Decision Diagrams for Discrete Optimization: A Survey of Recent Advances

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

Andre A. Cire
Department of Management, University of Toronto Scarborough and Rotman School of Management, Toronto, Ontario M1E 1A4, Canada, andre.cire@utoronto.ca

J. Christopher Beck
Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Ontario M5S 3G8, Canada, jcb@mie.utoronto.ca

In the last decade, decision diagrams (DDs) have been the basis for a large array of novel approaches for modeling and solving optimization problems. Many techniques now use DDs as a key tool to achieve state-of-the-art performance within other optimization paradigms, such as integer programming and constraint programming. This paper provides a survey of the use of DDs in discrete optimization, particularly focusing on recent developments. We classify these works into two groups based on the type of diagram (i.e., exact or approximate) and present a thorough description of their use. We discuss the main advantages of DDs, point out major challenges, and provide directions for future work.

Key words: decision diagrams; discrete optimization

1. Introduction

Decision diagrams (DDs) are graph-based structures with a large number of applications in computer science and operations research literature. While they have a long history in the Boolean function community (Akers 1978, Bryant 1986, Wegener 2000), their use in optimization is much more recent. For example, in the past ten years, researchers have successfully applied DDs to methodologies in scheduling (Cire and van Hoeve 2013), routing (Kinable et al. 2017, Castro et al. 2020a), and healthcare applications (Guo et al. 2021, Riascos-Álvarez et al. 2020), to name a few, and related literature and applications are growing steadily.

In the context of optimization, a DD is a directed acyclic graph that encodes solutions and/or their associated costs as directed paths from a root node to a terminal node. The

benefit of this representation is that it provides an explicit and potentially compact representation of the solution space of a problem that exposes network structure. Such a network can be manipulated directly, e.g., to obtain valid bounding procedures (Bergman et al. 2014c), or can be integrated with other optimization techniques. For example, DDs have been combined with integer programming (IP) solvers using a network-flow formulation over the underlying graph (Behle 2007) and with constraint programming (CP) technologies by developing inference procedures (Andersen et al. 2007).

This paper provides an in-depth treatment of DDs for discrete optimization with a focus on recent advances. Our objective is to provide a systematic classification of the field, identify unifying themes, and discuss challenges and opportunities that may be the basis of new research. Specifically, our classification partitions the area into the two pervasive DD types in related works, i.e., exact and approximate DDs. Exact DDs encode the exact problem, in that any property that holds for the DD is also valid for the original problem. Approximate DDs, in contrast, provide an over- or under-approximation of the feasible space or the objective function, and are the basis of combined and enumerative approaches (e.g., Bergman et al. 2016b).

Using this classification, we divide works employing exact DDs into four themes based on the methodology's purpose: (i) modeling, (ii) feasibility checking, (iii) solution extraction, and (iv) solution-space analysis. Table 1 presents the paper classification for exact DDs and its respective sub-classes. Similarly, we divide the works that consider approximate DDs into three themes: (i) approximate DD compilation, (ii) DD-based bounds, and (iii) CP propagation. Table 2 summarize the works that focus on approximate DDs for each class and sub-class.

The paper is organized as follows. Section 2 provides a background on DDs and the notation used throughout the paper. Section 3 presents works related to exact DDs, highlighting recent advances in the field. Similarly, Section 4 focuses on approximate DDs and novel methodologies in the literature. We concluded this survey in Section 5 with final remarks and future work directions.

2. Preliminaries

We now formally define DDs and present the notation used throughout the paper. We start with the DD encoding of optimization problems in Section 2.1, show the connections to

Table 1 Paper classification for exact decision diagrams.

Sub-class	Papers
	Modeling
Recursive Model	Hooker (2013), Hooker (2017), Bergman et al. (2016b).
Network Flow Formulation	Behle (2007), Bergman and Cire (2016a), Latour et al. (2017), Haus et al. (2017), Bergman and Cire (2018), Latour et al. (2019), Serra et al. (2019), Hosseininasab and van Hoeve (2021), Cire et al. (2019), Ploskas et al. (2019), Bergman et al. (2019), Lozano et al. (2020a), Lozano et al. (2020b), Nadarajah and Cire (2020), Bergman and Lozano (2021), Mehrani et al. (2021).
Global Constraints	Andersen et al. (2007), Cheng and Yap (2008), Cheng and Yap (2010), Perez and Régin (2014), Amilhastre et al. (2014), Perez and Régin (2015b), Perez and Régin (2015a), Perez and Régin (2016), Roy et al. (2016), Perez and Régin (2017c), Perez and Régin (2018), Verhaeghe et al. (2018), Vion and Piechowiak (2018), Verhaeghe et al. (2019), de Uña et al. (2019).
Continuous Variables	Davarnia (2021), Salemi and Davarnia (2021).
	Feasibility Checking
General	Nishino et al. (2015), Xue and van Hoeve (2019).
Cutting Planes	Becker et al. (2005), Behle (2007), Tjandraatmadja and van Hoeve (2019), Davarnia and van Hoeve (2021), Castro et al. (2021).
Benders Decomposition	Lozano and Smith (2018), Guo et al. (2021), Salemi and Davarnia (2021), Bergman and Lozano (2021).
Inference	Subbarayan (2008), Hadžić et al. (2009), Gange et al. (2011), Gange et al. (2013), Kell et al. (2015), Jung and Régin (2021).
	Solution Extraction
General	Hadžić et al. (2004).
Column Generation	Morrison et al. (2016), Kowalczyk and Leus (2018), Raghunathan et al. (2018), Riascos-Álvarez et al. (2020).
	Solution-Space Analysis
Post-Optimiality Analysis	Hadžić and Hooker (2006), Hadžić and Hooker (2007), Serra and Hooker (2020).
Solution Enumeration	Bergman and Cire (2016b), Haus and Michini (2017), Suzuki and Minato (2018), Suzuki et al. (2018), Bergman et al. (2021).
Polyhedral Analysis	Behle and Eisenbrand (2007), Tjandraatmadja and van Hoeve (2019).

recursive models in Section 2.2, and discuss basic approximation concepts in Section 2.3. While there are several DD variants in the literature (see, e.g., Wegener 2000), we primarily focus on ordered decision diagrams. That is, layered diagrams where each layer is associated with a single variable. This DD variant is the most commonly used by the discrete optimization community and, as such, most contributions are based on this structure.

2.1. DD Representation of Optimization Problems

In this paper, we consider maximization problems $\mathscr{P}$ of the form

$$\max_{\boldsymbol{x}} \{ f(\boldsymbol{x}) : \boldsymbol{x} \in \mathcal{X} \}. \tag{$\mathscr{P}$}$$

where $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ is a tuple of n variables, $\mathcal{X} \subset \mathbb{Z}^n$ is a finite (integer) feasible space, and $f : \mathbb{Z}^n \to \mathbb{R}$ is the objective function. Moreover, we let $I \equiv \{1, \ldots, n\}$ denote the

Table 2 Paper classification for approximate decision diagrams.

Sub-class	Papers
	Approximate DD Compilation
Top-down and Iterative Refinement	Hadžić et al. (2008a), Bergman et al. (2011), Bergman et al. (2014d), Frohner and Raidl (2019a), Frohner and Raidl (2019b), de Weerdt et al. (2021), Horn et al. (2021).
Other Construction Algorithms	Cire and Hooker (2014), Bergman and Cire (2016c),Bergman and Cire (2017), Römer et al. (2018), Horn et al. (2021), Horn and Raidl (2021).
Variable Ordering	Behle (2008), Bergman et al. (2011), Cappart et al. (2019), Karahalios and van Hoeve (2021), Parjadis et al. (2021).
	DD-based Bound
Dual Bounds	Andersen et al. (2007), Cire and van Hoeve (2013), Kell and Van Hoeve (2013), Bergman et al. (2014c), Hooker (2017), Kinable et al. (2017), van den Bogaerdt and de Weerdt (2018), Maschler and Raidl (2018), Castro et al. (2019), Castro et al. (2020a), Castro et al. (2020b), van Hoeve (2020), van Hoeve (2021).
Lagrangian Bounds	Bergman et al. (2015a), Bergman et al. (2015b), Hooker (2019), Castro et al. (2020a), Tjandraatmadja and van Hoeve (2020), Lange and Swoboda (2021).
Primal Bounds	Kell and Van Hoeve (2013), Bergman et al. (2014d), ONeil and Hoffman (2019), Horn and Raidl (2019).
Branch-and-Bound	Bergman et al. (2014a), Bergman et al. (2016b), González et al. (2020a), González et al. (2020b), Gillard et al. (2021).
	CP Propagation
	Andersen et al. (2007), Hadžić et al. (2008b), Hoda et al. (2010), Hadžić et al. (2009), Cire and van Hoeve (2012), Bergman et al. (2014b), Perez and Régin (2017b), Perez and Régin (2017a), Kinable et al. (2017).

variable index set, and consider that each x_i assumes values in a finite domain $D_i \subset \mathbb{Z}$ where $\mathcal{X} \subseteq D_1 \times \cdots \times D_n$.

DDs are graphical representations of solutions to $\mathscr{P}$, where layers map to variables and arcs map to variable-value assignments (see, e.g., Figure 1). Formally, a DD $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ is a layered-directed acyclic graph with node set $\mathcal{N}$ and arc set $\mathcal{A}$. The node set $\mathcal{N}$ is partitioned into $n+1$ layers $\mathcal{N} = (\mathcal{N}_1, \ldots, \mathcal{N}_{n+1})$. The first and last layers are the singletons $\mathcal{N}_1 = \{\mathbf{r}\}$ and $\mathcal{N}_{n+1} = \{\mathbf{t}\}$, respectively, where $\mathbf{r}$ is the root node and $\mathbf{t}$ is the terminal node. An arc $a = (u, u') \in \mathcal{A}$ has a source node $s(a) = u$ and a target node $t(a) = u'$ in consecutive layers, i.e., $u' \in \mathcal{N}_{i+1}$ whenever $u \in \mathcal{N}_i$ for every $i \in I$.

The solutions in $\mathcal{X}$ are mapped to paths in the graph as follows. The arcs emanating from layer $\mathcal{N}_i$, $i \in I$, are associated with values in the domain of variable x_i. Every arc $a \in \mathcal{A}$ with source $s(a) \in \mathcal{N}_i$ has a value $v_a \in D_i$, and each node $u \in \mathcal{N}_i$ has at most $|D_i|$ emanating arcs, each with a different value. Given an arc-specified $\mathbf{r} - \mathbf{t}$ path $p = (a_1, \ldots, a_n)$ with $s(a_1) = \mathbf{r}$ and $t(a_n) = \mathbf{t}$, we let $\boldsymbol{x}^p = (v_{a_1}, v_{a_2}, \ldots, v_{a_n}) \in D_1 \times \cdots \times D_n$ be the n-dimensional

point encoded by path p. Then, the solutions represented by the DD is

$$\text{Sol}(\mathcal{D}) \equiv \bigcup_{p \in \mathcal{P}} \{\boldsymbol{x}^p\},$$

where $\mathcal{P}$ is the set of all $\mathbf{r}-\mathbf{t}$ paths in $\mathcal{D}$. Finally, each arc $a \in \mathcal{A}$ is also associated with an arc length ℓ_a. The length encodes, e.g., the contribution to the objective function of paths crossing that arc. We denote the length of a path $p \in \mathcal{P}$ by $\ell(p) \equiv \sum_{a \in p} \ell_a$.

