable $x_i$ in the re-indexed function, we denote the relation between these indices by the mapping $R(j) = i$.

In the example discussed previously, the re-indexed function of $g(x_1, x_2)$ is $g^{\text{rx}}(y_1, y_2, y_3) = \frac{y_1^4\, e^{-y_2}}{\arctan(y_3)+1}$, where $R(1) = 2$, $R(2) = 2$, and $R(3) = 1$. It is easy to verify that $g^{\text{rx}}(y_1, y_2, y_3)$ is monotone over the positive orthant, allowing the application of Proposition 3.8 to calculate its lower bound. The next proposition shows that this lower bound can also serve as a lower bound for the original function $g(x_1, x_2)$.

**Proposition 3.9** *Consider a function $g(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ over a box domain described by $x_i \in \mathcal{D}_i$ for all $i \in [n]$. Let $g^{rx}(\boldsymbol{y}) : \mathbb{R}^p \to \mathbb{R}$ be the re-indexed function of $g(\boldsymbol{x})$ with re-index mapping $R(.)$. Assume that $g^{rx}(\boldsymbol{y})$ is monotone over the box domain described by $y_j \in \mathcal{D}_{R(j)}$ for all $j \in [p]$. Then, a lower bound for $g(\boldsymbol{x})$ over the above box domain can be calculated as $\eta = g^{rx}(\tilde{\boldsymbol{y}})$ where $\tilde{y}_j = \mathcal{D}_{R(j)}\downarrow$ if $g^{rx}(y_j, \bar{\boldsymbol{y}}_{-j})$ is monotonically non-decreasing, and $\tilde{y}_j = \mathcal{D}_{R(j)}\uparrow$ if $g^{rx}(y_j, \bar{\boldsymbol{y}}_{-j})$ is monotonically non-increasing, for each $j \in [p]$ and all fixed values $\bar{y}_k \in \mathcal{D}_{R(k)}$ for $k \in [p] \setminus \{j\}$.*

*Proof* Since $g^{\text{rx}}(\boldsymbol{y})$ is monotone over the box domain described by $y_j \in \mathcal{D}_{R(j)}$ for all $j \in [p]$, Proposition 3.8 implies that $\eta$ is the minimum of $g^{\text{rx}}(\boldsymbol{y})$ over this box domain, i.e., $\eta = \min\left\{g^{\text{rx}}(\boldsymbol{y}) \,\middle|\, \boldsymbol{y} \in \prod_{j=1}^{p} \mathcal{D}_{R(j)}\right\}$. Now consider point $\boldsymbol{x}^* \in \prod_{i=1}^{n} \mathcal{D}_i$ that achieves the minimum of $g(\boldsymbol{x})$ over the box domain $\prod_{i=1}^{n} \mathcal{D}_i$, i.e., $g(\boldsymbol{x}^*) \leq g(\boldsymbol{x})$ for all $\boldsymbol{x} \in \prod_{i=1}^{n} \mathcal{D}_i$. Construct the point $\boldsymbol{y}^* \in \mathbb{R}^n$ such that $y_j^* = x^*_{R(j)}$ for all $j \in [p]$. Therefore, for each $\boldsymbol{x} \in \prod_{i=1}^{n} \mathcal{D}_i$, we can write that $g(\boldsymbol{x}) \geq g(\boldsymbol{x}^*) = g^{\text{rx}}(\boldsymbol{y}^*) \geq \eta$, where the first inequality holds because of the previous argument, the first equality follows from the definition of $g^{\text{rx}}(\boldsymbol{y})$, and the second inequality is due to the first argument in the proof. As a result, $\eta$ is a lower bound for $g(\boldsymbol{x})$ over the above box domain $\prod_{i=1}^{n} \mathcal{D}_i$.

□

From a practical standpoint, the combination of exploiting the monotone property of functions as outlined in Proposition 3.8, the re-indexing technique described in Proposition 3.9, and the DD intersection method discussed in Remark 3.1 offers a unique and powerful modeling tool for constructing relaxed DDs across a wide range of MINLP structures. In fact, our observations suggest that these techniques can handle most structures in the MINLP library, including the most complex types that remain unsolved due to their inadmissibility by state-of-the-art global solvers; see the computational studies in Section 6.

### 3.6 Time Complexity of Algorithms

In this section, we analyze the time complexity of Algorithms 3 and 5, incorporating the merging policies outlined in Section 3.4 and lower bound calculation rules discussed in Section 3.5. Specifically, for all these algorithms, we assume that the lower bounds on the functions, whether separable or non-separable, are calculated using the monotone property and the re-indexing technique described in Section 3.5. Proposition 3.10 presents the time complexity results for the algorithm that constructs a DD corresponding to constraints with separable functions, as developed in Section 3.2.

**Proposition 3.10** *Consider set* $\mathcal{G} = \{\boldsymbol{x} \in \prod_{i=1}^{n} \mathcal{D}_i \,|\, g(\boldsymbol{x}) \leq b\}$ *where* $g(\boldsymbol{x}) = \sum_{i=1}^{n} g_i(x_i)$ *is separable, the sub-domain partitions* $\mathcal{D}_i^j$ *with* $j \in L_i$ *for* $i \in [n]$*, a width limit* $\omega$*, and a merging oracle* $\texttt{Merge}(\omega, \texttt{V})$ *described in Algorithm 6 or 7. Then, Algorithm 3 constructs a DD* $\texttt{D} = (\texttt{U}, \texttt{A}, \texttt{l}(.))$ *corresponding to* $\mathcal{G}$ *in time* $\mathcal{O}(\sum_{i=1}^{n} \tau_i \log(\tau_i))$*, where* $\tau_i = \omega|L_i|$ *for* $i \in [n]$*.*

*Proof* It follows from the for-loop in line 4 of Algorithms 3 that at each layer $i \in [n]$, the arcs in that layer and the nodes in the next layer are computed in $\mathcal{O}(|\texttt{U}_i||L_i|)$. However, the steps of the algorithm ensure that $|\texttt{U}_i|$ does not exceed the width limit $\omega$, yielding the time complexity $\mathcal{O}(\tau_i)$ for the above operations. On the other hand, Propositions 3.6 and 3.7 imply that the merging operation in line 8 of Algorithms 3 can be performed in $\mathcal{O}(\tau_i \log(\tau_i))$ for each $i \in [n]$. Considering all layers, we obtain the total time complexity of $\mathcal{O}(\sum_{i=1}^{n} \tau_i \log(\tau_i))$.

□

Proposition 3.11 provides the time complexity results for the algorithm that builds a DD corresponding to constraints with non-separable functions, as developed in Section 3.3.

**Proposition 3.11** *Consider set* $\mathcal{G} = \{\boldsymbol{x} \in \prod_{i=1}^{n} \mathcal{D}_i \,|\, g(\boldsymbol{x}) \leq b\}$ *where* $g(\boldsymbol{x}) = \sum_{k=1}^{q} g_k(\boldsymbol{x}_{H_k})$ *such that* $g_k(\boldsymbol{x}_{H_k}) : \mathbb{R}^{|H_k|} \to \mathbb{R}$ *is a non-separable function that contains variables with indices in* $H_k$*. Consider the sub-domain partitions* $\mathcal{D}_i^j$ *with* $j \in L_i$ *for* $i \in [n]$*, a width limit* $\omega$*, and a merging oracle* $\texttt{Merge}(\omega, \texttt{V})$ *described in Algorithm 6 or 7. Then, Algorithm 5 constructs a DD* $\texttt{D} = (\texttt{U}, \texttt{A}, \texttt{l}(.))$ *corresponding to* $\mathcal{G}$ *in time* $\mathcal{O}\big(\sum_{i=1}^{n} \tau_i \log(\tau_i) + \sum_{k=1}^{q} \theta_k\big)$*, where* $\tau_i = \omega|L_i|$*, and* $\theta_k = \omega|H_k| \sum_{l=H_k^{min}}^{H_k^{max}-1} |L_l|$ *with* $H_k^{max} = \max_{l \in H_k}\{l\}$ *and* $H_k^{min} = \min_{l \in H_k}\{l\}$*.*

*Proof* It follows from the for-loops in lines 3–5 of Algorithm 5 that at each layer $i \in [n]$, the lower bound calculation in lines 6–10 of the algorithm, as well as the node and arc creation in lines 12–16 of the algorithm can be performed in $\mathcal{O}(|\texttt{U}|_i|L_i|)$. Considering that $|\texttt{U}_i|$ is bounded by the width limit $\omega$, we obtain the time complexity of $\mathcal{O}(\omega|L_i|)$ for the above operations. Further, it follows from an argument similar to that in the proof of Proposition 3.10 that the merging operation in line 18 of Algorithm 5 can be performed in $\mathcal{O}\big(\omega|L_i|\log(\omega|L_i|)\big) = \mathcal{O}(\tau_i \log(\tau_i))$ for each $i \in [n]$. Considering all layers, we obtain the total time complexity for these tasks to be $\mathcal{O}(\sum_{i=1}^{n} \tau_i \log(\tau_i))$.

Next, we obtain the time complexity for calculating the relative sub-domains in line 14 of the algorithm. Corollary 3.1 implies that the sub-domain $\mathcal{D}_j(\texttt{v})$ relative to variable $x_j$ for all nodes $\texttt{v}$ in layer $i \in [n]$ can be computed in $\mathcal{O}(\sum_{l=j}^{i-1} |\texttt{A}_l|)$. Using a similar argument to that given previously, we can bound the above term by $\mathcal{O}(\omega \sum_{l=j}^{i-1} |L_l|)$. These values are calculated for each layer $i = H_k^{\max}$ for all $k \in [q]$. Furthermore, for the nodes $\texttt{v}$ in layer $i = H_k^{\max}$ for each $k \in [q]$, we need to calculate the sub-domain $\mathcal{D}_j(\texttt{v})$ for all $j \in H_k \setminus \{i\}$. Therefore, for a given $k \in [q]$, the time complexity for calculating the sub-domains relative to all $x_j$ with $j \in H_k \setminus \{H_k^{\max}\}$ can be bounded by $\mathcal{O}\big(\omega|H_k| \sum_{l=H_k^{\min}}^{H_k^{\max}-1} |L_l|\big) = \mathcal{O}(\theta_k)$. This yields the total time complexity for calculating the relative sub-domains in line 14 of the algorithm to be bounded by $\mathcal{O}\big(\sum_{k=1}^{q} \theta_k\big)$.

□

## 4 Outer Approximation

In this section, we describe the oracle `Outer_Approx` in Algorithm 1. This oracle produces a linear outer approximation for the solutions of the DD constructed by `Construct_DD` to find dual bounds. Recently, [22, 23] proposed efficient methods to obtain a convex hull description for the solution set of DDs in the original space of variables through a successive generation of cutting planes. In this section, we present a summary of those methods, adapted for the DDs constructed in Section 3; refer to the references above for detailed derivations. We begin by describing the convex hull in an extended space of variables.