We say that $\mathcal{D}$ exactly represents $\mathcal{X}$ if $\text{Sol}(\mathcal{D}) = \mathcal{X}$ and $f(\boldsymbol{x}^p) = \ell(p)$ for every $p \in \mathcal{P}$, i.e., there is a one-to-one correspondence between points in $\mathcal{X}$ and paths in $\mathcal{P}$, and path lengths evaluate to the objective value of their associated solution.

EXAMPLE 1. Consider a knapsack problem with feasible set $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^4 : 7x_1 + 5x_2 + 4x_3 + x_4 \leq 8\}$ with weight vector $\boldsymbol{w} = (7,5,4,1)$ and a linear objective $f(\boldsymbol{x}) = \boldsymbol{c}^\top \boldsymbol{x}$ with cost vector $\boldsymbol{c} = (4,2,5,1)$. Figure 1 illustrates the set of feasible solutions in $\mathcal{X}$ over two graphs. Dashed arrows represent arcs associated with a zero-value variable assignment (i.e., $v_a = 0$) and solid arrows correspond to arcs with a one-value variable assignments (i.e., $v_a = 1$). The lengths of solid and dashed arcs in layer i are c_i and 0, respectively.

The left graph in Figure 1 corresponds to a search tree (e.g., in a branch-and-bound procedure) where paths from the root $\mathbf{r}$ to each leaf node correspond to feasible variable assignments. The right graph illustrates a DD for $\mathcal{X}$. Each DD layer is associated with a variable and arcs denote feasible variable-value assignments. Note that there is a one-to-one correspondence between root-to-leaf paths in the search tree and $\mathbf{r}-\mathbf{t}$ paths in the DD, i.e., the DD exactly represents $\mathcal{X}$. Moreover, by construction, each path length evaluates to the objective value of the corresponding solution. Note that any longest path, here defined by $\boldsymbol{x}^* = (0,0,1,1)$ with a length of 6, is an optimal solution to the problem. ■

The DD variant above is the most commonly used by the discrete optimization community due to its simpler layer-variable structure. The original DD definition for Boolean functions (Bryant 1986, 1992) differs in that: (a) nodes are directly associated with variables (i.e., literals) without the layered constraint, (b) arcs may traverse multiple layers, and (c) the DD has two terminal nodes, one for feasible and one for infeasible solutions. While some researchers employ the classic DD definition to create smaller diagrams for specific applications at the cost of more intricate implementations (Perez and Régin 2014), most works in optimization adopt the restricted DD structure above. A second notable DD

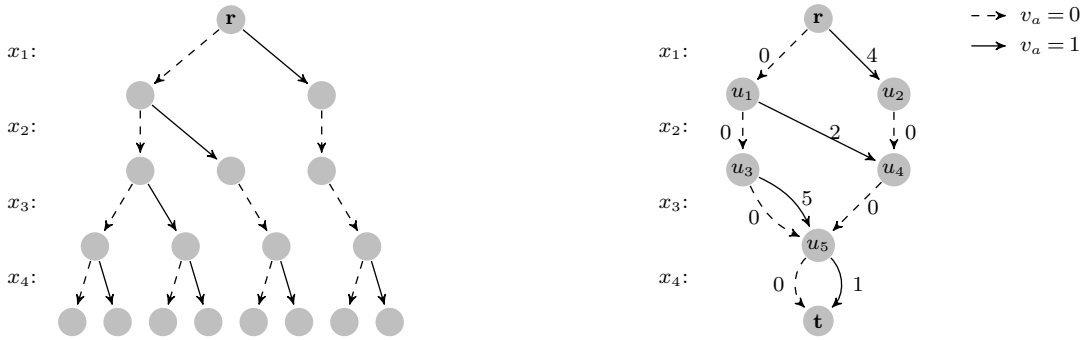

Figure 1 **A decision tree and a DD for** $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^4 : 7x_1 + 5x_2 + 4x_3 + x_4 \leq 8\}$.

variant is a zero-suppressed decision diagram (ZDD), which can compactly represent combinatorial sets and has been used, for instance, in column generation algorithms (Morrison et al. 2016, Kowalczyk and Leus 2018) and enumeration procedures (Suzuki and Minato 2018). We refer the reader to the work of Minato (1993) for further details on ZDDs.

2.2. Decision Diagrams and Recursive Formulations

A common practice in the literature is to conceptualize DDs based on a recursive formulation (RF) of $\mathscr{P}$. As investigated by Hooker (2013), there is a strong relationship between DDs and the state transition graph in the dynamic programming (DP) literature. DPs represent optimization problems via a recursive model where decisions are made sequentially (Bertsekas 2017). The state transition graph can be obtained by unfolding the recursive model for any possible state in the system, where nodes map to states, arcs represent the state-transition function, and arc lengths encode the immediate rewards.

To detail this relationship, we start by introducing the general form of a recursive model using syntax from the DP literature (Bertsekas 2017). In particular, such models are defined in terms of stages, where transitions across stages are driven by actions (or controls). We consider an $n+1$ stage system with actions $\boldsymbol{x} \in \mathbb{Z}^n$, one per stage $i \in I$. For a given stage $i \in \{1, \ldots, n+1\}$, state variables $\boldsymbol{S} \in \mathcal{S}_i$ represent a summary of past actions that reach that state, where the state space $\mathcal{S}_i$ denotes the set of states reachable at the i-th stage. The set of feasible actions x_i at a state $\boldsymbol{S} \in \mathcal{S}_i$ is given by the action set $X_i(\boldsymbol{S})$.

The system transitions to a new state according to the current state and action that has been applied. Transitions are governed by the transition function $\phi_i : \mathcal{S}_i \times X_i \to \mathcal{S}_{i+1}$, which maps the current state-action pair to a state in the next stage. Moreover, each state $\boldsymbol{S} \in \mathcal{S}_i$ and decision variable $x \in X_i(\boldsymbol{S})$ at stage $i \in \{1, \ldots, n+1\}$ incurs an immediate reward $g_i(\boldsymbol{S}, x)$.

The optimal actions of a DP model solve the Bellman equation

$$V_i(\boldsymbol{S}) = \max_{x \in X_i(\boldsymbol{S})} \{g_i(\boldsymbol{S}, x) + V_{i+1}(\phi_i(\boldsymbol{S}, x))\}, \qquad \forall i \in I, \tag{RF}$$

where $V_i(\cdot)$ is the value function with $V_{n+1}(\boldsymbol{S}) = 0$ for any $\boldsymbol{S} \in \mathcal{S}_{n+1}$. We assume the initial state is the singleton $\mathcal{S}_1 = \{\boldsymbol{S}_1\}$, where $\boldsymbol{S}_1$ is the root state.

A recursive model is a reformulation of problem $\mathscr{P}$ if

(a) There is an one-to-one mapping between solutions $x \in \mathcal{X}$ and a state-action trajectory $(\boldsymbol{S}_1, x_1), (\boldsymbol{S}_2, x_2), \ldots, (\boldsymbol{S}_n, x_n)$, where $\boldsymbol{S}_{i+1} = \phi_i(\boldsymbol{S}_i, x_i)$.

(b) For every solution $\boldsymbol{x} \in \mathcal{X}$ and associated state trajectory $(\boldsymbol{S}_1, x_1), (\boldsymbol{S}_2, x_2), \ldots, (\boldsymbol{S}_n, x_n)$, we have $f(\boldsymbol{x}) = \sum_{i=1}^{n} g_i(\boldsymbol{S}, x_i)$, i.e., the solution value matches the sum of rewards.

EXAMPLE 2. We depict the classical recursive model of the knapsack problem introduced in Example 1. The actions are $\boldsymbol{x} \in \{0,1\}^4$ and the state $\boldsymbol{S} = Q \subseteq \mathbb{Z}$ captures the load of the knapsack at each stage. For an initial state $Q_1 = 0$, the Bellman equation is

$$V_i(Q) = \max_{x \in X_i(Q)} \{c_i x + V_{i+1}(Q + w_i x)\}, \qquad \forall i \in \{1, \ldots, 5\}. \tag{R-KNP}$$

The transition function $\phi_i(Q, x) = Q + w_i x$ updates the load of the knapsack, while the immediate reward $g_i(Q, x) = c_i x$ corresponds to the gain from choosing item i. Since the weight of each item is positive, the action space for state $Q \in \mathcal{S}_i$ is $X_i(Q) = \{x \in \{0,1\} : Q + w_i x \leq 8\}$, for all stages $i \in \{1, \ldots, 4\}$. ■

A DD can be obtained immediately from the state-transition graph of RF (Bergman et al. 2011). Specifically, with each state $\boldsymbol{S} \in \mathcal{S}_i$ we associate a node $u_{\boldsymbol{S}} \in \mathcal{N}_i$, $i \in I \cup \{n+1\}$; note that the root node $\mathbf{r}$ corresponds to the initial state $\boldsymbol{S}_1$. There exists an arc a with value $v_a = x$ between nodes $u_{\boldsymbol{S}_i} \in \mathcal{N}_i$ and $u_{\boldsymbol{S}_{i+1}} \in \mathcal{N}_{i+1}$, $i \in I$, if and only if $\boldsymbol{S}_{i+1} = \phi_i(\boldsymbol{S}_i, x)$. The length of such an arc is the immediate reward of the transition, i.e., $\ell_a = g_i(\boldsymbol{S}_i, x)$. Finally, the nodes in the last layer, in case several terminal states are present, are merged into the single terminal node $\mathbf{t}$.

2.3. DD Approximations

While the size of the DD can be considerably reduced using the classical reduction procedure by Bryant (1992) (i.e., by merging isomorphic subgraphs), an exact DD $\mathcal{D}$ for $\mathcal{X}$ is, in general, exponentially large in the number of variables n. An alternative to overcome

this limitation is to consider a DD approximation instead, i.e., a DD that either under or over-approximates the set of feasible solutions and/or the objective function.

This idea was first proposed by Andersen et al. (2007), who over-approximates $\mathcal{X}$ using a relaxed DD, i.e., a DD $\mathcal{D}$ with solution set such that $\mathcal{X} \subseteq \text{Sol}(\mathcal{D})$. Thus, every point in $\mathcal{X}$ maps to a $\mathbf{r}-\mathbf{t}$ path in $\mathcal{D}$, but the converse does not necessarily hold. Bergman and Cire (2018), Hooker (2013), and Hooker (2019) also consider variants that relax the objective function value, in that any $\mathbf{r}-\mathbf{t}$ path p satisfies $f(\boldsymbol{x}^p) \leq \ell(p)$. Relaxed DDs provide a discrete relaxation of $\mathcal{X}$ that can be used, for instance, to compute optimistic bounds for an optimization problem and have been key in state-of-the-art DD methodologies.