**Proposition 4.1** *Consider a DD* $\mathtt{D} = (\mathtt{U}, \mathtt{A}, \mathtt{l}(.))$ *with solution set* $\mathrm{Sol}(\mathtt{D}) \subseteq \mathbb{R}^n$. *Define* $\mathcal{P} = \left\{(\boldsymbol{x}; \boldsymbol{y}) \in \mathbb{R}^n \times \mathbb{R}^{|\mathtt{A}|} \middle| (4.1a), (4.1b)\right\}$ *where*

$$\sum_{\mathtt{a} \in \delta^+(\mathtt{u})} y_{\mathtt{a}} - \sum_{\mathtt{a} \in \delta^-(\mathtt{u})} y_{\mathtt{a}} = f_{\mathtt{u}}, \qquad \forall \mathtt{u} \in \mathtt{U} \tag{4.1a}$$

$$\sum_{\mathtt{a} \in \mathtt{A}_i} \mathtt{l}(\mathtt{a})\, y_{\mathtt{a}} = x_i, \qquad \forall i \in [n] \tag{4.1b}$$

$$y_{\mathtt{a}} \geq 0, \qquad \forall \mathtt{u} \in \mathtt{U}, \tag{4.1c}$$

*where* $f_{\mathtt{r}} = -f_{\mathtt{t}} = 1$, $f_{\mathtt{u}} = 0$ *for* $\mathtt{u} \in \mathtt{U} \setminus \{\mathtt{r}, \mathtt{t}\}$, *and* $\delta^+(\mathtt{u})$ *(resp.* $\delta^-(\mathtt{u})$*) denotes the set of outgoing (resp. incoming) arcs at node* $\mathtt{u}$. *Then,* $\mathrm{proj}_x \mathcal{P} = \mathrm{conv}(\mathrm{Sol}(\mathtt{D}))$. □

Viewing $y_{\mathtt{a}}$ as the network flow variable on arc $\mathtt{a} \in \mathtt{A}$ of $\mathtt{D}$, the formulation (4.1a)–(4.1c) implies that the LP relaxation of the network model that routes one unit of supply from the root node to the terminal node of the DD provides a convex hull description for the solution set of $\mathtt{D}$ in a higher dimension. Thus, projecting out the arc-flow variables $\boldsymbol{y}$ from this formulation would yield $\mathrm{conv}(\mathrm{Sol}(\mathtt{D}))$ in the original space of variables. This result leads to a separation oracle that can be used to separate any point $\bar{\boldsymbol{x}} \in \mathbb{R}^n$ from $\mathrm{conv}(\mathrm{Sol}(\mathtt{D}))$ through solving the cut-generating LP given in Proposition 4.2 below. In this model, $\boldsymbol{\theta} \in \mathbb{R}^{|\mathtt{U}|}$ and $\boldsymbol{\gamma} \in \mathbb{R}^n$ are dual variables associated with constraints (4.1a) and (4.1b), respectively.

**Proposition 4.2** *Consider a DD* $\mathtt{D} = (\mathtt{U}, \mathtt{A}, \mathtt{l}(.))$ *with solution set* $\mathrm{Sol}(\mathtt{D}) \subseteq \mathbb{R}^n$. *Consider a point* $\bar{\boldsymbol{x}} \in \mathbb{R}^n$, *and define*

$$\omega^* = \max \quad \sum_{i \in [n]} \bar{x}_i \gamma_i - \theta_{\mathtt{t}} \tag{4.2}$$

$$\theta_{\mathtt{t}(\mathtt{a})} - \theta_{\mathtt{h}(\mathtt{a})} + \mathtt{l}(\mathtt{a})\gamma_i \leq 0, \qquad \forall i \in [n], \mathtt{a} \in \mathtt{A}_k \tag{4.3}$$

$$\theta_{\mathtt{r}} = 0. \tag{4.4}$$

*Then,* $\bar{\boldsymbol{x}} \in \mathrm{conv}(\mathrm{Sol}(\mathtt{D}))$ *if* $\omega^* = 0$. *Otherwise,* $\bar{\boldsymbol{x}}$ *can be separated from* $\mathrm{conv}(\mathrm{Sol}(\mathtt{D}))$ *via* $\sum_{i \in [n]} x_i \gamma_i^* \leq \theta_{\mathtt{t}}^*$ *where* $(\boldsymbol{\theta}^*; \boldsymbol{\gamma}^*)$ *is an optimal recession ray of* (4.2)–(4.4). □

The above separation oracle requires solving an LP whose size is proportional to the number of nodes and arcs of the DD, which could be computationally intensive when used repeatedly inside an outer approximation framework. As a result, an alternative subgradient-type method is proposed to solve the same separation problem, but with a focus on detecting a violated cut faster.

In Algorithm 8, *Terminate_Flag* contains criteria to stop the loop, such as iteration number, elapsed time, objective function improvement tolerance, among others. We summarize the recursive step of the separation method employed in this algorithm as follows. The vector $\boldsymbol{\gamma}^\tau \in \mathbb{R}^n$ is used in line 3 to assign weights to the arcs of the DD, which are then used to obtain the longest `r-t` path. The solution $\boldsymbol{x}^\tau$ corresponding to this longest path is subtracted from the separation point $\bar{\boldsymbol{x}}$, yielding the subgradient value for the objective function of the separation problem at the point $\boldsymbol{\gamma}^\tau$; see Proposition 3.5 in [23]. The subgradient direction is then updated in line 7 for a step size $\rho_\tau$, and subsequently projected onto the unit sphere of the variables $\boldsymbol{\gamma}$ in line 8. It is shown in [23] that for an appropriate step size, this algorithm converges to an optimal recession ray of the separation problem (4.2)–(4.4), thereby producing the desired cutting plane in line 11. This algorithm is derivative-free, as it computes subgradient values by solving a longest path problem over a weighted DD. Consequently, it is highly effective in identifying violated cutting planes compared to the LP (4.2)–(4.4), making it well-suited for implementation within the spatial branch-and-cut framework used in Algorithm 1.

**Algorithm 8:** A subgradient-type separation algorithm

**Data:** A DD $\mathtt{D} = (\mathtt{U}, \mathtt{A}, \mathtt{l}(.))$ and a point $\bar{\boldsymbol{x}}$
**Result:** A valid inequality to separate $\bar{\boldsymbol{x}}$ from conv(Sol($\mathtt{D}$))
initialize $\tau = 0$, $\boldsymbol{\gamma}^0 \in \mathbb{R}^n$, $\tau^* = 0$, $\Delta^* = 0$
**while** *Terminate_Flag* = *False* **do**
  assing weights $\mathtt{w}(\mathtt{a}) = \mathtt{l}(\mathtt{a})\gamma_i^\tau$ to each arc $\mathtt{a} \in \mathtt{A}_i$ of $\mathtt{D}$ for all $i \in [n]$
  find a longest $\mathtt{r}$-$\mathtt{t}$ path in the weighted DD and compute its encoding point $\boldsymbol{x}^\tau$
  **if** $\boldsymbol{\gamma}^\tau(\bar{\boldsymbol{x}} - \boldsymbol{x}^\tau) > \max\{0, \Delta^*\}$ **then**
    update $\tau^* = \tau$ and $\Delta^* = \boldsymbol{\gamma}^\tau(\bar{\boldsymbol{x}} - \boldsymbol{x}^\tau)$
  update $\boldsymbol{\phi}^{\tau+1} = \boldsymbol{\gamma}^\tau + \rho_\tau(\bar{\boldsymbol{x}} - \boldsymbol{x}^\tau)$ for step size $\rho_\tau$
  find the projection $\boldsymbol{\gamma}^{\tau+1}$ of $\boldsymbol{\phi}^{\tau+1}$ onto the unit sphere defined by $||\boldsymbol{\gamma}||_2 \leq 1$
  set $\tau = \tau + 1$
**if** $\Delta^* > 0$ **then**
  return inequality $\boldsymbol{\gamma}^{\tau^*}(\boldsymbol{x} - \boldsymbol{x}^{\tau^*}) \leq 0$

The cutting planes obtained from the separation methods in Proposition 4.2 and Algorithm 8 can be incorporated into `Outer_Approx` as follows. In the recursive steps of Algorithm 1, the LP relaxation $LP$ at a node of the B&B tree is solved to obtain an optimal solution $\boldsymbol{x}^*$, if one exists. For each constraint $k \in K$ in the MINLP (2.1a)–(2.1c), the solution $\boldsymbol{x}^*$ is evaluated to identify any violated constraints. For each violated constraint, the aforementioned separation methods are employed to generate a cutting plane that separates $\boldsymbol{x}^*$ from conv(Sol($\mathtt{D}^k$)), where $\mathtt{D}^k$ is the DD constructed for set $\mathcal{G}^k$ in line 7 of Algorithm 1. The resulting cutting plane is then added to the LP relaxation, and the process is repeated until no new cuts are introduced or a stopping criterion, such as a maximum number of iterations or gap tolerance, is met. Subsequently, the bounds are updated, and if the current node is not pruned, a spatial branch-and-bound scheme is applied, as discussed in the following section.

## 5 Spatial Branch-and-Bound

In global optimization of MINLPs, a divide-and-conquer strategy, such as spatial branch-and-bound (SB&B), is employed to achieve convergence to a global optimal solution of the problem. The SB&B strategy reduces the domain of the variables by successively partitioning their original box domains. These partitions are typically rectangular, dividing the variable domain into smaller hyper-rectangles as a result of branching. For each such partition, a convex relaxation is constructed to calculate a dual bound. As the process advances, tighter relaxations are obtained, leading to updated dual bounds, which continue to improve until they approach the global optimal value of the problem within a specified tolerance. To establish convergence, it must be shown that the convexification method applied at each partition converges (in the Hausdorff sense) to the convex hull of the feasible region restricted to that partition; refer to [7, 50, 62] for a detailed discussion on SB&B methods for MINLPs.

In this section, we discuss the convergence results for the SB&B procedure employed in Algorithm 1. After solving a linear outer approximation of the MINLP at the current node of the SB&B tree and updating the bounds, the `Branch` oracle in line 13 of the algorithm is invoked to perform the branching operation, provided the node is not pruned. This operation creates two child nodes by partitioning the domain of the selected branching variable based on the branching value. In the sequel, we show that the convex hull of the solution set of the DDs obtained from Algorithms 2–5 converges to the convex hull of the solutions of the original set $\mathtt{G}$ as the partition volume decreases.

We establish these results for DDs of unit width as a DD with a larger width can be decomposed into a union of finitely many unit-width DDs while preserving the convex hull; see the discussions in Section 3. Throughout this section, we assume that the domain partitioning performed through SB&B takes into account the integrality requirements for integer variables. For instance, if an integer variable $x$ within the domain $[l, u]$, where $l, u \in \mathbb{Z}$ and $l < u$, is selected for branching at a value $w \in [l, u]$, the new domain partitions will be $[l, \lfloor w \rfloor]$ and $[\lfloor w \rfloor + 1, u]$.

First, we prove that reducing the variables' domain through SB&B partitioning leads to tighter convex relaxations obtained by the proposed DD-based convexification method described in Sections 3 and 4. Propositions 5.1 and 5.2 establish this result for the separable and non-separable case, respectively. To prove this, we rely on a key property of the lower bound calculation rules used in the SB&B process, which we define next.

**Definition 5.1** Consider a function $g(\boldsymbol{x}) : \mathcal{D} \to \mathbb{R}$, where $C \subseteq [n]$ and $I = [n] \setminus C$ represent the index sets of continuous and integer variables, respectively, and where $\mathcal{D} = \prod_{i=1}^{n} \mathcal{D}_i$ with $\mathcal{D}_i = [\mathcal{D}_i\downarrow, \mathcal{D}_i\uparrow]$ for $i \in C$ and $\mathcal{D}_i = [\mathcal{D}_i\downarrow, \mathcal{D}_i\uparrow] \cap \mathbb{Z}$ for $i \in I$. Consider a lower bound calculation rule that outputs a lower bound $\eta(\bar{\mathcal{D}})$ for $g(\boldsymbol{x})$ over a box domain $\bar{\mathcal{D}} \subseteq \mathcal{D}$. We say that this lower bound calculation rule is *consistent with respect to* $g(\boldsymbol{x})$ *over* $\mathcal{D}$ if $\eta(\mathcal{D}^1) \geq \eta(\mathcal{D}^2)$ for any $\mathcal{D}^1 \subseteq \mathcal{D}^2 \subseteq \mathcal{D}$.