Conversely, Bergman et al. (2014d) introduce the concept of restricted DDs, i.e., $\mathcal{D}$ under-approximates $\mathcal{X}$ by capturing only a subset of the feasible solutions. Such an approximation can be obtained, for example, by limiting the number of nodes in each layer and heuristically discarding nodes. The main advantage of this technique is to obtain primal bounds for an optimization problem, as we will further discuss in Section 4.2.

EXAMPLE 3. We recall the knapsack instance from Example 1 with feasible set $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^4 : 7x_1 + 5x_2 + 4x_3 + x_4 \leq 8\}$ and recursive model R-KNP. Figure 2 depicts an exact, a relaxed, and a restricted DD for $\mathcal{X}$. The paths of the exact DD $\mathcal{D}_1$ (left diagram) have a one-to-one relationship with the points in $\mathcal{X}$. Moreover, each node in $\mathcal{D}_1$ is associated with a single state of R-KNP, e.g., node u_1 has $Q = 0$ and node u_2 has $Q = 7$.

The restricted DD $\mathcal{D}_2$ (middle) represents only a subset of the solutions in $\mathcal{X}$. For instance, the solution $(1,0,0,1) \in \mathcal{X}$ is not in $\text{Sol}(\mathcal{D}_2)$. This restricted DD can be obtained by discarding the states of R-KNP that have value greater than 6. Lastly, relaxed DD $\mathcal{D}_3$ represents all feasible solutions in $\mathcal{X}$ and some infeasible assignments. Specifically, the shaded $\mathbf{r}-\mathbf{t}$ path in $\mathcal{D}_3$ corresponds to infeasible solution $(1,0,1,1) \notin \mathcal{X}$. ■

When generating decision diagrams from a recursive model RF, each node and arc in an exact DD maps to a state and action, respectively. Thus, the transition and reward functions are well-defined in such settings, as they can be inherited directly from RF. In a relaxed DDs, however, this property does not hold because relaxing a DD modifies the state space by incorporating infeasible paths into the solution set.

Hooker (2017) provides a formal connection between RF and a relaxed DD through the use of *merged states*. More specifically, given two states $\boldsymbol{S}, \boldsymbol{S}' \in \mathcal{S}_i$, we define $\widetilde{\boldsymbol{S}} = \boldsymbol{S} \oplus \boldsymbol{S}'$ as

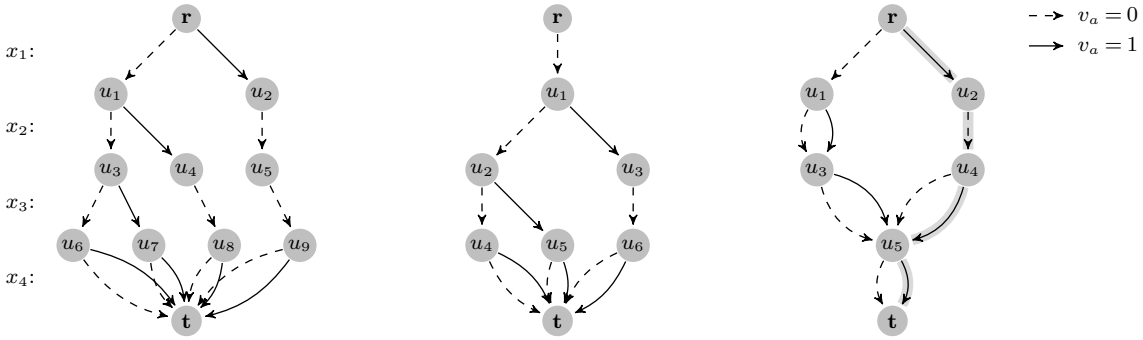

Figure 2 **An exact DD $\mathcal{D}_1$ (left), a restricted DD $\mathcal{D}_2$ (middle), and a relaxed DD $\mathcal{D}_3$ (right) for $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^4 : 7x_1 + 5x_2 + 4x_3 + x_4 \leq 8\}$.**

a merged state where $\oplus$ is an appropriate merging operator, i.e., $\oplus$ satisfies the following properties for any $i \in I$:

(C1) The set of feasible actions over states $\boldsymbol{S}$ and $\boldsymbol{S}'$ are also feasible over $\widetilde{\boldsymbol{S}}$, i.e., $X_i(\boldsymbol{S}), X_i(\boldsymbol{S}') \subseteq X_i(\widetilde{\boldsymbol{S}})$.

(C2) The immediate reward at state $\widetilde{\boldsymbol{S}}$ is greater than or equal to the immediate reward at states $\boldsymbol{S}$ and $\boldsymbol{S}'$. Thus, for any $x \in X_i(\boldsymbol{S})$ and $x' \in X_i(\boldsymbol{S}')$ we have $g_i(\boldsymbol{S}, x) \leq g_i(\widetilde{\boldsymbol{S}}, x)$ and $g_i(\boldsymbol{S}', x') \leq g_i(\widetilde{\boldsymbol{S}}, x')$, respectively.

Following the above conditions, we say that $\widetilde{\boldsymbol{S}}$ relaxes a state $\boldsymbol{S} \in \mathcal{S}_i$ if $X_i(\boldsymbol{S}) \subseteq X_i(\widetilde{\boldsymbol{S}})$ and the immediate reward function over any $x \in X_i(\boldsymbol{S})$ is larger for $\widetilde{\boldsymbol{S}}$, i.e., $g_i(\boldsymbol{S}, x) \leq g_i(\widetilde{\boldsymbol{S}}, x)$. Properties (C1) and (C2) are necessary (but not sufficient) conditions to define a proper relaxation of RF. Hooker (2017) shows that operator $\oplus$ defines a proper relaxation for RF if, in addition to (C1) and (C2), we impose a condition over the transition function:

(C3) If $\widetilde{\boldsymbol{S}}$ relaxes state $\boldsymbol{S} \in \mathcal{S}_i$, then, given any value $x \in X_i(\boldsymbol{S})$, $\phi_i(\widetilde{\boldsymbol{S}}, x)$ relaxes state $\phi_i(\boldsymbol{S}, x)$, for all $i \in I$. Thus, $\widetilde{\boldsymbol{S}}$ defines a relaxed state in the following stage for each feasible action.

The relevance of establishing such a connection is to provide a framework for building relaxed DDs using RF. For example, one could construct a relaxed DD using a top-down approach starting with $\mathbf{r}$ and building one layer at a time. If the number of nodes in a layer (width) is too large, nodes can be merged using a state redefinition that satisfies (C1)-(C3). We provide details in Section 4.1.

EXAMPLE 4. Recall the knapsack instance from Example 1 with feasible set $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^4 : 7x_1 + 5x_2 + 4x_3 + x_4 \leq 8\}$ and recursive model R-KNP. We define the merge operator over states $Q, Q' \in \mathcal{S}_i$ at stage $i \in \{1, \ldots, 4\}$ as the minimum over both quantities, i.e.,

$Q \oplus Q' = \min\{Q, Q'\}$. Operator $\oplus$ satisfies (C1) since any feasible solution for a knapsack load Q or Q' is also feasible for $\min\{Q, Q'\}$. Condition (C3) also holds for $\oplus$ since the transition function is an increasing function over Q (see Example 2). Lastly, (C2) holds since the immediate reward $g_i(Q, x) = c_i x$ is independent of the current state Q. Note that $\mathcal{D}_3$ in Figure 2 is a relaxed DD created with this merging operator by choosing to merge nodes with small state value first. ■

3. Exact Decision Diagrams

Exact DDs encode the set of solutions of a discrete optimization problem as paths over a directed acyclic graph. As shown in Table 1, we distinguish four different purposes observed in the literature for constructing an exact DD: modeling, feasibility checking, solution extraction, and solution-space analysis.

We review each of these purposes and show how to integrate them into integer programming (IP) and constraint programming (CP) solvers. This section focuses on works where one or multiple exact DDs represent either the complete problem or a subset of its constraints, i.e., where no merging operation is applied. We first present different modeling techniques based on DDs. We then review works that use DDs to check feasibility or to extract solutions. The last subsection describes enumeration procedures and post-optimality analysis algorithms over DDs.

3.1. Modeling

One of the most common purposes of DDs is to model complex combinatorial structures. A DD can represent any combinatorial problem by enumerating solutions as paths. This characteristic is particularly appealing for problems that consider constraints that are usually hard to represent with standard methodologies, e.g., non-linear inequalities.

A simple procedure to represent a combinatorial problem as a DD is to enumerate all the solutions in a tree and then merge nodes with equivalent paths to the terminal. This procedure is a naïve approach that is impractical in many applications due to the exponential growth of the solution set with respect to the number of variables. Thus, most works create DDs using algorithms that avoid explicitly enumerating all the solutions.

This section discusses general procedures to encode combinatorial problems into DDs, in particular the recursive formulation strategy presented in Section 2.2. We then review modeling techniques that integrate DDs into IP and CP methodologies. Lastly, we present recent extensions of DDs to model problems with continuous variables.

3.1.1. Recursive Models. Hooker (2013) studied the relationship between DDs and dynamic programming, showing that DDs can be seen as a compact representation of the state-transition graph (see Section 2.2). Thus, we can create a DD by building the state-transition graph and merging nodes representing equivalent partial solutions (e.g., the set of paths from the merged nodes to $\mathbf{t}$ are identical). However, this algorithm can be computationally intractable depending on the size of the state-transition graph.

Bergman et al. (2016b) revisited this idea and presented a general procedure to create DDs based on top-down algorithms to build relaxed DDs (see Section 4.1 for further details). The authors presented recursive models for several classic combinatorial problems (e.g., the maximum independent set) and showed that their procedure can efficiently create exact and relaxed DDs based on such recursive models. Hooker (2017) analyzed the relationship between DDs and recursive models for sequencing problems. The author formalized some of the ideas introduced by Bergman et al. (2016b) to create valid DD relaxations and presented a general framework to define recursive models that are suited for exact and relaxed DDs.

Most papers in this survey use a recursive model to create a DD for two reasons. First, several discrete problems have a natural recursive formulation, e.g., the knapsack problem and many sequencing problems. Second, there exists a wide range of algorithms to construct a DD from a general recursive formulation (Bergman et al. 2016b). Nonetheless, there are specific cases where other mechanisms are more suited to create a DD encoding, e.g., global constraints that represent a list of feasible solutions (Cheng and Yap 2008).

3.1.2. Network Flow Formulation. One of the most appealing characteristics of DDs for the mathematical programming community is their network flow reformulation (Behle 2007). This formulation can be integrated into any IP model by adding new variables and constraints. Moreover, the network flow model is a core component for advanced procedures that combine DDs and IP methodologies, e.g., cutting planes, Benders decomposition, and Lagrangian relaxations.

Given a DD $\mathcal{D} = (\mathcal{N}, \mathcal{A})$, the network flow model $\text{NF}(\mathcal{D})$ uses variables $\boldsymbol{y} \in \mathbb{R}_+^{|\mathcal{A}|}$ to represent the flow traversing each DD arc and the original variables $\boldsymbol{x} \in \mathbb{R}^n$ to limit the flow in each layer. Equalities (1a) and (1b) are balance-of-flow constraints over $\mathcal{D}$. Constraint (1c) links the arcs of $\mathcal{D}$ with solutions $\boldsymbol{x}$.

$$\text{NF}(\mathcal{D}) = \{(\boldsymbol{x}; \boldsymbol{y}) \in \mathbb{R}^n \times \mathbb{R}_+^{|\mathcal{A}|} :$$

$$\sum_{a \in \mathcal{A}^{\text{out}}(u)} y_a - \sum_{a \in \mathcal{A}^{\text{in}}(u)} y_a = 0, \qquad \forall u \in \mathcal{N} \setminus \{\mathbf{r}, \mathbf{t}\}, \tag{1a}$$

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

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

where $\mathcal{A}^{\text{out}}(u)$ and $\mathcal{A}^{\text{in}}(u)$ are the outgoing and incoming arcs at a node u, respectively. Intuitively, the flow over each path $p \in \mathcal{P}$ can be seen as the weight of its corresponding point $\boldsymbol{x}^p$. By restricting the total flow to have value one, the flow variables represent a convex combination of the points in $\text{Sol}(\mathcal{D})$. In particular, the polytope $\text{NF}(\mathcal{D})$ projected over the $\boldsymbol{x}$ variables is equivalent to the convex hull of all solutions represented by $\mathcal{D}$, i.e., $\text{Proj}_{\text{x}}(\text{NF}(\mathcal{D})) = \text{conv}(\mathcal{X})$ (Behle 2007, Tjandraatmadja and van Hoeve 2019). Thus, the network flow model $\text{NF}(\mathcal{D})$ is an ideal linear formulation of the solution set of $\mathcal{D}$. In particular, $\text{NF}(\mathcal{D})$ can be used to create extended linear formulations of complex combinatorial structures as we review in what follows.

There is a strong relationship between DD network flow models and arc flow formulations based on dynamic programming models (Martin 1987, de Lima et al. 2022). As previously mentioned, a DD $\mathcal{D}$ can be seen as a compact representation of a state-space graph of a recursive model. Thus, $\text{NF}(\mathcal{D})$ corresponds to the arc flow formulation for this compact state-space graph. These two lines of research mainly differ on how the graphical structures are constructed and manipulated. For example, DDs have generic reduction algorithms to build exact diagrams with the minimum number of nodes and arcs (Bryant 1992), while graphical manipulations for arc flow formulations are mostly problem specific (de Lima et al. 2022). Moreover, there are methodological differences when creating relaxations of the original feasibility set instead of an exact representation (see Section 4.2 for a further discussion).

We now review notable applications using this reformulation as a modeling tool. Recent works have shown the advantage of using a DD network flow formulation as a modeling mechanism in a wide variety of real-world applications. Cire et al. (2019) tackled a clinical rotation scheduling problem for medical students, creating a DD-based network flow model to represent all feasible schedules coupled with additional constraints to model other problem characteristics. Another notable real-world application is the design of a heat

exchange circuit (Ploskas et al. 2019). The authors created a DD to represent all possible tube configurations and used its network flow formulation in an MILP model.

While these last two works create a single DD to represent the complex combinatorial component of the problems, several works consider multiple DD network flow models together. Bergman and Cire (2016a) first proposed the idea of decomposing a problem using multiple DDs that represents specific aspects of the problem. Their procedure creates a network flow model NF($\mathcal{D}$) for each DD $\mathcal{D}$ where the $\boldsymbol{x}$ variables are common to each model, i.e., (1c) are linking constraints that synchronize the solutions among all DDs. Lozano et al. (2020a) studied the complexity of this multiple network flow model and showed that it is NP-hard in the general case. The authors also proposed a cutting-plane algorithm that solves a maximum flow problem over the DDs to derive cuts and solve the problem more efficiently.

Despite its complexity, the idea of using multiple DDs has been particularly successful in representing non-linear functions as network flow models. Bergman and Cire (2018) employed this procedure to represent non-linear objective functions that admit a recursive formulation. Their technique assumes that the objective function corresponds to the sum of non-linear functions and considers one DD for each function. Bergman and Lozano (2021) presented a similar decomposition for quadratically constrained problems. Their procedure decomposes the matrix of a quadratic constraint as the sum of multiple smaller matrices, where a DD encodes the solution set induced by each sub-matrix. The authors then used a network flow formulation with linking constraints to linearize the quadratic constraint. Lastly, Nadarajah and Cire (2020) created an approximate linear program in the context of DP by employing multiple DD-based network flow models, showing that stronger linear reformulations can be obtained if the (merged) states of the DDs are taken into acount when generating the model.

Recent works also employ network flow models based on multiple DDs to solve challenging applications. Serra et al. (2019) considered a problem that assigns train trips and commuter vehicles to an uncertain set of passengers to minimize the number of commuter trips and total traveling time. Their approach considers a DD for each possible destination and scenario to model the set of passengers associated with a commuter vehicle. Bergman et al. (2019) addressed the bin packing problem with minimum color fragmentation by

building one DD for each bin and creating an IP formulation that links the solutions represented in each DD network flow model. Since all bins are identical, the authors also proposed an IP formulation that uses a single DD network flow model with some modifications to consider multiple bins (Mehrani et al. 2021). Hosseininasab and van Hoeve (2021) used a similar strategy to tackle the multiple sequence alignment problem with a DD flow model to represent all pairwise sequence alignments and linking constraints to synchronize the DD solutions. The problem is then solved using a Logic-Based Benders Decomposition (LBBD), where the master problem corresponds to the DD flow models with linking constraints and the sub-problems enforce additional constraints over the chosen alignments. Lastly, Lozano et al. (2020b) employed multiple DDs for the paired job scheduling problem, where each DD represents the specific constraints for a job. The authors proposed a lifted reformulation based on the network flow model where the flow variables are projected out. The main advantage of their lifted reformulation is that it can be constructed without explicitly constructing the DDs.

While the aforementioned works employ the network flow model of Behle (2007), other authors have presented variants for DDs that encode stochastic constraints. Latour et al. (2017, 2019) represented probabilistic constraints with DDs where the parameters follow a probability distribution that depends on the decision variables. The authors used the DD to model the probability of each constraint by encoding the decision variables and the stochastic parameters within the DD. The DD is reformulated into a quadratic constraint model that is linearized and introduced into a MILP formulation of the problem.

Haus et al. (2017) also considered a variant of the network flow model for a class of two-stage stochastic programs. Their problem considers endogenous uncertainty, i.e., the first stage decision influence the stochastic process. Their proposed procedure aggregates scenarios with equal cost using multiple DDs and relates these scenarios with the first stage decisions. Similar to Latour et al. (2017), each DD computes the probability of achieving a cost value using a MILP reformulation.

We note that most of the works described here create a DD network flow model to represent a subset of constraints that are generally hard to encode with linear inequalities. For example, several works model the sequencing aspect of the problem using a DD (Cire et al. 2019, Ploskas et al. 2019, Serra et al. 2019, Hosseininasab and van Hoeve 2021) since other alternatives would involve big-M constraints that have a loose linear relaxation.

Alternatively, some authors employ DDs as linearization tools for non-linear expressions (Bergman and Cire 2018, Bergman and Lozano 2021) and probabilistic structures (Latour et al. 2017, 2019, Haus et al. 2017). Thus, a DD network flow formulation is most beneficial when standard procedures lead to poor relaxations and the DD encoding is small.

3.1.3. Global Constraints. In contrast to IP technologies, CP solvers represent combinatorial problems using global constraints, i.e., general-form constraints that encode sub-structures of the problem (Rossi et al. 2006). A global constraint has an inference procedure that retrieves information about feasible variable assignments and a propagation mechanism to update the set of feasible solutions inside the constraint. From this perspective, a DD that represents a set of solutions is a global constraint where the inference procedure checks the arc values to determine the current domain of the variables and the propagation procedure removes arcs to update the feasibility set.

Andersen et al. (2007) introduced a general framework for exact and relaxed DDs in CP solvers. Their DD implementation encodes the complete solution set (i.e., the domain store) and propagates the branching decisions. Since the DD could grow exponentially, the authors approximate the solution space using a relaxed DD. Alternatively, we can represent sub-structures of the problem using exact DDs.

One of the main advantages of the DD representation of a global constraint is that it achieves generalized arc consistency (GAC). We say that a variable x is GAC for a constraint $\mathcal{C}$ if for every value of its domain there exists a feasible solution with respect to $\mathcal{C}$. Since the DD represents all feasible solutions of a global constraint $\mathcal{C}$, we can check in polynomial time if a variable assignment is part of a feasible solution or not (Andersen et al. 2007). Another important characteristic is that CP solvers construct a DD only once and can efficiently update the graph to eliminate infeasible assignments according to other constraints or branching decisions. We note that these two properties do not hold for relaxed DDs and, thus, it is necessary to build sophisticated procedures to efficiently integrate relaxed DD with CP technologies when exact representations are too large.

Several researchers have represented different global constraints with DDs and even precompile sub-problem structures into DDs to enhance propagation (de Uña et al. 2019). This line of research has inspired new DD variants that are suitable for CP technologies, such as non-deterministic DDs (Amilhastre et al. 2014). In the following, we review several

works that represent global constraints using exact DDs. Section 4.3 discusses encoding global constraints with relaxed DDs.

One of the main applications of DDs in the CP community is to encode global constraints that explicitly enumerate the set of solutions. Cheng and Yap (2008) first proposed to model `Table` constraints (i.e., a list of feasible solutions) using an exact DD. The authors presented an algorithm to convert a `Table` constraint into a DD by first representing the set of solutions in a tree and then merging identical sub-trees to obtain a reduced DD. Their DD construction procedure was later improved by Cheng and Yap (2010) and generalized to consider negative `Table` constraints (i.e., a list of infeasible solutions).

These preliminary works became the stepping stone for DD-based `Table` constraints and similar structures. Perez and Régin (2014) presented a new algorithm for DD inference over `Table` constraints that shows superior run-time performance compared to existing methodologies. The same authors also introduced novel algorithms to construct DDs from specialized `Table` constraints (Perez and Régin 2015b). Moreover, the authors presented improved procedures to manipulate DDs, e.g., algorithms to reduce a DD or to add/remove solutions (Perez and Régin 2015a, 2016), and introduced a parallelization strategy for these algorithms (Perez and Régin 2018). Lastly, Perez and Régin (2017c) studied DD-based propagation for linear cost functions. Their approach improves the cost-based propagator of a DD (i.e., propagation of a linear cost function) and introduces a DD propagator for soft constraints (i.e., constraints that allow infeasible solutions with a cost penalty).