**Proposition 5.1** *Consider a separable function $g(\boldsymbol{x}) : \mathcal{P} \to \mathbb{R}$, where $C \subseteq [n]$ and $I = [n] \setminus C$ represent the index sets of continuous and integer variable, respectively, and where $\mathcal{P} = \prod_{i=1}^{n} \mathcal{P}_i$ with $\mathcal{P}_i = [\mathcal{P}_i\downarrow, \mathcal{P}_i\uparrow]$ for $i \in C$ and $\mathcal{P}_i = [\mathcal{P}_i\downarrow, \mathcal{P}_i\uparrow] \cap \mathbb{Z}$ for $i \in I$. Define $\mathcal{F}^j = \{\boldsymbol{x} \in \mathcal{P}^j \mid g(\boldsymbol{x}) \leq b\}$ for $j = 1, 2$, where $b \in \mathbb{R}$, and $\mathcal{P}^j = \prod_{i=1}^{n} \mathcal{P}_i^j \subseteq \mathcal{P}$. For each $j = 1, 2$, let $\mathtt{D}^j$ be the DD constructed via Algorithm 2 or 3 for a single sub-domain partition $\mathcal{P}_i^j$ of variable $x_i$ for each $i \in [n]$ using a lower bound calculation rule that is consistent with respect to $g_i(x_i)$ over $\mathcal{P}_i$. If $\mathcal{P}^2 \subseteq \mathcal{P}^1$, then* $\text{conv}(\text{Sol}(\mathtt{D}^2)) \subseteq \text{conv}(\text{Sol}(\mathtt{D}^1))$.

*Proof* Since there is only one sub-domain partition for each variable, the DDs constructed via Algorithm 2 and 3 are the same. Thus, we show the result assuming Algorithm 2 is used. According to this algorithm, because there is only one sub-domain partition $\mathcal{P}_i^2$ for the domain of variable $x_i$ for $i \in [n]$, each node layer of $\mathtt{D}^2$ contains one node, referred to as $\mathtt{u}_i$. Following the top-down construction steps of Algorithm 2, for each $i \in [n-1]$, the node $\mathtt{u}_i$ is connected via two arcs with label values $\mathcal{P}_i^2\downarrow$ and $\mathcal{P}_i^2\uparrow$ to the node $\mathtt{u}_{i+1}$. There are two cases for the arcs at layer $i = n$ based on the value of $\xi^*$ calculated in line 8 of this algorithms.

For the first case, assume that $\xi^* > b$. Then, the if-condition in line 9 of Algorithm 2 is not satisfied. Therefore, the node $\mathtt{u}_n$ is not connected to the terminal node $\mathtt{t}$ of $\mathtt{D}^2$. As a result, there is no $\mathtt{r}$-$\mathtt{t}$ path in this DD, leading to an empty solution set, i.e., $\text{conv}(\text{Sol}(\mathtt{D}^2)) = \text{Sol}(\mathtt{D}^2) = \emptyset$. This proves the result since $\emptyset \subseteq \text{conv}(\text{Sol}(\mathtt{D}^1))$.

For the second case, assume that $\xi^* \leq b$. Then, the if-condition in line 9 of Algorithm 2 is satisfied, and node $\mathtt{u}_n$ is connected to the terminal node $\mathtt{t}$ of $\mathtt{D}^2$ via two arcs with label values $\mathcal{P}_n^2\downarrow$ and $\mathcal{P}_n^2\uparrow$. Therefore, the solution set of $\mathtt{D}^2$ contains $2^n$ points encoded by all the $\mathtt{r}$-$\mathtt{t}$ paths of the DD, each composed of arcs with label values $\mathcal{P}_i^2\downarrow$ or $\mathcal{P}_i^2\uparrow$ for $i \in [n]$. It is clear that these points correspond to the extreme points of the rectangular partition $P^2 = \prod_{i=1}^{n} \mathcal{P}_i^2$. Pick one of these points, denoted by $\bar{\boldsymbol{x}}$. We show that $\bar{\boldsymbol{x}} \in \text{conv}(\text{Sol}(\mathtt{D}^1))$. It follows from lines 1–6 of Algorithm 2 that each layer $i \in [n]$ of $\mathtt{D}^1$ includes a single node $\mathtt{v}_i$. Further, each node $\mathtt{v}_i$ is connected to $\mathtt{v}_{i+1}$ via two arcs with label values $\mathcal{P}_i^1\downarrow$ and $\mathcal{P}_i^1\uparrow$ for $i \in [n-1]$. To determine whether $\mathtt{v}_n$ is connected to the terminal node of $\mathtt{D}^1$, we need to calculate $\xi^*$ (which we refer to as $\dot{\xi}^*$ to distinguish it from that calculated for $\mathtt{D}^2$) according to line 8 of Algorithm 2. Using an argument similar to that in the proof of Proposition 3.2, we write that $\dot{\xi}^* = \sum_{i=1}^{n} \dot{\eta}_i$, where $\dot{\eta}_i \leq g_i(x_i)$ for all $x_i \in \mathcal{P}_i^1$, which is obtained from the lower bound calculation rule employed for this algorithm. We can similarly calculate the value of $\xi^*$ for $\mathtt{D}^2$ as $\xi^* = \sum_{i=1}^{n} \eta_i \leq b$, where the inequality holds by the assumption for this case, and where $\eta_i \leq g_i(x_i)$ for all $x_i \in \mathcal{P}_i^2$, which is obtained from the lower bound calculation rule employed for this algorithm. On the other hand, because $\mathcal{P}^2 \subseteq \mathcal{P}^1$, we have that

$\mathcal{P}_i^1\downarrow \leq \mathcal{P}_i^2\downarrow \leq \mathcal{P}_i^2\uparrow \leq \mathcal{P}_i^1\uparrow$ for each $i \in [n]$. As a result, due to the consistency property of the lower bound calculation rule, we have that $\dot{\eta}_i \leq \eta_i$ for each $i \in [n]$. Combining the above results, we obtain that $\dot{\xi}^* = \sum_{i=1}^n \dot{\eta}_i \leq \sum_{i=1}^n \eta_i \leq b$. Therefore, the if-condition in line 9 of Algorithm 2 is satisfied for $\mathtt{D}^1$, and thus $\mathtt{v}_n$ is connected to the terminal node of $\mathtt{D}^1$ via two arcs with label values $\mathcal{P}_n^1\downarrow$ and $\mathcal{P}_i^1\uparrow$. Consequently, $\text{Sol}(\mathtt{D}^1)$ includes all extreme points of the rectangular partition $\mathcal{P}^1$ encoded by the $\mathtt{r}$-$\mathtt{t}$ paths of this DD. Since $\mathcal{P}^2 \subseteq \mathcal{P}^1$, the extreme point $\bar{\boldsymbol{x}}$ of $\mathcal{P}^2$ is in $\text{conv}(\text{Sol}(\mathtt{D}^1))$, proving the result. □

**Proposition 5.2** *Consider a function $g(\boldsymbol{x}) : \mathcal{P} \to \mathbb{R}$, where $g(\boldsymbol{x}) = \sum_{j=1}^q g_j(\boldsymbol{x}_{H_j})$, with each $g_j(\boldsymbol{x}_{H_j})$ being a non-separable function that contains variables with indices in $H_j \subseteq [n]$. Let $C \subseteq [n]$ and $I = [n]\backslash C$ represent the index sets of continuous and integer variable, respectively. Consider $\mathcal{P} = \prod_{i=1}^n \mathcal{P}_i$, where $\mathcal{P}_i = [\mathcal{P}_i\downarrow, \mathcal{P}_i\uparrow]$ for $i \in C$ and $\mathcal{P}_i = [\mathcal{P}_i\downarrow, \mathcal{P}_i\uparrow] \cap \mathbb{Z}$ for $i \in I$. Define $\mathcal{F}^k = \{\boldsymbol{x} \in \mathcal{P}^k \mid g(\boldsymbol{x}) \leq b\}$ for $k = 1, 2$, where $b \in \mathbb{R}$, and $\mathcal{P}^j = \prod_{i=1}^n \mathcal{P}_i^j \subseteq \mathcal{P}$. For each $k = 1, 2$, let $\mathtt{D}^k$ be the DD constructed via Algorithm 4 or 5 for a single sub-domain partition $\mathcal{P}_i^k$ of variable $x_i$ for each $i \in [n]$ using a lower bound calculation rule consistent with respect to $g_k(\boldsymbol{x}_{H_k})$ over $\prod_{l\in H_k} \mathcal{P}_l$ for each $k \in [q]$. If $\mathcal{P}^2 \subseteq \mathcal{P}^1$, then* $\text{conv}(\text{Sol}(\mathtt{D}^2)) \subseteq \text{conv}(\text{Sol}(\mathtt{D}^1))$.

*Proof* The main idea of the proof is similar to that of Proposition 5.1, with the key difference being that the calculation of lower bounds for state values requires the use of the backtracking method from Section 3.3, which is demonstrated next. Since there is only one sub-domain partition for each variable, the DDs constructed via Algorithm 4 and 5 are the same. Thus, we show the result assuming Algorithm 4 is used. According to this algorithm, because $\mathtt{D}^2$ has a unit width, we denote by $\mathtt{u}_i$ the only node at each node layer $i \in [n]$ of this DD. Following the top-down construction steps of Algorithm 4, for each $i \in [n-1]$, $\mathtt{u}_i$ is connected via two arcs with label values $\mathcal{P}_i^2\downarrow$ and $\mathcal{P}_i^2\uparrow$ to $\mathtt{u}_{i+1}$. For layer $i = n$, we refer to the $\xi$ value computed in line 9 of this algorithm as $\xi^*$ to distinguish it from the values calculated at the previous layers. There are two cases for $\xi^*$.

For the first case, assume that $\xi^* > b$. Then, the if-condition in line 14 of Algorithm 4 is not satisfied. Therefore, node $\mathtt{u}_n$ is not connected to the terminal node $\mathtt{t}$ of $\mathtt{D}^2$. As a result, there is no $\mathtt{r}$-$\mathtt{t}$ path in this DD, leading to an empty solution set, i.e., $\text{conv}(\text{Sol}(\mathtt{D}^2)) = \text{Sol}(\mathtt{D}^2) = \emptyset$. This proves the result since $\emptyset \subseteq \text{conv}(\text{Sol}(\mathtt{D}^1))$.

For the second case, assume that $\xi^* \leq b$. Then, the if-condition in line 14 of Algorithm 4 is satisfied, and node $\mathtt{u}_n$ is connected to the terminal node $\mathtt{t}$ of $\mathtt{D}^2$ via two arcs with label values $\mathcal{P}_n^2\downarrow$ and $\mathcal{P}_n^2\uparrow$. Therefore, the solution set of $\mathtt{D}^2$ contains $2^n$ points encoded by all the $\mathtt{r}$-$\mathtt{t}$ paths of the DD, each composed of arcs with label values $\mathcal{P}_i^2\downarrow$ or $\mathcal{P}_i^2\uparrow$ for $i \in [n]$. It is clear that these points correspond to the extreme points of the rectangular partition $\mathcal{P}^2 = \prod_{i=1}^n \mathcal{P}_i^2$. Pick one of these points, denoted by $\bar{\boldsymbol{x}}$. We show that $\bar{\boldsymbol{x}} \in \text{conv}(\text{Sol}(\mathtt{D}^1))$. It follows from lines 1–12 of Algorithm 4 that each layer $i \in [n]$ of $\mathtt{D}^1$ includes a single node $\mathtt{v}_i$. Further, each node $\mathtt{v}_i$ is connected to $\mathtt{v}_{i+1}$ via two arcs with label values $\mathcal{P}_i^1\downarrow$ and $\mathcal{P}_i^1\uparrow$ for $i \in [n-1]$. To determine whether $\mathtt{v}_n$ is connected to the terminal node of $\mathtt{D}^1$, we need to calculate $\xi^*$ (which we refer to as $\dot{\xi}^*$ to distinguish it from that calculated for $\mathtt{D}^2$) according to line 9 of Algorithm 4. Since all the nodes $\mathtt{v}_1, \ldots, \mathtt{v}_n$ are connected via the arcs described above, we conclude that the sub-domain of variable $x_i$ relative to node $\mathtt{v}_j$ for each $i \in [n-1]$ and $j > i$ is the entire variable domain $\mathcal{P}_i^1$. Using an argument similar to that in the proof of Proposition 3.5, we write that $\dot{\xi}^* = \sum_{k=1}^q \dot{\eta}_k$, where $\dot{\eta}_k \leq g_k(\boldsymbol{x}_{H_k})$ for all $x_j \in \mathcal{P}_j^1$ with $j \in H_k$, which is obtained from the lower bound calculation rule employed for this algorithm. We can similarly calculate the value of $\xi^*$ for $\mathtt{D}^2$ as $\xi^* = \sum_{k=1}^q \eta_k \leq b$, where the inequality holds by the assumption for this case, and where $\eta_k \leq g_k(\boldsymbol{x}_{H_k})$ for all $x_j \in \mathcal{P}_j^2$ with $j \in H_k$, which is obtained from the lower bound calculation rule employed for this algorithm. On the other hand, because $\mathcal{P}^2 \subseteq \mathcal{P}^1$, we have that $\mathcal{P}_i^1\downarrow \leq \mathcal{P}_i^2\downarrow \leq \mathcal{P}_i^2\uparrow \leq \mathcal{P}_i^1\uparrow$ for each $i \in [n]$. As a result, due to consistency property of the lower bound calculation rule, we have that $\dot{\eta}_k \leq \eta_k$ for each $k \in [q]$. Combining the above results, we obtain that $\dot{\xi}^* = \sum_{i=1}^q \dot{\eta}_q \leq \sum_{i=1}^q \eta_q \leq b$. Therefore, the if-condition in line 14 of Algorithm 4 is satisfied for $\mathtt{D}^1$, and thus $\mathtt{v}_n$ is connected to the terminal node of $\mathtt{D}^1$ via two arcs with