Verhaeghe et al. (2018) also studied how to efficiently encode `Table` constraints into DDs. Their work proposes a related data structure called semi-DD (or sDD) where the middle layer is non-deterministic. Intuitively, an sDD is a DD obtained by connecting the leaf nodes of two trees representing partial solutions for half of the variables. The main advantage of this structure is that the maximum number of nodes in each layer can be exponentially smaller than a standard DD. The authors introduced several mechanisms to construct and manipulate an sDD, including a reduction procedure and an algorithm to remove solutions. In a related work, Verhaeghe et al. (2019) proposed a new DD variant to model `Smart-Table` constraints. Their basic smart DDs (or bs-DDs) can represent unary constraints (e.g., $x_1 \geq 1$) over arcs to compactly encode sets of feasible values. Therefore, bs-DDs avoid having multiple arcs with the same source and target node. Vion and Piechowiak (2018) also proposed an alternative DD structure for `Table` constraints. Their

work presents a procedure to transform any MDD to special type of BDD where propagation and filtering procedure can be performed more efficiently. In particular, their work adapts filtering procedures proposed for MDDs and other tree structures and shows better results in terms of theoretical complexity bounds and empirical solution times.

Researchers have also employed exact and relaxed DDs to build novel global constraints. Roy et al. (2016) introduced a new global constraint based on DDs that models binary relations over sequences of temporal events. Their empirical results show that their DD-based global constraint is superior to a classical scheduling representation of the same temporal relations. We refer the reader to Section 4.3 for further examples of relaxed DD encodings of global constraints.

3.1.4. Continuous Variables. So far we have discussed how to model combinatorial problem using DDs. While DDs can represent any bounded set with integer points, until recently there was no DD representation of problems with continuous variables.

Davarnia (2021) first considered modeling a mixed-integer set $\mathcal{X} \in \mathbb{R}^n$ with a DD to build outer-approximations. This work shows that it is possible to construct a DD $\mathcal{D}$ such that $\text{conv}(\mathcal{X}) \subseteq \text{conv}(\text{Sol}(\mathcal{D}))$ where $\mathcal{D}$ has a finite number of arcs. The main result relies on the fact that it is sufficient to have at most two arcs between consecutive nodes u and v (i.e., $u \in \mathcal{N}_i$ and $v \in \mathcal{N}_{i+1}$ for some $i \in I$), where the arc labels take the minimum and maximum feasible value of their associated variable x. This type of DD is called an *arc-reduced DD* since it omits arcs with labels other than the minimum and maximum domain values. Therefore, any variable $x \in [a, b]$ can be represented in an arc-reduced DD with at most two arcs emanating from nodes in the layer associated with x.

Davarnia (2021) shows how to build arc-reduced DDs that over-approximate $\text{conv}(\mathcal{X})$ for any $\mathcal{X} \in \mathbb{R}^n$, but there are no guarantees on the tightness of such approximations. Salemi and Davarnia (2021) studied this problem and identified necessary and sufficient conditions to exactly represent $\text{conv}(\mathcal{X})$ with an exact DD. In particular, their paper states that a set $\mathcal{X} \in \mathbb{R}^n$ is *DD-representable* (i.e., there exists a DD $\mathcal{D}$ such that $\text{conv}(\mathcal{X}) = \text{conv}(\text{Sol}(\mathcal{D}))$) if and only if $\mathcal{X}$ admits a rectangular decomposition (we refer to the original paper for a formal definition). While determining if $\mathcal{X}$ admits a rectangular decomposition can be challenging, the paper points out that any compact integer set $\mathcal{X} \in \mathbb{Z}^n$ fulfills this property. Moreover, the authors showed that any compact mixed-integer set $\mathcal{X} \in \mathbb{R}^n$ with a single continuous variable is DD-representable. This last result is particularly important for decomposition algorithms, as we discuss in Section 3.2.

3.2. Feasibility Checking

Another important use of exact DDs is checking the feasibility of a solution. This general idea attracted the attention of researchers in discrete optimization since there are several feasibility checking procedures for IP and CP technologies. The main advantage of using exact DDs for feasibility checking is that we construct the DD only once and repeatedly call it for each new candidate solution.

While feasibility checking is most commonly used in conjunction with IP and CP solvers, two recent works employ this idea without these technologies. In both applications, the solution generation procedure first solves a relatively easier problem to generate solutions fast, and the DD encodes the set of constraints that are hard to represent. Nishino et al. (2015) used a DD for feasibility checking to solve the constrained shortest path problem. They employed Dijkstra's algorithm (Dijkstra et al. 1959) to solve a shortest path problem but in each step a DD checks the feasibility with respect to the additional constraints. The second work generates solutions for a traveling salesman problem (TSP) with preferences using a generative adversarial neural network (GAN) (Xue and van Hoeve 2019). Since the GAN solutions might violate some of the TSP constraints, the authors used a DD to identify infeasible candidate solutions and guarantee feasibility.

The following subsections review feasibility checking algorithms in the IP and CP community where DDs are suitable alternatives to other methodologies. As with the procedures described above, the DD can compactly encode the set of hard constraints for the problem and check feasibility in polynomial time with respect to its size. Thus, these techniques are suited for combinatorial structures that lead to small DDs or where alternative procedures (e.g., an IP model) take exponential time to check feasibility.

3.2.1. Cutting Planes. Cut generation is a mathematical programming procedure to separate infeasible points from the solution set by generating inequalities that are valid for the problem but violated by the infeasible points. This separation problem is NP-hard in the general case, so most cutting plane procedures focus on specific problem structures or use a relaxation of the original problem to generate valid cuts (Cornuéjols 2008). In this context, DDs are an attractive method to solve the separation problem since they provide compact representations of several combinatorial problems, and their network flow formulation lead to cut generation linear programs (CGLP).

We now describe a simple procedure to separate points for a discrete set $\mathcal{X} \subseteq \mathbb{Z}^n$ using a DD $\mathcal{D}$ that exactly represents $\mathcal{X}$ (Becker et al. 2005). Given a point $\boldsymbol{x}' \in \mathbb{R}^n$, we can identify if $\boldsymbol{x}' \notin \text{conv}(\mathcal{X})$ if there exists a vector $\boldsymbol{\lambda} \in \mathbb{R}^n$ such that $\boldsymbol{\lambda}^\top \boldsymbol{x}' > \lambda^*$, where $\lambda^* = \max\{\boldsymbol{\lambda}^\top \boldsymbol{x} : \ \boldsymbol{x} \in \text{conv}(\text{Sol}(\mathcal{D}))\}$. The main advantage of this separation problem is that computing λ^* can be done in polynomial time (w.r.t. to DD size) using a longest-path procedure over the DD with $\boldsymbol{\lambda}$ coefficients. Moreover, the resulting cut $\boldsymbol{\lambda}^\top \boldsymbol{x} \leq \lambda^*$ is valid for every $\boldsymbol{x} \in \text{conv}(\mathcal{X}) = \text{conv}(\text{Sol}(\mathcal{D}))$.

Becker et al. (2005) first studied this DD-based separation problem and proposed a procedure to find vector $\boldsymbol{\lambda}$ that maximizes distance $|\boldsymbol{\lambda}^\top \boldsymbol{x}' - \lambda^*|$ for any $\boldsymbol{x}' \notin \text{conv}(\mathcal{X})$. The authors used an iterative sub-gradient procedure that computes λ^* using a longest-path over $\mathcal{D}$ and updates $\boldsymbol{\lambda}$ with the longest-path solution. Davarnia and van Hoeve (2021) studied this procedure to create outer-approximations for non-linear problems and enhanced the sub-gradient routine by adding normalization constraints to improve convergence.

This simple DD-based separation algorithm can also be reformulated as a CGLP using the dual of the network flow model $\text{NF}(\mathcal{D})$ (Behle 2007). While this CGLP might be computationally slower than a sub-gradient routine (Davarnia and van Hoeve 2021), it is easier to implement and can be modified to obtained structural properties for the resulting cuts. Specifically, two recent works develop novel DD-based CGLPs based on Behle (2007) CGLP. Tjandraatmadja and van Hoeve (2019) proposed a DD-based CGLP to generate target cuts based on the polar set of $\text{NF}(\mathcal{D})$. Their CGLP can compute facet-defining inequalities by just considering a few additional variables and constraints. In contrast, Castro et al. (2021) introduced a reformulation of $\text{NF}(\mathcal{D})$ for binary sets and created a CGLP that resembles a maximum flow problem. The authors also proposed a combinatorial approach to generate cuts that solves a max-flow problem over the DD. Their procedure is orders of magnitude faster than DD-based CGLPs and the sub-gradient routines but might not cut all infeasible fractional points $\boldsymbol{x}' \notin \text{conv}(\text{Sol}(\mathcal{D}))$.

In addition to the previously mentioned approaches, Behle (2007) proposed two types of DD-based logical cuts for branch-and-cut routines. The first one returns an exclusion cut when the DD is infeasible given the current branching decisions (i.e., the DD has no $\mathbf{r}-\mathbf{t}$ paths/feasible solutions due to branching decisions). The second one is an implication cut that is returned when the values of some variables are fixed after updating the DD with the branching decisions.

Cutting-plane procedures are usually coupled with constraint enhancement routines so that the resulting cut removes a larger portion of the fractional space. However, there is currently a limited number of DD-based algorithms to enhance valid inequalities. Becker et al. (2005) introduced a simple procedure to modify the coefficients of a valid inequality to increase its face dimension. However, the resulting inequality might not separate the set of points that the original constraint does. Behle (2007) proposed a DD-based lifting procedure for cover cut inequalities based on classic techniques (Wolsey and Nemhauser 1999) in which the coefficient sub-problem is solved using a DD instead of an integer or linear program. Recently, Castro et al. (2021) introduced a general lifting procedure for combinatorial problems where a DD iteratively tilts valid inequalities in order to increase their dimension. Their procedure resembles the simple approach by Becker et al. (2005) but the lifted inequality is guaranteed to dominate the starting inequality.

As with other DD procedures reviewed so far, these cutting plane techniques leverage the fact that the DD is built only once and can be used to find optimal solutions with different objective functions. For example, the sub-gradient routines use this property to create new sub-gradients based on the DD optimal solutions. The same is true for the CGLP models and the inequality enhancement procedures. We note that these cutting plane techniques are also valid if we replace the exact DD with a relaxed DD (Tjandraatmadja and van Hoeve 2019, Castro et al. 2021, Davarnia and van Hoeve 2021). In this case, the generated inequalities are valid for the original problem but the cutting plane procedure will not separate all infeasible points.

3.2.2. Benders Decomposition. Benders decomposition (Benders 1962) is another IP methodology that has benefited from DDs. The overall idea is to decompose the problem into a master problem and one or more sub-problems, where each sub-problem is represented by a DD. Each DD $\mathcal{D}$ can then be used to generate logic-based cuts or traditional Benders cuts with the help of the network flow model NF($\mathcal{D}$). Bergman and Lozano (2021) first studied this approach for quadratically constrained integer problems. Their procedure decomposes the quadratic matrix into multiple smaller matrices where a DD can easily represent the solution set of each component matrix. Their work proposes a Benders decomposition scheme where each sub-problem solves the shortest path problem over its corresponding DD to generate a Benders cut.

Similar ideas have also been used to solve stochastic programming problems. Lozano and Smith (2018) introduced a DD-based Benders decomposition for a family of two-stage binary stochastic programming problems and applied them to a stochastic TSP variant. Their approach uses a DD to represent the sub-problem (i.e., one for each scenario) and solves a max-flow problem over the DD to create Benders cuts. Guo et al. (2021) applied this same procedure to tackle a stochastic distributed operation room scheduling problem.

In these applications the DD represents a combinatorial problem that might take a considerable time for an IP model to solve. Since the sub-problems of these Benders decompositions are relatively small (e.g., 20 to 40 cities for the traveling salesman problem), the resulting DD is small enough to store in memory and can solve the sub-problems in fractions of a second. Since the set of solutions in the DD remains the same, the overhead of constructing the DD is negligible if we consider that we need to solve the sub-problem thousands of times. Moreover, the DD network flow model is a convex reformulation of the original discrete non-convex sub-problem. Thus, this model can be used to generate LP-based Benders cuts, which is impossible for the original discrete sub-problem.

Alternatively, Salemi and Davarnia (2021) proposed a DD-based Benders decomposition where the master problem is given by a DD and the sub-problems are, for example, linear programs. Their procedure relies on the fact that mixed-integer programming models with a single continuous variable are DD-representable. Thus, we can create a DD for the master problem with the integer variables of the original problem and a single continuous variable to approximate the cost of the sub-problems. Their decomposition, however, needs to update the master problem DD for each sub-problem cut, which can be computationally expensive. To mitigate the potential exponential growth of the exact DD, the authors proposed an iterative DD-based Benders approach that uses relaxed and restricted DDs.

3.2.3. Inference. The CP community has also considered DDs for feasibility checking to infer no-good assignments, i.e., an infeasible set of variable assignments (Schiex and Verfaillie 1994). CP solvers employ no-goods to avoid exploring portions of the solution space that it has already shown to be infeasible.

Subbarayan (2008) first proposed the idea of using DDs to infer no-good assignments in CP solvers. The author showed that the problem of finding a minimum no-good over a DD is NP-hard and proposed a heuristic procedure that traverses the DD to find small sets of no-good assignments. Gange et al. (2011) revisited this idea and presented an incremental

algorithm to find no-goods that searches just a portion of the DD in the vicinity of the last set of arcs removed from a DD. Gange et al. (2013) extended this no-good algorithm for cost-based reasoning, i.e., to identify assignments that lead to sub-optimal solutions. A similar idea was implemented for SAT solvers (Kell et al. 2015) where a DD represents a subset of the constraints (i.e., clauses) and identifies no-goods for clause generation.

We note that no-good inference is related to the cutting plane procedures in IP technologies. A no-good set can be encoded as a linear constraint and added into an IP solver in a branch-and-cut procedure. In fact, the DD-based logic constraints introduce by Behle (2007) are a particular type of no-good that a DD infers during the search. Thus, these advanced technologies for no-good inference in the CP literature can also benefit IP solvers.

Alternatively, DDs can be used to efficiently check if one or more solutions satisfy a global constraint. Jung and Régin (2021) proposed an inference procedure for this task that relies on a novel DD inclusion operator. Their approach creates two DDs, one for the solution set and one for the constraint, and then employs their inclusion operator to check if the set of solutions satisfies the constraint. The authors showed that their inclusion operator is more efficient than a DD intersection operator when applied to some well-know global constraints, e.g., the `Sequence` constraint and `GCC` (global cardinality constraint).

3.3. Solution Extraction

In contrast to feasibility checking, we can use a DD to generate candidate solutions for a given problem. Since DDs represent all the solutions of a combinatorial problem, we can easily extract solutions by choosing any path in the DD. In particular, we can use this property to extract solutions with certain structure or solutions that are optimal for a specific linear objective function.

Hadžić et al. (2004) first explored this idea for a manufacturing problem. The authors created a DD to represent all product configurations and proposed a polynomial-time algorithm to extract solutions that satisfy a set of configuration requirements specified by a user. Their algorithm ignores all arcs that represent infeasible configurations and returns a sub-diagram without the ignored arcs. While the authors tested their procedure in a manufacturing problem, the same idea can be applied for any other combinatorial problem to extract solutions with user-specified variable assignments.

Solution extraction has the advantage that we can decompose the problem into two parts. The first part, encoded with a DD, represents a relaxation of the original problem

to create candidate solutions. The second part of the problem checks the feasibility of the extracted solution and gives feedback to the DD to extract new solutions. This mechanism is a generalization of the column generation procedure and, thus, it can be applied with other technologies. We also point out that this decomposition scheme is suited for problems where the solution extraction and feasibility checking are relatively easy to solve and a DD compactly represents the set of solutions.

3.3.1. Column Generation. Solution extraction has been mostly used as a sub-routine of the column generation procedures in IP/LP technologies. Morrison et al. (2016) proposed a general scheme that applies DDs in a branch-and-price algorithm (Barnhart et al. 1998). The DD solves the pricing problem of the column generation sub-routine, i.e., the DD returns a promising solution and adds a new column to the master problem. The algorithm then separates the last queried solution from the DD to avoid duplicated solutions. Their implementation also considers a branching rule over the original variables, which can be easily integrated into the DD by ignoring arcs with specific values.

There are three advantages of the procedure by Morrison et al. (2016). First, the DD can solve the pricing problem in polynomial time (with respect to its size) for any linear objective function. Thus, their implementation constructs the DD only once and can solve the pricing problem multiple times. Second, the DD can return all optimal solutions for the pricing problem and, thus, add multiple columns in each iteration. Third, branching decisions can be easily included to the DD without changing the complexity of the pricing problem. Therefore, this procedure can be a better alternative to other methodologies when there is a compact DD representation of the pricing problem.

Morrison et al. (2016) tested their procedure on the graph coloring problem. Kowalczyk and Leus (2018) implemented the DD-based branch-and-price procedure for a parallel machine scheduling problems. Raghunathan et al. (2018) applied a similar branch-and-price approach to solve a transportation scheduling problem. Their application has multiple pricing problems, each one modeling the tours of an origin-destination pair using a DD. Lastly, Riascos-Álvarez et al. (2020) employed the DD-based branch-and-price technique to solve a kidney exchange problem where multiple DDs represent the set of possible chain donations of different sizes.

3.4. Solution-Space Analysis

DDs provide a compact representation of the solution space, which is also useful if we want to enumerate and analyze the set of solutions. For example, we can analyze the set of (near) optimal solutions or study the convex hull of the solutions of a DD. We now review procedures to analyze the solutions encoded in a DD and to enumerate solutions with specific characteristics.

3.4.1. Post Optimality Analysis. Hadžić and Hooker (2006) studied this topic for discrete optimization problems and introduced three procedures over an exact DD. The first one is a cost-based domain analysis that identifies the set of variable assignments that are present in at least one near-optimal solution. The second is a conditional cost-based domain analysis, which restricts a subset of variables to take specific values and then performs cost-based domain analysis over the subset of solutions. Lastly, their frequency analysis computes the percentage of solutions with a particular variable assignment.

Since representing the set of solutions using a DD is intractable for most combinatorial problems, Hadžić and Hooker (2007) introduced the concept of sound DDs for post-optimality analysis. A DD is sound for a specific post-optimality analysis if it yields the same results that an exact DD would. Thus, a sound DD might represent a larger solution set than an exact DD but preserves certain properties to correctly perform the analysis. The authors presented a pruning and contraction procedure to create sound DDs for cost-based domain analysis. While the resulting sound DD is significantly smaller than an exact DD, their procedure requires a starting DD that is either exact or represents all feasible solutions within a cost range. Thus, creating sound DDs with this procedure is intractable for large problems.

Recently, Serra and Hooker (2020) revisited the idea of sound DDs for post-optimality analysis and provided new insights. The authors presented several theoretical results related to sound DDs, including a sound-reduction algorithm that constructs the smallest sound DD. They also introduced a general construction procedure for ILP models that creates a sound DD from a branching tree. Their experiments over a wide range of ILP benchmarks show that the resulting sound DD is a more suitable alternative to post-optimality analysis than, for example, branch-and-bound enumeration.

3.4.2. Solution Enumeration. Recent works have also employed DDs to represent the set of non-dominated solutions for a multi-objective discrete optimization problem, i.e., the Pareto frontier. Bergman and Cire (2016b) first tackled the problem by representing the set of feasible solutions with an exact DD and then enumerating the non-dominated solutions using a multi-criteria shortest path algorithm over the DD. This work was extended by Bergman et al. (2021) where the authors presented three procedures that modify the DD while preserving the set of non-dominated solutions. The authors also proposed a bidirectional multi-criteria shortest path procedure to enumerate the non-dominated solutions, which outperforms the unidirectional approach (Bergman and Cire 2016b). Suzuki and Minato (2018) presented a similar procedure to enumerate non-dominated solutions for the 0-1 multi-objective knapsack problem. Their algorithm encodes all feasible solutions using a ZDD and employs specialized filtering procedures based on the knapsack structure to prune dominated solutions.

All the papers discussed so far enumerate solutions to analyze optimal or near-optimal solutions. Conversely, we can employ the DD structure to obtain insightful information on the combinatorial set and, thus, ignore the objective function. Suzuki et al. (2018) built a DD to compute the strongly connected reliability of a network, i.e., the probability that the network will remain strongly connected when removing one or more edges. The authors showed how to create a DD that enumerates all possible strongly connected sub-networks and how to compute the desired probability by traversing the DD once. Thus, this work employed DDs to get structural properties of the connectivity problem over a network. Haus and Michini (2017) constructed a DD to encode all the members of an independent set to analyze the size of the solution set. The authors showed that the DD representation has polynomial size in the number of variables for packing and set covering problems with specific characteristics. Thus, this work opens the possibility of solution set analysis for problems with tractable DD size.

3.4.3. Polyhedral Analysis. We now review two works that employ DDs for polyhedral analysis, an important area of research in mathematical programming due to its relevance for solving IP problems. Behle and Eisenbrand (2007) introduced a procedure to enumerate vertices and facets of the convex hull of a DD solution set, i.e., $\text{conv}(\text{Sol}(\mathcal{D}))$. Their technique considers a binary solution set in an exact DD, where each path in the DD

corresponds to a 0/1 vertex of the convex hull polytope. The enumeration procedure starts with an initial facet that is rotated over the DD to obtain a new facet.

Tjandraatmadja and van Hoeve (2019) also presented a polyhedral analysis procedure based on DDs to certify the dimension of any inequality that is a face of conv(Sol($\mathcal{D}$)). Their procedure finds a set of affine independent points in the DD that are tight for the face by solving a flow problem over the DD. Their procedure then uses a heuristic approach to generate affine independent points based on the flow values of each arc in the DD. The number of affine independent points gives a lower bound on the dimension of the face.