label values $\mathcal{P}_n^1\downarrow$ and $\mathcal{P}_n^1\uparrow$. Consequently, $\text{Sol}(\mathtt{D}^1)$ includes all extreme points of the rectangular partition $\mathcal{P}^1$ encoded by the `r-t` paths of this DD. Since $\mathcal{P}^2 \subseteq \mathcal{P}^1$, the extreme point $\bar{\boldsymbol{x}}$ of $\mathcal{P}^2$ is in $\text{conv}(\text{Sol}(\mathtt{D}^1))$, proving the result. □

Although Propositions 5.1 and 5.2 imply that the dual bounds obtained by our proposed DD-based outer approximation framework can improve through SB&B as a result of partitioning the variables' domain, an additional property of the employed lower bound calculation rules is needed to guarantee convergence to the global optimal value of the problem, as described next.

**Definition 5.2** Consider a function $g(\boldsymbol{x}) : \mathcal{D} \to \mathbb{R}$, where $C \subseteq [n]$ and $I = [n] \setminus C$ represent the index sets of continuous and integer variable, respectively, and where $\mathcal{D} = \prod_{i=1}^n \mathcal{D}_i$ with $\mathcal{D}_i = [\mathcal{D}_i\downarrow, \mathcal{D}_i\uparrow]$ for $i \in C$ and $\mathcal{D}_i = [\mathcal{D}_i\downarrow, \mathcal{D}_i\uparrow] \cap \mathbb{Z}$ for $i \in I$. Consider a lower bound calculation rule that outputs a lower bound $\eta(\bar{\mathcal{D}})$ for $g(\boldsymbol{x})$ over a box domain $\bar{\mathcal{D}} \subseteq \mathcal{D}$. We say that this lower bound calculation rule is *convergent with respect to* $g(\boldsymbol{x})$ *over* $\mathcal{D}$ if (i) it is consistent with respect to $g(\boldsymbol{x})$ over $\mathcal{D}$, and (ii) $\lim_{j\to\infty} \eta(\mathcal{D}^j) = g(\bar{\boldsymbol{x}})$ for any nested sequence of box domains $\{\mathcal{D}^j\}_{j=1}^\infty$ with $\mathcal{D}^j \subseteq \mathcal{D}$ that converges (in the Hausdorff sense) to a singleton set $\{\bar{x}\}$, i.e., $\{\mathcal{D}^j\} \searrow \{\bar{\boldsymbol{x}}\}$.

As the next step, Propositions 5.3 and 5.4 give the convergence results for constraints with separable and non-separable terms, respectively.

**Proposition 5.3** *Consider a separable function $g(\boldsymbol{x}) : \mathcal{P} \to \mathbb{R}$, where $g(\boldsymbol{x}) = \sum_{i=1}^n g_i(x_i)$. Let $C \subseteq [n]$ and $I = [n] \setminus C$ represent the index sets of continuous and integer variable, respectively, and let $\mathcal{P} = \prod_{i=1}^n \mathcal{P}_i$ with $\mathcal{P}_i = [\mathcal{P}_i\downarrow, \mathcal{P}_i\uparrow]$ for $i \in C$ and $\mathcal{P}_i = [\mathcal{P}_i\downarrow, \mathcal{P}_i\uparrow] \cap \mathbb{Z}$ for $i \in I$. Define $\mathcal{F}^j = \{\boldsymbol{x} \in \mathcal{P}^j \mid g(\boldsymbol{x}) \leq b\}$ for any $j \in \mathbb{N}$, where $b \in \mathbb{R}$, and $\mathcal{P}^j = \prod_{i=1}^n \mathcal{P}_i^j \subseteq \mathcal{P}$. For $j \in \mathbb{N}$, let $\mathtt{D}^j$ be the DD representing $\mathcal{F}^j$, which is constructed via Algorithm 2 or 3 for the single sub-domain partition $P_i^j$ for $i \in [n]$ using a lower bound calculation rule convergent with respect to $g_i(x_i)$ over $\mathcal{P}_i$. Assume that $\{\mathcal{P}^1, \mathcal{P}^2, \dots\}$, with $\mathcal{P}^j \subseteq \mathcal{P}$, is a nested sequence of rectangular partitions of the variables' domain created through the SB&B process, i.e., $\mathcal{P}^j \supseteq \mathcal{P}^{j+1}$ for each $j \in \mathbb{N}$. Let $\tilde{\boldsymbol{x}} \in \mathbb{R}^n$ with $\tilde{x}_i \in \mathbb{Z}$ for $i \in I$ be the point in a singleton set to which the above sequence converges (in the Hausdorff sense), i.e., $\{\mathcal{P}^j\} \searrow \{\tilde{\boldsymbol{x}}\}$. Then, the following statements hold:*

*(i) If $g(\tilde{\boldsymbol{x}}) \leq b$, then $\left\{\text{conv}(\text{Sol}(\mathtt{D}^j))\right\} \searrow \{\tilde{\boldsymbol{x}}\}$.*
*(ii) If $g(\tilde{\boldsymbol{x}}) > b$, then there exists $m \in \mathbb{N}$ such that $\text{Sol}(\mathtt{D}^j) = \emptyset$ for all $j \geq m$.*

*Proof* (i) Assume that $g(\tilde{\boldsymbol{x}}) \leq b$. Since there is only one sub-domain partition for each variable, the DDs constructed via Algorithm 2 and 3 are the same. Thus, we show the result assuming Algorithm 2 is used. Consider $j \in \mathbb{N}$. Note that $\mathcal{F}^j \subseteq \text{conv}(\mathcal{F}^j) \subseteq \text{conv}(\text{Sol}(\mathtt{D}^j))$ according to Proposition 3.2. We prove that $\text{Sol}(\mathtt{D}^j) \subseteq \mathcal{P}^j$. There are two cases. For the first case, assume that the if-condition in line 9 of Algorithm 2 is not satisfied. It implies that there are no `r-t` paths in $\mathtt{D}^j$, i.e., $\text{Sol}(\mathtt{D}^j) = \emptyset \subseteq \mathcal{P}^j$. For the second case, assume that the if-condition in line 9 of Algorithm 2 is satisfied. Then, $\text{Sol}(\mathtt{D}^j)$ contains the points encoded by all `r-t` paths in $\mathtt{D}^j$ composed of arcs with label values $\mathcal{P}_i^j\downarrow$ or $\mathcal{P}_i^j\uparrow$ for each $i \in [n]$, i.e., $\text{Sol}(\mathtt{D}^j) \subseteq \mathcal{P}^j$. As a result, $\text{conv}(\text{Sol}(\mathtt{D}^j)) \subseteq \mathcal{P}^j$. Because $\{\mathcal{P}^1, \mathcal{P}^2, \dots\}$ is a nested set sequence, it follows from Proposition 5.1 that the sequence $\{\text{conv}(\text{Sol}(\mathtt{D}_1)), \text{conv}(\text{Sol}(\mathtt{D}_2)), \dots\}$ is also nested, i.e., $\text{conv}(\text{Sol}(\mathtt{D}^j)) \supseteq \text{conv}(\text{Sol}(\mathtt{D}^{j+1}))$ for $j \in \mathbb{N}$. On the other hand, we can write $\mathcal{F}^j = \{\boldsymbol{x} \in \mathbb{R}^n \mid g(\boldsymbol{x}) \leq b\} \cap \mathcal{P}^j$ by definition. Since $\{\mathcal{P}^j\} \searrow \{\tilde{\boldsymbol{x}}\}$, we obtain that $\{\mathcal{F}^j\} \searrow \{\boldsymbol{x} \in \mathbb{R}^n \mid g(\boldsymbol{x}) \leq b\} \cap \{\tilde{\boldsymbol{x}}\} = \{\tilde{\boldsymbol{x}}\}$ since $g(\tilde{\boldsymbol{x}}) \leq b$ by assumption. Therefore, based on the previous arguments, we can write that $\mathcal{F}^j \subseteq \text{conv}(\text{Sol}(\mathtt{D}^j)) \subseteq \mathcal{P}^j$. Because $\{\mathcal{F}^j\} \searrow \{\tilde{\boldsymbol{x}}\}$ and $\{\mathcal{P}^j\} \searrow \{\tilde{\boldsymbol{x}}\}$, we conclude that $\left\{\text{conv}(\text{Sol}(\mathtt{D}^j))\right\} \searrow \{\tilde{\boldsymbol{x}}\}$.

(ii) Assume that $g(\tilde{\boldsymbol{x}}) > b$. We prove the result for the general case where both integer and continuous variables are present in the model, i.e., $I \neq \emptyset$ and $C \neq \emptyset$. The result for the case where $I = \emptyset$ (resp., $C = \emptyset$) follows similarly, by omitting the arguments used for $I$ (resp., $C$). For each DD $\mathtt{D}^j$ for $j \in \mathbb{N}$, using an argument similar to that of Proposition 5.1, we can calculate the value