We note that polyhedron analysis is intractable for general IP models since constructing the convex hull of all the solutions is NP-hard (Wolsey and Nemhauser 1999). DDs provide a more manageable procedure to enumerate all the solutions and, thus, give valuable insights to the polyhedral structure of problems that have a compact DD encoding.

4. Approximate DDs

Relaxed and restricted DDs are limited size DDs that over- and under-approximate the set of feasible solutions, respectively (see Section 2.3 for a formal definition). These graphical structures are desirable when tackling large-scale problems since the size of an exact DD can grow exponentially with the number of variables. Specifically, approximate DDs provide primal and dual bounds for a problem and, thus, can be embedded in a search procedure to find optimal solutions for large problems.

This section presents an overview of existing procedures to construct and employ DD approximations. We distinguish three main research areas related to approximate DDs (see Table 2). The first research area focuses on DD construction procedures to create tight approximations. The second area studies bound computation based on DD approximations and specialized search procedures that employ these bounds. The last research area focuses on propagation procedures for CP solvers based on relaxed DDs.

4.1. Approximate DD Compilation

While several works have shown the advantage of using approximate DDs (Bergman et al. 2016a), the question of how to construct one that provides tight bounds remains open. The size of the DD plays an important role in the quality of the approximation, i.e., bigger diagrams are expected to produce tighter bounds. Nonetheless, two DDs with equal size can provide significantly different bounds.

In the following, we introduce the most commonly used procedures to create approximate DDs and discuss recent DD construction algorithms. These procedures can be applied to a large range of problems, in particular to problems that can be reformulated recursively. Also, note that there is no clear superiority between these construction algorithms but their advantages and disadvantages can be used as a guideline to choose the most appropriate procedure for a specific application.

4.1.1. Top-down and Iterative Refinement. The most common techniques to construct approximate DDs are the top-down and the iterative refinement procedures (Bergman et al. 2016a). The main advantage of both procedures is that they can be applied to any combinatorial problem that has a recursive formulation RF. These procedures create an approximate DD $\mathcal{D}$ of limited size by restricting the maximum number of nodes per layer, i.e., limiting its width $w(\mathcal{D}) = \max_{i \in I}\{|\mathcal{N}_i|\}$. Thus, we can control the size of the DD approximation by changing the maximum width limit $\mathcal{W} \geq 0$.

We first describe the top-down construction procedure (Bergman et al. 2011), which can create exact, relaxed, and restricted DDs by making small changes in the implementation (see Algorithm 1). The algorithms starts with a DD structure with two nodes, the root node and terminal node (lines 2-4). The procedure traverses the layers creating the emanating arcs of each node and the nodes in the following layer (lines 5-12). Note that the procedure creates one node for each reachable state given by RF, thus, it will return an exact DD if the maximum width $\mathcal{W}$ is large enough. If the number of nodes in a layer is larger than the maximum width, the procedure will reduce the number of nodes to create either a relaxed or restricted DD (line 13-14).

In the case of a relaxed DD, the top-down procedure will merge the set of nodes into at most $\mathcal{W}$ nodes to enforce the DD width limit. The $\mathsf{MergeDDNodes}(\mathcal{N}_i)$ procedure utilizes an appropriate merging operator $\oplus$ to guarantee that no feasible solutions are lost and, thus, to construct a valid relaxation for the problem (see Section 2.3 for further details on merging operators). In contrast, the $\mathsf{DiscardDDNodes}(\mathcal{N}_i)$ procedure selects a subset of nodes to eliminate from the diagram. By doing so, the resulting restricted DD represents a sub-set of the feasible solutions of the original problem.

The decisions on which nodes to merge/discard are usually done heuristically based on the state information of the nodes. Since these decisions are crucial to create stronger approximations, several researchers have developed general heuristics that lead to better

Algorithm 1 DD Top-Down Construction

procedure TopDownDD(RF, $\mathcal{W}$)
  Create DD $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ with $n+1$ empty layers
  Create the root and terminal node, i.e., $\mathbf{r} \in \mathcal{N}_1$ and $\mathbf{t} \in \mathcal{N}_{n+1}$
  Assign the initial state to the root node, i.e., $\boldsymbol{S}(\mathbf{r}) = \boldsymbol{S}_1$
  for $i \in \{1, \dots, n-1\}$ **do**
    for $u \in \mathcal{N}_i$ **do**
      Create an arc a emanating from u for each possible value in $X_i(\boldsymbol{S}(u))$
      for $a \in \mathcal{A}^{\text{out}}(u)$ **do**
        if there exists node $u' \in \mathcal{N}_{i+1}$ with $\boldsymbol{S}(u') = \phi_i(\boldsymbol{S}(u), v_a)$ **then**
          Direct arc a to node u', i.e., $t(a) = u'$
        else
          Create u' in $\mathcal{N}_{i+1}$ with $\boldsymbol{S}(u') = \phi_i(\boldsymbol{S}(u), v_a)$ and point arc a to u'
    if $|\mathcal{N}_i| > \mathcal{W}$ **then**
      MergeDDNodes($\mathcal{N}_i$) (relaxed DD) or DiscardDDNodes($\mathcal{N}_i$) (restricted DD)
  for $u \in \mathcal{N}_n$ **do**
    Create an arc a emanating from u for each value in $X_n(\boldsymbol{S}(u))$ with $t(a) = \mathbf{t}$
  return $\mathcal{D}$

approximations than a random node selection. We note that there are several papers on merging procedures for relaxed DDs but that is not the case for restricted DDs and discarding heuristics (Bergman et al. 2014d). This discrepancy is due to the wider exploration of relaxed DDs in the literature compared to restricted DDs.

For the relaxed DD case, Bergman et al. (2011) proposed a simple strategy that merges nodes with respect to their longest/shortest path from the root. Thus, the heuristic avoids merging nodes that can potentially impact the DD bound. A similar idea was then proposed for discarding nodes to avoid removing optimal solutions (Bergman et al. 2014d). Frohner and Raidl (2019b) further studied the Bergman et al. (2011) merging heuristic and proposed tie-breaking rules to merge nodes that have similar relaxed states. Frohner and Raidl (2019a) also presented a binary classifier procedure that chooses a merging heuristic in each DD layer. Their heuristic outperforms the simple heuristics in terms of dual bounds but it can be harder to implement due to the time needed to train the classifier. Recently,

de Weerdt et al. (2021) proposed a novel merging procedure for a single-machine scheduling problem that guarantees building an ϵ-approximation of the problem. Their procedure groups states with values in a certain range that depends on ϵ. While the de Weerdt et al. (2021) merging procedure was designed for a specific scheduling problem, similar ideas could be applied to other problems.

The main advantage of the top-down algorithm is its flexibility and simplicity. The procedure can be used to construct any type of DD (exact, relaxed, or restricted) given a recursive formulation of the problem. Moreover, the algorithm has shown to be computationally efficient in practice, allowing researchers to construct DDs with large widths in fractions of a second (Bergman et al. 2016a). However, one of it main disadvantages its the looseness of the approximation that it produces. The procedure merges/discards nodes considering only the state information of the current nodes, which can lead to poor approximations and, thus, weak bounds. While there exist alternatives that include a look-ahead step, these procedures can be computationally expensive (Horn et al. 2021).

An alternative is to construct a relaxed DD using the iterative refinement procedure (Hadžić et al. 2008a). In contrast to the top-down algorithm, this procedure starts with an initial relaxed DD and iteratively increase its width by splitting nodes. The main advantage is that we can use information on the resulting DD to guide the refinement in the next iteration. This procedure can create exact and relaxed DDs but it cannot construct restricted DDs since it only removes infeasible solutions (Algorithm 2, line 7).

Algorithm 2 DD Iterative Refinement Construction Procedure

1: **procedure** ConstructDD(RF, $\mathcal{W}$, $\oplus$)
2: Create a width-one DD $\mathcal{D}$
3: **while** $\mathcal{D}$ has been modified **do**
4: **for** $i \in I$ **do**
5: Update (relaxed) state information in each node $u \in \mathcal{N}_i$
6: Split nodes in $\mathcal{N}_i$ and update their relaxed states if needed
7: Check all outgoing arcs of layer $\mathcal{N}_i$ and eliminate infeasible arcs
8: Update bottom-up state information in every node $u \in \mathcal{N}$
9: **return** $\mathcal{D}$

Algorithm 2 illustrates the iterative refinement procedure given a recursive model RF, a maximum width $\mathcal{W}$, and a merging operator $\oplus$. The procedure starts by constructing a width-one DD, i.e., a DD $\mathcal{D}$ where each layer $\mathcal{N}_i$, with $i \in I$, has a single node and emanating arcs for each value in their corresponding domain (line 1). The root node is associated with the initial state (i.e., $\boldsymbol{S}(\mathbf{r}) = \boldsymbol{S}_1$) and each following node corresponds to a relaxed state computed using the merging operator and its incoming arcs:

$$\boldsymbol{S}(u) = \bigoplus_{a \in \mathcal{A}^{\text{in}}(u)} \phi_{i-1}(\boldsymbol{S}(s(a)), v_a), \qquad \forall u \in \mathcal{N}_i,\ i \in \{2, \dots, n+1\}. \tag{2}$$

The procedure iteratively refines $\mathcal{D}$ one layer at a time starting with the first layer until it cannot be updated any further (lines 3-8). It first updates the relaxed states of each node in the current layer using (2), to guarantee up-to-date state information for each node (line 5). The algorithm uses the relaxed states to split a node $u \in \mathcal{N}_i$, i.e., it creates a new node u' with the same outgoing arcs that u has and redirects a portion of the incoming arcs of u to u'. The states of the split nodes are then updated and used to identify if any of the outgoing arcs lead to infeasible assignments.

The main advantage of this procedure is that in each iteration we obtain a DD that better approximates the original problem. Moreover, we can use the current DD relaxation to decide how to split nodes in order to remove as many infeasible arcs (i.e., arcs representing infeasible assignments) as possible. We can also create additional relaxed states for a node u using the partial assignments from u to node $\mathbf{t}$ (i.e., bottom-up relaxed states). These relaxed states are commonly used to find infeasible arcs that are not necessarily identifiable with standard relaxed states (Hadžić and Hooker 2007, Cire and van Hoeve 2013).

Example 5. Consider the knapsack problem with feasible set $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^4 : 7x_1 + 5x_2 + 4x_3 + x_4 \leq 8\}$ and recursive model R-KNP from our previous examples. We construct a relaxed DD for $\mathcal{X}$ as follows. For each node $u \in \mathcal{N}_i$, we consider a relaxed states $\boldsymbol{S}(u) = (Q_{\min}(u), Q_{\max}(u))$ where $Q_{\min}(u)$ and $Q_{\max}(u)$ represent the minimum and maximum load of the knapsack at node u and stage $i \in \{1, \dots, 5\}$, respectively. Thus, the merging operator is given by $\oplus = (\min, \max)$. We choose this relaxed state representation to identify if a node encodes multiple states and, therefore, if it is a candidate for splitting. Intuitively, a node $u \in \mathcal{N}$ represents a single state of R-KNP if $Q_{\min}(u) = Q_{\max}(u)$.