$\xi^* = \sum_{i=1}^n \eta_i$ in line 8 of Algorithm 2, where $\eta_i \leq g_i(x_i)$ for all $x_i \in \mathcal{P}_i^j$, which is obtained from the lower bound calculation rule employed for this algorithm. Since the sequence of domain partitions $\{\mathcal{P}^j\}$ is nested and converges to $\{\tilde{\boldsymbol{x}}\}$, we can equivalently write that the sequence of variable lower bounds $\{\mathcal{P}_i^1\downarrow, \mathcal{P}_i^2\downarrow, \dots\}$ in these partitions is monotone non-decreasing and converges to $\tilde{x}_i$ for $i \in [n]$, i.e., $\mathcal{P}_i^1\downarrow \leq \mathcal{P}_i^2\downarrow \leq \dots$, and $\lim_{j\to\infty} \mathcal{P}_i^j\downarrow = \tilde{x}_i$. Similarly, the sequence of variable upper bounds $\{\mathcal{P}_i^1\uparrow, \mathcal{P}_i^2\uparrow, \dots\}$ in these partitions is monotone non-increasing and converge to $\tilde{x}_i$ for $i \in [n]$, i.e., $\mathcal{P}_i^1\uparrow \geq \mathcal{P}_i^2\uparrow \geq \dots$, and $\lim_{j\to\infty} \mathcal{P}_i^j\uparrow = \tilde{x}_i$. Therefore, for each $i \in I$, since $\tilde{x_i} \in \mathbb{Z}$ and the partitions $\mathcal{P}_i^j$ include integer values for all $j \in \mathbb{N}$ by design, there exists $m_i \in \mathbb{N}$ such that $\mathcal{P}_i^{m_i}\downarrow = \mathcal{P}_i^{m_i}\uparrow = \tilde{x}_i$. Thus, the convergence property of the employed lower bound calculation rule implies that $\eta_i = g(\tilde{x}_i)$ for any $\mathtt{D}^j$ with $j \geq m_i$. On the other hand, it follows from the assumption of this case that $g(\tilde{\boldsymbol{x}}) = \sum_{i=1}^n g_i(\tilde{x}_i) > b$. Define $\epsilon = \frac{\sum_{i=1}^n g_i(\tilde{x}_i) - b}{|C|} > 0$. For each $i \in C$, by definition of convergence for the lower bound calculating rule, the lower bounds $\eta_i$ of $g_i(x_i)$ computed in line 5 of Algorithm 2 monotonically converge to $g_i(\tilde{x}_i)$ as the domain partitions $\mathcal{P}_i^j$ converge to $\{\tilde{x}_i\}$. Therefore, there exists $m_i \in \mathbb{N}$ such that $\eta_i > g_i(\tilde{x}_i) - \epsilon$, where $\eta_i$ is computed over the domain partition $\mathcal{P}_i^j$ for all $j \geq m_i$. Pick $m = \max_{i\in[n]} m_i$. The value of $\xi^*$ for $\mathtt{D}^m$ is calculated as $\xi^* = \sum_{i\in I} \eta_i + \sum_{i\in C} \eta_i > \sum_{i\in I} g_i(\tilde{x}_i) + \sum_{i\in C} g_i(\tilde{x}_i) - \epsilon = \sum_{i=1}^n g_i(\tilde{x}_i) - |C|\epsilon = b$, where the inequality follows from the value of $\eta_i$ obtained in the previous parts for both $i \in I$ and $i \in C$, and the last equality is due to the definition of $\epsilon$ given previously. Since $\xi^* > b$, the if-condition in line 9 of Algorithm 2 is not satisfied, and thus the single node $\mathtt{v}_n$ at layer $n$ of $\mathtt{D}^m$ is not connected to the terminal node of this DD, implying that $\mathrm{Sol}(\mathtt{D}^m) = \emptyset$. Finally, it follows from Proposition 5.1 that $\mathrm{Sol}(\mathtt{D}^j) \subseteq \mathrm{conv}(\mathrm{Sol}(\mathtt{D}^j)) \subseteq \mathrm{conv}(\mathrm{Sol}(\mathtt{D}^m)) = \emptyset$, for all $j > m$, proving the result.

□

**Proposition 5.4** *Consider a function $g(\boldsymbol{x}) : \mathcal{P} \to \mathbb{R}$, where $g(\boldsymbol{x}) = \sum_{j=1}^q g_j(\boldsymbol{x}_{H_j})$, with each $g_j(\boldsymbol{x}_{H_j})$ being a non-separable function that contains variables with indices in $H_j \subseteq [n]$. Let $C \subseteq [n]$ and $I = [n]\backslash C$ represent the index sets of continuous and integer variable, respectively. Consider $\mathcal{P} = \prod_{i=1}^n \mathcal{P}_i$, where $\mathcal{P}_i = [\mathcal{P}_i\downarrow, \mathcal{P}_i\uparrow]$ for $i \in C$ and $\mathcal{P}_i = [\mathcal{P}_i\downarrow, \mathcal{P}_i\uparrow] \cap \mathbb{Z}$ for $i \in I$. Define $\mathcal{F}^j = \{\boldsymbol{x} \in \mathcal{P}^j \mid g(\boldsymbol{x}) \leq b\}$ for any $j \in \mathbb{N}$, where $\mathcal{P}^j = \prod_{i=1}^n \mathcal{P}_i^j \subseteq \mathcal{P}$. For $j \in \mathbb{N}$, let $\mathtt{D}^j$ be the DD representing $\mathcal{F}^j$, which is constructed via Algorithm 4 or 5 for the single sub-domain partition $\mathcal{P}_i^j$ for $i \in [n]$ using a lower bound calculation rule convergent with respect to $g_k(\boldsymbol{x}_{H_k})$ over $\prod_{l\in H_k} \mathcal{P}_l$ for each $k \in [q]$. Assume that $\{\mathcal{P}^1, \mathcal{P}^2, \dots\}$, with $\mathcal{P}^j \subseteq \mathcal{P}$, is a nested sequence of rectangular partitions of the variables domain created through the SB&B process, i.e., $\mathcal{P}^j \supseteq \mathcal{P}^{j+1}$ for each $j \in \mathbb{N}$. Let $\tilde{\boldsymbol{x}} \in \mathbb{R}^n$ with $\tilde{x}_i \in \mathbb{Z}$ for $i \in I$ be the point in a singleton set to which the above sequence converges (in the Hausdorff sense), i.e., $\{\mathcal{P}^j\} \searrow \{\tilde{\boldsymbol{x}}\}$. Then, the following statements hold:*

*(i) If $g(\tilde{\boldsymbol{x}}) \leq b$, then $\left\{\mathrm{conv}(\mathrm{Sol}(\mathtt{D}^j))\right\} \searrow \{\tilde{\boldsymbol{x}}\}$.*

*(ii) If $g(\tilde{\boldsymbol{x}}) > b$, then there exists $m \in \mathbb{N}$ such that $\mathrm{Sol}(\mathtt{D}^j) = \emptyset$ for all $j \geq m$.*

*Proof* (i) The proof of this part follows from arguments similar to those given in part (i) of Proposition 5.3, but uses the result of Proposition 3.5 in place of Proposition 3.2.

(ii) Assume that $g(\tilde{\boldsymbol{x}}) > b$. For each DD $\mathtt{D}^j$ for $j \in \mathbb{N}$, using a similar approach to that of Proposition 5.2, we can calculate the value $\xi^* = \sum_{k=1}^q \eta_k$ at layer $n$ of the DD in line 9 of Algorithm 4, where $\eta_k \leq g_k(\boldsymbol{x}_{H_k})$ for all $x_i \in \mathcal{P}_i^j$ for each $i \in H_k$, which is obtained from the lower bound calculation rule employed for this algorithm. In this relation, we have used the fact that $\mathtt{D}^j$ has a unit width, thus the sub-domain of each variable $x_i$ relative to the single node at any layer of the DD is the entire domain $\mathcal{P}_i^j$. The assumption of this case implies that $g(\tilde{\boldsymbol{x}}) = \sum_{k=1}^q g_k(\tilde{\boldsymbol{x}}_{H_k}) > b$. Define $\epsilon = \frac{\sum_{k=1}^q g_k(\tilde{\boldsymbol{x}}_{H_k}) - b}{q} > 0$. For each $k \in [q]$, by definition of convergence for the lower bound calculation rule, the lower bounds $\eta_k$ of $g_k(\boldsymbol{x}_{H_k})$ computed in line 6 of Algorithm 4 monotonically converge to $g_k(\tilde{\boldsymbol{x}}_{H_k})$ as the domain partitions $\mathcal{P}_i^j$ converge to $\{\tilde{x}_i\}$. Therefore, there exists $m_i \in \mathbb{N}$ such that $\eta_k > g_k(\tilde{\boldsymbol{x}}_{H_k}) - \epsilon$ computed over the domain

partition $\mathcal{P}_i^j$ for all $j \geq m_i$. Pick $m = \max_{i\in[n]} m_i$. The value of $\xi^*$ for $\mathtt{D}^m$ is calculated as $\xi^* = \sum_{i=1}^q \eta_k > \sum_{i=1}^q \big(g_k(\tilde{\boldsymbol{x}}_{H_k}) - \epsilon\big) = \sum_{i=1}^q g_k(\tilde{\boldsymbol{x}}_{H_k}) - q\epsilon = b$, where the inequality follows from the value of $\eta_k$ computed above, and the last equality is due to the definition of $\epsilon$ given previously. Since $\xi^* > b$, the if-condition in line 14 of Algorithm 4 is not satisfied, and thus the single node $\mathtt{v}_n$ at layer $n$ of $\mathtt{D}^m$ is not connected to the terminal node of this DD, implying that $\text{Sol}(\mathtt{D}^m) = \emptyset$. Finally, it follows from Proposition 5.2 that $\text{Sol}(\mathtt{D}^j) \subseteq \text{conv}(\text{Sol}(\mathtt{D}^j)) \subseteq \text{conv}(\text{Sol}(\mathtt{D}^m)) = \emptyset$, for all $j > m$, proving the result.

□

The result of Propositions 5.3 and 5.4 show that the convex hull of the solution set, as represented by the DDs constructed through the proposed convexification technique, converges to the feasible region of the underlying MINLP constraint during the SB&B process. This guarantees convergence to the global optimal value of the MINLP (if one exists), as implemented in Algorithm 1.

Since the convergence results above depend on the convergence properties of the lower bound calculation rules used in the DD construction method, we conclude this section by outlining the conditions required to achieve these properties. First, we demonstrate that a necessary condition for this property pertains to a variant of lower semicontinuity in the functions defined over the space of their continuous variables, if such variables are present. Consider a function $g(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$, where $I \subset [n]$ and $C = [n] \setminus I$ represent the index sets of integer and continuous variables, respectively. Following the Definition 3.2, we denote by $g(\boldsymbol{x}_C, \bar{\boldsymbol{x}}_I) : \mathbb{R}^{|C|} \to \mathbb{R}$ the restriction of $g(\boldsymbol{x})$ in the space of $\boldsymbol{x}_C$, where variables $x_k$ are fixed at value $\bar{x}_k$ for all $k \in I$. Further, we say that $g(\boldsymbol{x})$ is *lower semicontinuous over a domain* $\mathcal{D} \subseteq \mathbb{R}^n$ if for any point $\bar{\boldsymbol{x}} \in \mathcal{D}$ and any $\epsilon > 0$, there exists $\delta > 0$ such that $g(\boldsymbol{x}) > g(\bar{\boldsymbol{x}}) - \epsilon$ for every $\boldsymbol{x} \in \mathcal{D}$ with $||\boldsymbol{x} - \bar{\boldsymbol{x}}||_2 < \delta$.

**Proposition 5.5** *Consider a function $g(\boldsymbol{x}) : \mathcal{D} \to \mathbb{R}$, where $I \subset [n]$ and $C = [n] \setminus I$ represent the index sets of integer and continuous variables, respectively, and where $\mathcal{D} = \prod_{i=1}^n \mathcal{D}_i$ with $\mathcal{D}_i = [\mathcal{D}_i\downarrow, \mathcal{D}_i\uparrow]$ for $i \in C$ and $\mathcal{D}_i = [\mathcal{D}_i\downarrow, \mathcal{D}_i\uparrow] \cap \mathbb{Z}$ for $i \in I$. Consider a lower bound calculation rule that outputs a lower bound $\eta(\bar{\mathcal{D}})$ for $g(\boldsymbol{x})$ over a box domain $\bar{\mathcal{D}} \subseteq \mathcal{D}$. If this lower bound calculation rule is convergent with respect to $g(\boldsymbol{x})$ over $\mathcal{D}$, then $g(\boldsymbol{x}_C, \bar{\boldsymbol{x}}_I)$ is lower semicontinuous over $\prod_{i\in C} \mathcal{D}_i$ for any $\bar{\boldsymbol{x}}_I \in \prod_{i\in I} \mathcal{D}_i$.*

*Proof* Assume by contradiction that there exists $\bar{\boldsymbol{x}}_I \in \prod_{i\in I} \mathcal{D}_i$ such that $g(\boldsymbol{x}_C, \bar{\boldsymbol{x}}_I)$ is not lower semicontinuous over $\prod_{i\in C} \mathcal{D}_i$. Therefore, there exist $\tilde{\boldsymbol{x}}_C \in \prod_{i\in C} \mathcal{D}_i$ and $\epsilon > 0$ such that, for any $\delta > 0$, there is a point $\hat{\boldsymbol{x}}_C^\delta \in \prod_{i\in C} \mathcal{D}_i$ with $g(\hat{\boldsymbol{x}}_C^\delta, \bar{\boldsymbol{x}}_I) \leq g(\tilde{\boldsymbol{x}}_C, \bar{\boldsymbol{x}}_I) - \epsilon$ and $||\hat{\boldsymbol{x}}_C^\delta - \tilde{\boldsymbol{x}}_C||_2 < \delta$. Consider a sequence $\{\delta^j\}$ with $\delta^j = 1/j$ for $j \in \mathbb{N}$. Define a sequence of box domains $\{\mathcal{D}^j\}$ with $\mathcal{D}^j = \prod_{i=1}^n \mathcal{D}_i^j$ where $\mathcal{D}_i^j = [\bar{x}_i, \bar{x}_i]$ for each $i \in I$ and $\mathcal{D}_i^j = [\tilde{x}_i - \delta^j, \tilde{x}_i + \delta^j] \cap [\mathcal{D}_i\downarrow, \mathcal{D}_i\uparrow]$ for each $i \in C$. It is clear that $\{\mathcal{D}^j\}$ converges to $\{(\tilde{\boldsymbol{x}}_C, \bar{\boldsymbol{x}}_I)\}$. Furthermore, it follows from the definition of $\hat{\boldsymbol{x}}_C^\delta$ that $(\hat{\boldsymbol{x}}_C^{\delta^j}, \bar{\boldsymbol{x}}_I) \in \mathcal{D}^j$ for each $j \in \mathbb{N}$. As a result, the lower bound $\eta(\mathcal{D}^j)$ obtained by the lower bound calculation rule satisfies $\eta(\mathcal{D}^j) \leq g(\hat{\boldsymbol{x}}_C^{\delta^j}, \bar{\boldsymbol{x}}_I) \leq g(\tilde{\boldsymbol{x}}_C, \bar{\boldsymbol{x}}_I) - \epsilon < g(\tilde{\boldsymbol{x}}_C, \bar{\boldsymbol{x}}_I) - \frac{\epsilon}{2}$, where the second inequality follows from the contradiction assumption, and the last inequality holds because $\epsilon > 0$ by assumption. This is a contradiction to the assumption that the considered lower bound calculation rule is convergent with respect to $g(\boldsymbol{x})$ over $\mathcal{D}$ as for the domain sequence $\{\mathcal{D}^j\}$, we must have $\lim_{j\to\infty} \eta(\mathcal{D}^j) \neq g(\tilde{\boldsymbol{x}}_C, \bar{\boldsymbol{x}}_I)$. □

Next, we show that the lower bound calculation rules introduced in Section 3.5 possess the convergence property when applied to functions that satisfy the necessary condition outlined in Proposition 5.5. In other words, as long as this functional property for the MINLP is fulfilled, our proposed lower bound calculation rules guarantee convergence to a global solution. Considering that the lower semicontinuity of Proposition 5.5 holds for a broad range of functions commonly used in MINLP models, including test instances in the MINLP library, our proposed framework provides a powerful tool for globally solving various families of MINLPs.

**Proposition 5.6** *Consider a monotone function $g(\boldsymbol{x}) : \mathcal{D} \to \mathbb{R}$, where $I \subseteq [n]$ and $C = [n] \setminus I$ represent the index sets of integer and continuous variables, respectively, and where $\mathcal{D} = \prod_{i=1}^{n} \mathcal{D}_i$ with $\mathcal{D}_i = [\mathcal{D}_i\downarrow, \mathcal{D}_i\uparrow]$ for $i \in C$ and $\mathcal{D}_i = [\mathcal{D}_i\downarrow, \mathcal{D}_i\uparrow] \cap \mathbb{Z}$ for $i \in I$. Assume that $g(\boldsymbol{x}_C, \bar{\boldsymbol{x}}_I)$ is lower semicontinuous over $\prod_{i\in C} \mathcal{D}_i$ for any $\bar{\boldsymbol{x}}_I \in \prod_{i\in I} \mathcal{D}_i$. Then, the lower bound calculation rule described in Proposition 3.8 is convergent with respect to $g(\boldsymbol{x})$ over $\mathcal{D}$.*

*Proof* Let $\eta(\bar{\mathcal{D}})$ be the lower bound of $g(\boldsymbol{x})$ over $\bar{\mathcal{D}} \subseteq \mathcal{D}$ calculated by the considered lower bound calculation rule. It follows from the definition of Proposition 3.8 that $\eta(\bar{\mathcal{D}}) = \min_{\boldsymbol{x}\in\bar{\mathcal{D}}}\{g(\boldsymbol{x})\}$. This definition implies that $\eta(\mathcal{D}^1) \leq \eta(\mathcal{D}^2)$ for any $\mathcal{D}^2 \subseteq \mathcal{D}^1 \subseteq \mathcal{D}$, proving condition (i) of convergence property. To prove condition (ii), consider a nested domain sequence $\{\mathcal{D}^j\}$ with $\mathcal{D}^j = \prod_{i=1}^{n} \mathcal{D}_1^j \subseteq \mathcal{D}$ for $j \in \mathbb{N}$ such that $\{\mathcal{D}^j\} \searrow \{\tilde{\boldsymbol{x}}\}$ for some $\tilde{\boldsymbol{x}} \in \mathcal{D}$. On the one hand, since $\tilde{\boldsymbol{x}} \in \mathcal{D}$, we must have $\tilde{x}_i \in \mathbb{Z}$ for $i \in I$. Further, for each $i \in I$, we have $\mathcal{D}_i^j \subseteq \mathbb{Z}$ for all $j \in \mathbb{N}$ by definition. Thus, there exist $\bar{m} \in \mathbb{N}$ such that $\mathcal{D}_i^j = [\tilde{x}_i, \tilde{x}_i]$ for each $i \in I$ and $j \geq \bar{m}$. On the other hand, it follows from the lower semicontinuity of $g(\boldsymbol{x}_C, \tilde{\boldsymbol{x}}_I)$ over $\prod_{i\in C} \mathcal{D}_i$ that, for any $\epsilon > 0$, there exists $\hat{m} \in \mathbb{N}$ such that $g(\boldsymbol{x}_C, \tilde{\boldsymbol{x}}_I) > g(\tilde{\boldsymbol{x}}_C, \tilde{\boldsymbol{x}}_I) - \epsilon$ for each $\boldsymbol{x}_C \in \prod_{i\in C} \mathcal{D}_i^j$ for all $j \geq \hat{m}$. Define $m = \max\{\bar{m}, \hat{m}\}$. For all $\boldsymbol{x} \in \mathcal{D}^j$ with $j \geq m$, we can write $g(\boldsymbol{x}) = g(\boldsymbol{x}_C, \tilde{\boldsymbol{x}}_I) > g(\tilde{\boldsymbol{x}}_C, \tilde{\boldsymbol{x}}_I) - \epsilon = g(\tilde{\boldsymbol{x}}) - \epsilon$, where the first equality follows from the fact that $x_i = \tilde{x}_i$ for $i \in I$, the inequality is due to the relation obtained previously, and the last equality holds because $(\tilde{\boldsymbol{x}}_C, \tilde{\boldsymbol{x}}_I) = \tilde{\boldsymbol{x}}$ by definition. As a result, $\eta(\mathcal{D}^j) = \min_{\boldsymbol{x}\in\mathcal{D}^j}\{g(\boldsymbol{x})\} > g(\tilde{\boldsymbol{x}}) - \epsilon$ for all $j \geq m$. Since this result holds for any $\epsilon > 0$, we conclude that $\lim_{j\to\infty} \eta(\mathcal{D}^j) = g(\tilde{\boldsymbol{x}})$, proving the result. □

**Proposition 5.7** *Consider a function $g(\boldsymbol{x}) : \mathcal{D} \to \mathbb{R}$, where $I \subseteq [n]$ and $C = [n] \setminus I$ represent the index sets of integer and continuous variables, respectively, and where $\mathcal{D} = \prod_{i=1}^{n} \mathcal{D}_i$ with $\mathcal{D}_i = [\mathcal{D}_i\downarrow, \mathcal{D}_i\uparrow]$ for $i \in C$ and $\mathcal{D}_i = [\mathcal{D}_i\downarrow, \mathcal{D}_i\uparrow] \cap \mathbb{Z}$ for $i \in I$. Let $g^{rx}(\boldsymbol{y}) : \mathbb{R}^p \to \mathbb{R}$ be the re-indexed function of $g(\boldsymbol{x})$ with re-index mapping $R(.)$. Assume that $g^{rx}(\boldsymbol{y})$ is monotone over the box domain described by $\mathcal{P} = \prod_{j=1}^{p} \mathcal{P}_j$ where $\mathcal{P}_j = \mathcal{D}_{R(j)}$ for $j \in [p]$. Define the index set of continuous variables in $g^{rx}(\boldsymbol{y})$ as $\dot{C} = \big\{j \in [p] \big| R(j) \in C\big\}$, and define the index set of integer variables in $g^{rx}(\boldsymbol{y})$ as $\dot{I} = [p] \setminus \dot{C}$. Assume that $g^{rx}(\boldsymbol{y}_{\dot{C}}, \bar{\boldsymbol{y}}_{\dot{I}})$ is lower semicontinuous over $\prod_{j\in\dot{C}} \mathcal{P}_j$ for any $\bar{\boldsymbol{y}}_{\dot{I}} \in \prod_{j\in\dot{I}} \mathcal{P}_j$. Then, the lower bound calculation rule described in Proposition 3.9 is convergent with respect to $g(\boldsymbol{x})$ over $\mathcal{D}$.*

*Proof* Since $g^{\text{rx}}(\boldsymbol{y})$ is monotone, it follows from the definition of the lower bound calculation rule of Proposition 3.9 that $\eta(\mathcal{D}) = \min_{\boldsymbol{y}\in\mathcal{P}}\{g^{\text{rx}}(\boldsymbol{y})\}$. To prove condition (i) of the convergence property, consider box domains $\mathcal{D}^1$ and $\mathcal{D}^2$ such that $\mathcal{D}^1 \subseteq \mathcal{D}^2 \subseteq \mathcal{D}$. Define $\mathcal{P}^1 = \prod_{j=1}^{p} \mathcal{D}^1_{R(j)}$ and $\mathcal{P}^2 = \prod_{j=1}^{p} \mathcal{D}^2_{R(j)}$. It follows that $\mathcal{P}^1 \subseteq \mathcal{P}^2$. Therefore, we can write that $\eta(\mathcal{D}^1) = \min_{\boldsymbol{y}\in\mathcal{P}^1}\{g^{\text{rx}}(\boldsymbol{y})\} \geq \min_{\boldsymbol{y}\in\mathcal{P}^2}\{g^{\text{rx}}(\boldsymbol{y})\} = \eta(\mathcal{D}^2)$, which proves condition (i). For condition (ii) of the convergence property, since $g^{\text{rx}}(\boldsymbol{y})$ is monotone and $g^{\text{rx}}(\boldsymbol{y}_{\dot{C}}, \bar{\boldsymbol{y}}_{\dot{I}})$ is lower semicontinuous over $\prod_{j\in\dot{C}} \mathcal{P}_j$ for any $\bar{\boldsymbol{y}}_{\dot{I}} \in \prod_{j\in\dot{I}} \mathcal{P}_j$, Proposition 5.6 implies that the lower bound calculation rule that outputs $\dot{\eta}(\mathcal{P}) = \min_{\boldsymbol{y}\in\mathcal{P}}\{g^{\text{rx}}(\boldsymbol{y})\}$ is convergent with respect to $g^{\text{rx}}(\boldsymbol{y})$ over $\mathcal{P}$. In other words, for any nested domain sequence $\{\mathcal{P}^k\}$, with $\mathcal{P}^k \subseteq \mathcal{P}$ for $k \in \mathbb{N}$, that converges to $\tilde{\boldsymbol{y}}$, we have $\lim_{k\to\infty} \dot{\eta}(\mathcal{P}^k) = g^{\text{rx}}(\tilde{\boldsymbol{y}})$. Consider a nested domain sequence $\{\mathcal{D}^k\}$, with $\mathcal{D}^k \subseteq \mathcal{D}$ for $k \in \mathbb{N}$, that converges to $\tilde{\boldsymbol{x}}$. Define $\dot{\mathcal{P}}^k = \prod_{j=1}^{p} \mathcal{D}^k_{R(j)}$ for each $k \in \mathbb{N}$. It is clear that $\{\dot{\mathcal{P}}^k\} \searrow \{\dot{\boldsymbol{y}}\}$ where $\dot{y}_j = \tilde{x}_{R(j)}$ for each $j \in [p]$. Thus, we obtain $\lim_{k\to\infty} \dot{\eta}(\dot{\mathcal{P}}^k) = g^{\text{rx}}(\dot{\boldsymbol{y}})$ by the above definition. Using the fact that $\eta(\mathcal{D}^k) = \min_{\boldsymbol{y}\in\dot{\mathcal{P}}^k}\{g^{\text{rx}}(\boldsymbol{y})\} = \dot{\eta}(\dot{\mathcal{P}}^k)$ by definition of the considered lower bound calculation rule, we conclude that $\lim_{k\to\infty} \eta(\mathcal{D}^k) = g^{\text{rx}}(\dot{\boldsymbol{y}}) = g(\tilde{\boldsymbol{x}})$, where the last equality follows from the definition of re-indexed functions that preserve the function values at each given point. This shows that condition (ii) of the convergence property is satisfied. □

## 6 Computational Results

In this section, we present numerical results based on benchmark instances from the MINLP Library [1] to demonstrate the effectiveness and capabilities of our global solution framework, compared

to state-of-the-art global solvers. Since previous studies utilizing DD-based outer approximation [23, 22] have primarily focused on challenging problem classes that existing global solvers can handle, albeit with optimality gaps, this paper focuses on complementary problem classes that are unsolvable by current global solvers, marking them as the most challenging problems in the MINLP Library. As discussed in Section 3.5, one of the main advantages of our DD-based global solution framework, compared to existing methods, is its ability to model and solve a broader class of MINLPs, including those with complex structures that are not amenable to conventional convexification methods, such as the factorable decomposition technique that is widely used in existing solvers. To demonstrate this capability, in this section, we present computational experiments on benchmark test instances from the MINLP Library that contain functional forms not admissible by global solvers, such as BARON, the leading commercial solver, and SCIP, the leading open-source solver.

### 6.1 Algorithmic Settings

The numerical results presented in this section are obtained on a Windows 11 (64-bit) operating system, 64 GB RAM, 3.8 GHz AMD Ryzen CPU. The DD-ECP Algorithm is written in Julia v1.9 via JuMP v1.11.1, and the outer approximation models are solved with CPLEX v22.1.0. For comparison with existing global solvers, we use GAMS Release 47.6.0, equipped with BARON version 24.5.8 and SCIP version 9.1.0. In this section, we present the general settings for the algorithms used in our solution framework.

We use Algorithm 1 to solve the MINLP instances reformulated into the problem form described in (2.1a)–(2.1c). Since the model studied in this paper is bounded, for instances with variables that lack explicit bounds, we infer valid bounds based on the constraints of the model. To calculate the optimality gap, we use the primal bound reported for each instance in the MINLP Library. The stopping criteria employed in Stop_Flag in Algorithm 1 are a remaining optimality gap of 0.05 or an elapsed time of 5000 seconds, whichever occurs first. The initial LP relaxation $LP$ for each instance is obtained by removing all nonlinear constraints. The pruning rules in Prune_Node include: (i) the dual bound obtained at a node is smaller than the best current primal bound; (ii) the outer approximation is infeasible; (iii) the DD constructed for any constraints is infeasible; and (iv) the optimal solution of the outer approximation satisfies all constraints.

For the `Construct_DD` oracle, we use Algorithms 3 and 5 for constructing DDs for separable and non-separable constraints, respectively. In these algorithms, we create the sub-domain partitions for each variable $x_i$ for $i \in [n]$ such that the entire variable domain is divided into 50 intervals of equal length, i.e., $|L_i| = 50$. We impose a default width limit of $\omega = 5000$. To merge nodes at each layer, we apply the the merging policy $\texttt{Merge}^g(.)$ as described in Section 3.4. The state values at the DD nodes are computed using the lower bound calculation rules based on the monotonicity property and re-indexing techniques outlined in Section 3.5.

For the `Outer_Approx` oracle, we use the subgradient-type method of Algorithm 8 to generate cutting planes that are added to the outer approximation model. For this algorithm, we set a constant step size rule $\rho = 1$ and use the origin as the starting point for the subgradient algorithm. The termination criterion is defined by the number of iterations, which is set to 50.

For the `Branch` oracle, after obtaining the optimal solution of the outer approximation model, we select the variable whose optimal solution lies closest to the center of its domain interval. To continue the B&B process, we apply a node selection rule that prioritizes the node with the largest dual bound as the next candidate.

## 6.2 Test Instances

In this section, we present computational results for various benchmark instances from the MINLP Library. These instances feature complex functional structures that cannot be handled by existing global solvers like BARON and SCIP, and are therefore considered inadmissible/intractable. In contrast to global solvers, which fail to return dual bounds for these test instances, our DD-based global framework is capable of solving these problems and obtaining dual bounds, as shown in the tables for each instance. To provide better insight into the structure of each model, a summary of the problem specifications, their area of application, and their sources of difficulty is presented in the following sub-sections.

### *6.2.1 Test Instance:* `quantum`

This problem has applications in quantum mechanics [46]. The test instance has 2 continuous variables and 1 nonlinear constraint. This nonlinear constraint includes polynomial, fractional, exponential, and gamma functions. The following constraint illustrates a complex structure used in this model that is inadmissible in the current solvers used.

$$\frac{-0.5\sqrt{x_3}\,x_2^{\frac{1}{x_3}}\Gamma(2-\frac{0.5}{x_3})+0.5x_2^{\frac{-1}{x_3}}\Gamma(\frac{1.5}{x_3})+x_2^{\frac{-2}{x_3}}\Gamma(\frac{2.5}{x_3})}{\Gamma(\frac{0.5}{x_3})}+z=0,$$

where $\Gamma(.)$ is the gamma function. The performance of our DD-based solution framework is summarized in Table 6.1. The first two columns show the number of variables and constraints in each problem, respectively. The column labeled 'Primal' presents the primal bound for the test instance, as reported in the MINLP Library. The dual bound obtained from our proposed global method is listed in the 'Dual' column. The optimality gap is provided in the 'Gap' column and is calculated as $\frac{\text{dual bound}-\text{primal bound}}{\text{primal bound}}$. The next two columns, 'Node Explored' and 'Node Remaining,' represent the number of nodes explored and the number of nodes still open at the termination of the algorithm in the B&B tree. Finally, the last column shows the total solution time for the algorithm.

Table 6.1: Performance of the DD framework for test instance `quantum`

| Problem Specs | | Gap Closure | | | B&B Tree | | Time (s) |
|---|---|---|---|---|---|---|---|
| Var. # | Con. # | Primal | Dual | Gap | Node Explored | Node Remained | |
| 2 | 1 | −0.804 | −0.765 | 0.05 | 4 | 1 | 6.62 |

### *6.2.2 Test Instance:* `ann_fermentation_tanh`

This problem has applications in neural networks used to learn the fermentation process of gluconic acid, where the activation functions are represented by hyperbolic tangent operators [56]. The test instance has 12 continuous variables and 10 constraints. The nonlinear constraints include fractional and hyperbolic (trigonometric) functions. The following constraint illustrates a complex structure used in this model that is inadmissible in the current solvers used.

$$\tanh(x_{12})-x_8=0,$$

where tanh(.) is the hyperbolic tangent function. The performance of our proposed method when applied to this test instance is presented in Table 6.2, with columns are defined similarly to those in Table 6.1.

Table 6.2: Performance of the DD framework for test instance `ann_fermentation_tanh`

| Problem Specs | | Gap Closure | | | B&B Tree | | Time (s) |
|---|---|---|---|---|---|---|---|
| Var. # | Con. # | Primal | Dual | Gap | Node Explored | Node Remained | |
| 12 | 10 | 99.93 | 104.92 | 0.05 | 5104 | 767 | 25.74 |

### *6.2.3 Test Instance:* `fct`

This problem is included in the GAMS Model Library [47]. The test instance has 12 continuous variables and 10 constraints. The nonlinear constraints include absolute value, trigonometric, polynomial, and modulo functions. The following constraint shows a complex structure among the constraints used in this model, which is inadmissible in the current solvers.

$$\Big|\sin\big(4\bmod(x_2,\pi)\big)\Big| - x_3 = 0,$$

where $|.|$ is the absolute value function, and $\mathrm{mod}(a,b)$ is the modulo operator with dividend $a$ and divisor $b$. The performance of our proposed method when applied to this test instance is presented in Table 6.3, with columns are defined similarly to those in Table 6.1. For this test instance, we did not calculate the remaining gap as the primal bound is zero. Instead, we allowed the algorithm to run until it achieved a global optimal solution with a precision of $10^{-5}$ for the optimal value.

Table 6.3: Performance of the DD framework for test instance `fct`

| Problem Specs | | Gap Closure | | | B&B Tree | | Time (s) |
|---|---|---|---|---|---|---|---|
| Var. # | Con. # | Primal | Dual | Gap | Node Explored | Node Remained | |
| 12 | 10 | 0.00 | $6.83 \times 10^{-6}$ | – | 591 | 0 | 1668.09 |

### *6.2.4 Test Instance:* `worst`

This problem has applications in statistical models used for portfolio optimization and risk management [21]. The test instance has 35 continuous variables and 30 constraints. The nonlinear constraints include polynomial, exponential, logarithm, fractional, and modulo functions. The following constraint shows a complex structure among the constraints used in this model, which is inadmissible in the current solvers.

$$e^{-0.33889\,x_{32}} \times \big(\mathrm{erf}(x_3)\,x_{21} - 95\,\mathrm{erf}(x_{10})\big) - x_{23} = 0,$$

where erf(.) is the error function calculated as the integral of the standard normal distribution. The performance of our proposed method when applied to this test instance is presented in Table 6.4, with columns are defined similarly to those in Table 6.1.

Table 6.4: Performance of the DD framework for test instance `worst`

| Problem Specs | | Gap Closure | | | B&B Tree | | Time (s) |
|---|---|---|---|---|---|---|---|
| Var. # | Con. # | Primal | Dual | Gap | Node Explored | Node Remained | |
| 35 | 30 | −20762609 | −19583378 | 0.05 | 22 | 3 | 211.51 |

*6.2.5 Test Instance:* `ann_compressor_tanh`

This problem has applications in learning compressor powers via neural networks [56]. This test instance has 97 continuous variables and 96 constraints. The nonlinear constraints in this problem include quadratic and hyperbolic (trigonometric) functions. The following constraint shows a complex structure among the constraints used in this model, which is inadmissible in the current solvers.

$$\tanh(x_{32}) - x_{10} = 0$$

The performance of our proposed method when applied to this test instance is presented in Table 6.5, with columns are defined similarly to those in Table 6.1.

Table 6.5: Performance of the DD framework for test instance `ann_compressor_tanh`

| Problem Specs | | Gap Closure | | | B&B Tree | | Time (s) |
|---|---|---|---|---|---|---|---|
| Var. # | Con. # | Primal | Dual | Gap | Node Explored | Node Remained | |
| 97 | 96 | −213100.0 | −22331.90 | 0.05 | 867 | 52 | 276.33 |

*6.2.6 Test Instance:* `ann_peaks_tanh`

This problem has applications in neural networks [56]. This test instance has 100 continuous variables and 99 constraints. The nonlinear constraints in this problem include hyperbolic (trigonometric) functions. The following constraint shows a complex structure among the constraints used in this model, which is inadmissible in the current solvers.

$$\tanh(x_{55}) - x_6 = 0$$

The performance of our proposed method when applied to this test instance is presented in Table 6.6, with columns are defined similarly to those in Table 6.1.

Table 6.6: Performance of the DD framework for test instance `ann_peaks_tanh`

| Problem Specs | | Gap Closure | | | B&B Tree | | Time (s) |
|---|---|---|---|---|---|---|---|
| Var. # | Con. # | Primal | Dual | Gap | Node Explored | Node Remained | |
| 100 | 99 | 6.56 | 6.97 | 0.05 | 10 | 7 | 135.47 |

*6.2.7 Test Instance:* `cesam2cent`

This problem has applications in information theory, econometrics, and estimating social accounting matrices using cross entropy methods [29, 36, 49]. This test instance has 316 continuous variables and 166 constraints. The nonlinear constraints in this problem include polynomial, exponential, and cross entropy functions. The following constraint shows a complex structure among the constraints used in this model, which is inadmissible in the current solvers.

$$\sum_{i=160}^{316} \text{Centropy}(x_i, a_i) - z = 0,$$

where $a_i \in \mathbb{R}$ is a constant, and Centropy(.) is the cross-entropy function. The performance of our proposed method when applied to this test instance is presented in Table 6.7, with columns are defined similarly to those in Table 6.1.

Table 6.7: Performance of the DD framework for test instance `cesam2cent`

| Problem Specs | | Gap Closure | | | B&B Tree | | Time (s) |
|---|---|---|---|---|---|---|---|
| Var. # | Con. # | Primal | Dual | Gap | Node Explored | Node Remained | |
| 316 | 166 | −0.507 | −0.481 | 0.05 | 8 | 3 | 4604.13 |

## 7 Conclusion

We develop a novel graphical framework to globally solve general MINLPs. This paper details the key components of the framework, including (i) a method for constructing DDs that represent relaxations of the MINLP sets, (ii) a cut-generation technique that produces linear outer approximations of the underlying set, and (iii) a spatial branch-and-bound strategy that iteratively refines these approximations until convergence to a global optimal solution. Applicable to optimization problems of general structure, this framework represents the most comprehensive extension of previously developed DD-based approaches for MINLPs, addressing the longstanding need for a general-purpose DD-based method for globally solving MINLPs. Computational experiments on benchmark MINLP instances with complex structures, which are inadmissible in state-of-the-art global solvers, show the capabilities and effectiveness of the proposed framework.

## References

1. (2024) MINLP Library. `https://www.minlplib.org/`
2. Achterberg T (2009) SCIP: solving constraint integer programs. Mathematical Programming Computation 1:1–41
3. 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
4. Bao X, Sahinidis NV, Tawarmalani M (2011) Semidefinite relaxations for quadratically constrained quadratic programming: A review and comparisons. Mathematical programming 129(1):129–157
5. Bao X, Khajavirad A, Sahinidis NV, Tawarmalani M (2015) Global optimization of nonconvex problems with multilinear intermediates. Mathematical Programming Computation 7(1):1–37
6. Belotti P (2009) COUENNE: a user's manual, lehigh university, 2009. Tech. rep., Lehigh University
7. Belotti P, Lee J, Liberti L, Margot F, Wachter A (2009) Branching and bounds tightening techniques for non-convex minlp. Optimization Methods and Software 24:597–634
8. Belotti P, Kirches C, Leyffer S, Linderoth J, Luedtke J, Mahajan A (2013) Mixed-integer nonlinear optimization. Acta Numerica 22:1–131
9. Bergman D, Cire AA (2018) Discrete nonlinear optimization by state-space decompositions. Management Science 28:47–66
10. Bergman D, Cire AA, van Hoeve WJ (2015) Lagrangian bounds from decision diagrams. Constraints 20(3):346–361
11. Bergman D, Cire AA, van Hoeve WJ, Hooker J (2016) Decision Diagrams for Optimization. Springer International Publishing

12. Bergman D, Cire AA, van Hoeve WJ, Hooker J (2016) Discrete optimization with decision diagrams. INFORMS Journal on Computing 28:47–66
13. Bienstock D, Escobar M, Gentile C (2020) Mathematical programming formulations for the alternating current optimal power flow problem. 4OR 18:249–292
14. Bonami P, Lee J (2007) BONMIN user's manual. Numer Math 4:1–32
15. 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
16. Bonami P, Lee J, Leyffer S, Wächter A (2013) On branching rules for convex mixed-integer nonlinear optimization. Journal of Experimental Algorithmics (JEA) 18:2–31
17. Byrd RH, Nocedal J, Waltz R (2006) KNITRO: An integrated package for nonlinear optimization. In: di Pillo G, Roma M (eds) Large-Scale Nonlinear Optimization, Springer, pp 35–59
18. Castro M, Cire A, Beck J (2022) Decision diagrams for discrete optimization: A survey of recent advances. DOI 10.48550/arXiv.2201.11536
19. Castro PM (2017) Spatial branch-and-bound algorithm for MIQCPs featuring multiparametric disaggregation. Optimization Methods and Software 32(4):719–737
20. Coppe V, Gillard X, Schaus P (2024) Decision diagram-based branch-and-bound with caching for dominance and suboptimality detection. INFORMS Journal on Computing
21. Dahl H, Meeraus A, Zenios SA (1989) Some financial optimization models: I. risk management. Fishman-Davidson Center for the Study of the Service Sector, Wharton School ...
22. Davarnia D (2021) Strong relaxations for continuous nonlinear programs based on decision diagrams. Operations Research Letters 49(2):239–245
23. Davarnia D, Van Hoeve WJ (2020) Outer approximation for integer nonlinear programs via decision diagrams. Mathematical Programming 187:111–150
24. Davarnia D, Richard J, Tawarmalani M (2017) Simultaneous convexification of bilinear functions over polytopes with application to network interdiction. SIAM Journal on Optimization 27(3):1801–1833
25. Davarnia D, Rajabalizadeh A, Hooker J (2022) Achieving consistency with cutting planes. Mathematical Programming URL `https://doi.org/10.1007/s10107-022-01778-8`
26. Del Pia A, Khajavirad A (2018) On decomposability of multilinear sets. Mathematical Programming 170:387–415, URL `https://api.semanticscholar.org/CorpusID:13976684`
27. Duran MA, Grossmann IE (1986) An outer-approximation algorithm for a class of mixed-integer nonlinear programs. Mathematical programming 36(3):307–339
28. Gill PE, Murray W, Saunders MA (2005) SNOPT: An SQP algorithm for large-scale constrained optimization. SIAM Review 47:99–131
29. Golan A, Judge G, Miller D (1996) Maximum entropy econometrics. Tech. rep., Iowa State University, Department of Economics
30. Gonzalez JE, Cire AA, Lodi A, Rousseau LM (2020) Integrated integer programming and decision diagram search tree with an application to the maximum independent set problem. Constraints pp 1–24
31. Gupte A, Ahmed S, Cheon MS, Dey S (2013) Solving mixed integer bilinear problems using milp formulations. SIAM Journal on Optimization 23(2):721–744
32. 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)
33. Hijazi H, Bonami P, Ouorou A (2014) An outer-inner approximation for separable mixed-integer nonlinear programs. INFORMS Journal on Computing 26(1):31–44
34. van Hoeve WJ (2024) An introduction to decision diagrams for optimization. INFORMS TutORials in Operations Research pp 1–28
35. Hosseininasab A, Van Hoeve WJ (2021) Exact multiple sequence alignment by synchronized decision diagrams. INFORMS Journal on Computing 33(2):721–738

36. Judge GG, Mittelhammer RC (2011) An information theoretic approach to econometrics. Cambridge University Press
37. Khademnia E, Davarnia D (2024) Convexification of bilinear terms over network polytopes. Mathematics of Operations Research
38. Khajavirad A, Michalek JJ, Sahinidis NV (2014) Relaxations of factorable functions with convex-transformable intermediates. Mathematical Programming 144(1):107–140
39. Kronqvist J, Bernal D, Lundell A, Grossmann I (2017) A review and comparison of solvers for convex MINLP. optim. eng. 20, 397–455 (2017). Optimization and Engineering 20:397–455
40. Lee J, Leyffer S (2011) Mixed integer nonlinear programming, vol 154. Springer Science & Business Media
41. Lozano L, Smith JC (2018) A binary decision diagram based algorithm for solving a class of binary two-stage stochastic programs. Mathematical Programming pp 1–24
42. Luedtke J, Namazifar M, Linderoth J (2012) Some results on the strength of relaxations of multilinear functions. Mathematical programming 136(2):325–351
43. McCormick GP (1976) Computability of global solutions to factorable nonconvex programs: Part i — convex underestimating problems. Mathematical Programming 10:147–175, URL https://api.semanticscholar.org/CorpusID:12478942
44. Morrison DR, Jacobson SH, Sauppe JJ, Sewell EC (2016) Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. Discrete Optimization 19:79–102
45. Muts P, Nowak I, Hendrix EM (2020) The decomposition-based outer approximation algorithm for convex mixed-integer nonlinear programming. Journal of Global Optimization 77:75–96
46. Ogura A (1999) Post-gaussian variational method for quantum anharmonic oscillator. arXiv preprint physics/9905056
47. Pintér J (1999) Lgo–a model development system for continuous global optimization. user's guide. Pinter Consulting Services, Halifax, NS
48. Puranik Y, Sahinidis NV (2017) Domain reduction techniques for global nlp and minlp optimization. Constraints 22(3):338–376
49. Robinson S, Cattaneo A, El-Said M (2001) Updating and estimating a social accounting matrix using cross entropy methods. Economic Systems Research 13(1):47–64
50. Ryoo H, Sahinidis N (1996) A branch-and-reduce approach to global optimization. Journal of global optimization 8:107–138
51. Ryoo H, Sahinidis N (2001) Analysis of bounds for multilinear functions. Journal of global optimization 19:403–424
52. Sahinidis N (1996) BARON: A general purpose global optimization software package. Journal of global optimization 8:201–205
53. Salemi H, Davarnia D (2022) On the structure of decision diagram-representable mixed integer programs with application to unit commitment. Operations Research DOI 10.1287/opre.2022.2353
54. Salemi H, Davarnia D (2023) Solving unsplittable network flow problems with decision diagrams. Transportation Science DOI 10.1287/trsc.2022.1194
55. Saxena A, Bonami P, Lee J (2010) Convex relaxations of non-convex mixed integer quadratically constrained programs: extended formulations. Mathematical programming 124(1):383–411
56. Schweidtmann AM, Mitsos A (2019) Deterministic global optimization with artificial neural networks embedded. Journal of Optimization Theory and Applications 180(3):925–948
57. Serra T, Hooker JN (2019) Compact representation of near-optimal integer programming solutions. Mathematical Programming pp 1–34
58. Shectman P, Sahinidis NV (1998) A finite algorithm for global minimization of separable concave programs. Journal of global optimization 12:1–36
59. Smith EM, Pantelides CC (1999) A symbolic reformulation/spatial branch-and-bound algorithm for the global optimisation of nonconvex MINLPs. Computers & Chemical Engineering

23(4-5):457–478
60. Tawarmalani M, Sahinidis N (2001) Semidefinite relaxations of fractional programs via novel convexification techniques. Journal of Global Optimization 20:133–154
61. Tawarmalani M, Sahinidis N (2002) Convex extensions and envelopes of lower semi-continuous functions. Mathematical programming 93:247–263
62. Tawarmalani M, Sahinidis N (2004) Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. Mathematical programming 99:563–591
63. Tawarmalani M, Sahinidis N (2005) A polyhedral branch-and-cut approach to global optimization. Mathematical programming 103:225–249
64. Tits AL, Wächter A, Bakhtiari S, Urban TJ, Lawrence CT (2003) A primal-dual interior-point method for nonlinear programming with strong global and local convergence properties. SIAM Journal on Optimization 14(1):173–199
65. Tjandraatmadja C, van Hoeve WJ (2019) Target cuts from relaxed decision diagrams. INFORMS Journal on Computing 6:285–301
66. Wächter A, Biegler L (2006) On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. Mathematical Programming 106:25–57