Lastly, we identify if an arc $a \in \mathcal{A}$ with source $s(a) \in \mathcal{N}_i$ is infeasible if

$$Q_{\min}(s(a)) + w_i v_a > 8 \tag{KP-R1}$$

for any $i \in I$. If arc a satisfies condition KP-R1, then all paths traversing a represent solutions with a knapsack load above its limit. Thus, we can remove arc a from $\mathcal{D}$.

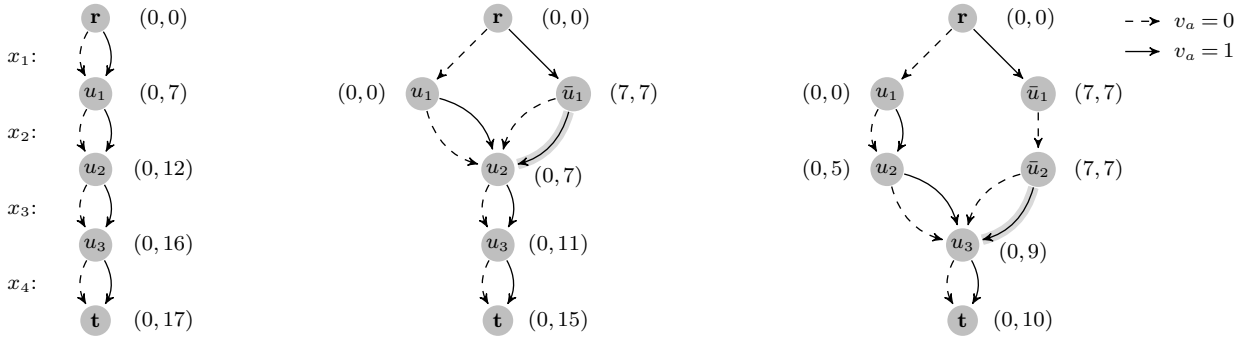

Figure 3 **Iterative refinement procedure for $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^4 : 7x_1 + 5x_2 + 4x_3 + x_4 \leq 8\}$ and $\mathcal{W} = 2$. The figure shows a width-one DD (left), a DD after splitting layer $\mathcal{N}_2$ (middle), and a DD after splitting layer $\mathcal{N}_3$ (right). Values next to nodes represent relaxed states. Highlighted arrows correspond to infeasible arcs that can be removed.**

Figure 3 illustrates the iterative refinement procedure for R-KNP with maximum width $\mathcal{W} = 2$ and relaxed states as defined above. The left most DD corresponds to a width-one DD, where the values next to each node represent its relaxed state. The middle DD depicts the update, split, and filter sub-routines in layer $\mathcal{N}_2$. Node $u_1 \in \mathcal{N}_2$ is split into two (i.e., u_1 and $\bar{u}_1$), each node with one incoming arc and relaxed states updated accordingly. Notice that arc $a = (\bar{u}_1, u_2)$ with value $v_a = 1$ is infeasible, since it satisfies KP-R1. The right graph shows the resulting DD after refining layer $\mathcal{N}_3$. In this DD, condition KP-R1 identifies infeasible arc $a = (\bar{u}_2, u_3)$ with value $v_a = 1$.

While condition KP-R1 identifies several infeasible arcs during the DD construction procedure, there exist paths in the right most DD of Figure 3 that correspond to infeasible solutions, e.g., path $(\mathbf{r}, u_1, u_2, u_3, \mathbf{t})$ associated with point $\boldsymbol{x} = (0, 1, 1, 1)$ is infeasible. This path cannot be removed from the DD since all its arcs are also associated with feasible solutions, i.e., all the arcs violate KP-R1. We can remove this path from the DD if we further split node u_2. ■

Analogous to the top-down procedure, the tightness of a relaxed DD from an iterative algorithm depends on the splitting heuristic. Ideally, we want to split nodes such that the relaxed states are close to exact states. However, an appropriate splitting procedure will greatly depend on the application at hand. For example, splitting nodes to exactly represent

specific problem characteristics is a good strategy to identify infeasible assignments and improve the DD relaxation (Cire and van Hoeve 2013, Castro et al. 2020a,b). By doing so, the relaxed DD will satisfy a subset of the constraints and, thus, have theoretical guarantees on the quality of the relaxation.

These two construction procedures have key distinctions that make them appealing for different purposes. Top-down procedures are considerably faster to implement and deploy than iterative refinement procedures, so a top-down construction is usually preferable for prototyping new ideas or when the user needs to construct multiple DDs. In contrast, iterative refinement procedures can lead to stronger bounds or exact enforcement of a subset of the constraints, thus, guaranteeing certain properties for the resulting relaxation. However, the iterative procedure is usually computationally slower than the top-down approach due to the multiple refinement iterations, and thus, it is preferable when we need to construct few relaxed DDs.

4.1.2. Other Construction Algorithms. A less commonly used technique to construct relaxed DDs is the separation procedure (Cire and Hooker 2014). This algorithm is a variant of the iterative refinement since it starts with a width-one DD and iteratively splits nodes. However, the separation procedure splits nodes systematically so that all the paths in the DD satisfy a constraint or have a longest-path value larger than a certain threshold (Bergman et al. 2011). This technique can generate highly-accurate relaxations if, for instance, we separate the constraint that is currently violated by the longest path in a maximization problem (Bergman and Cire 2016c). Recent work by van Hoeve (2020, 2021) shows that separation procedures are quite effective for the graph coloring problem: the DD relaxation yields bounds competitive with state-of-the-art methodologies.

Besides these construction techniques, two recent works have developed novel procedures to construct relaxed DDs. Römer et al. (2018) proposed a local search framework with operations to merge nodes, split nodes, and redirect arcs. Their procedure generalizes the top-down and iterative refinement algorithms and consistently yields better bounds than these alternatives. Horn et al. (2021) introduced an A^*-inspired algorithm to construct relaxed DDs. Starting at the root node, the procedure keeps a priority list of nodes to create next and can create and merge nodes in different layers. Recent works showed that the A^*-inspired approach can return tighter bounds than the top-down algorithm for the

prize-collecting scheduling problem (Horn et al. 2021) and the longest common subsequence problem (Horn and Raidl 2021).

Lastly, Bergman and Cire (2017) studied the problem of finding the best DD relaxation for a given width limit. The authors presented an IP model that partitions the nodes in each layer to obtain the strongest dual bound. Their experiments show that the resulting DD provides significantly stronger bounds than any other methodology. However, the computational time to solve their IP model can be quite long and, thus, impractical for many applications.

4.1.3. Variable Ordering. One of the main challenges when constructing a DD is the variable ordering, i.e., the assignment of variables to layers. The size of a DD depends on the variable ordering and the optimal ordering can lead to significantly smaller DDs. The problem of finding the optimal variable ordering is NP-hard (Bollig and Wegener 1996) and has been extensively studied for exact DDs. Several authors propose heuristic variable orderings for different applications, including the knapsack (Behle 2008) and the maximum independent set problem (Bergman et al. 2011).

Cappart et al. (2019) introduced a general ordering procedure based on reinforcement learning (RL). While their technique can be applied to any optimization problem, it is only compatible with the top-down construction procedure: during construction the RL agent chooses the variable that will be assigned to the next layer. A recent work by Parjadis et al. (2021) showed that the RL-based variable ordering can significantly improve the bound quality and the number of nodes explored during a branch-and-bound search.

Alternatively, Karahalios and van Hoeve (2021) studied portfolio algorithms to select an appropriate variable ordering for a problem. Their procedure considers several variable ordering heuristics and selects one for a given problem. In contrast to the Cappart et al. (2019) approach, the portfolio methodology provides a complete variable ordering and can be used with any DD construction algorithm.

4.2. DD-based Bounds

DD approximations are commonly used to compute bounds for optimization problems. The main advantage of DD bounds is that their tightness can be control by either changing the size of DD or the mechanism to build the diagram (see Section 4.1 for more details). Therefore, DD approximations provide a flexibility that is harder to obtained with other procedures (e.g., linear programming or standard primal heuristics).

In the following, we describe the main mechanisms to compute primal and dual bounds with DD approximations and how to enhance these bounds. We then discuss different ways to embed DD bounds into a branch-and-bound algorithm.

4.2.1. Dual Bounds. Andersen et al. (2007) first introduced the idea of using relaxed DDs to obtain dual bounds for an optimization problem. A relaxed DD over-approximates the set of feasible solutions, so optimizing over the DD provides a valid dual bound for the problem. The authors consider a DD $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ and the optimization problem $\max\{f(\boldsymbol{x}) : \boldsymbol{x} \in \text{Sol}(\mathcal{D})\}$ where the objective function is separable, i.e., $f(\boldsymbol{x}) = \sum_{i \in I} f_i(x_i)$. The main feature of this problem is that it can be solved using a longest-path algorithm over $\mathcal{D}$. Intuitively, we assign each arc $a \in \mathcal{A}$ a length given by $\ell_a = f_i(v_a)$ for any arc a emanating from node $u \in \mathcal{N}_i$. Then, the longest $\mathbf{r} - \mathbf{t}$ path is the optimal solution over $\text{Sol}(\mathcal{D})$.

This simple longest-path procedure provides a valid dual bound for any optimization problem where the feasible region is over-approximated by $\mathcal{D}$. Note that the longest-path optimization holds for separable objective functions (e.g., a linear objective), and it can be generalized for problems with a recursive formulation RF (Bergman et al. 2016b, Bergman and Cire 2018). In this case, the length of an arc $a \in \mathcal{A}$ corresponds to the immediate reward over v_a and the relaxed state of its emanating node, i.e., $\ell_a = g_i(\boldsymbol{S}(u), v_a)$ for any arc a emanating from node $u \in \mathcal{N}_i$. The procedure will return a valid dual bound if the node merging operator satisfies the conditions described in Section 2.3 for a proper relaxation (Hooker 2017).

Researchers have tested these bounds in a wide variety of combinatorial problems, including set covering (Bergman et al. 2011), multidimensional bin packing (Kell and Van Hoeve 2013), maximum independent set (Bergman et al. 2014c, 2016b), maximum cut, maximum 2-satisfiability (Bergman et al. 2016b), and graph coloring (van Hoeve 2020, 2021). These papers create a single relaxed DD to approximate the feasible set and compute bounds using the shortest/longest path procedure over the DD or, alternatively, the DD network flow model.

One of the most popular applications of relaxed DDs is for sequencing problems. These problems are generally formulated recursively and come with a natural variable order (i.e., choose elements in sequential order), which facilitates the construction of the DDs. Cire and van Hoeve (2013) first applied relaxed DDs to solve sequencing problems. Their

implementation updates the DD during branch-and-bound search to compute more accurate dual bounds. The authors tested their DD dual bounds with satisfactory results in several sequencing problems, including the traveling salesman problem (TSP) with time windows, TSP with precedence constraints, and sequencing problems with makespan and total tardiness minimization. This work was extended by Kinable et al. (2017) to tackle TSP variants with sequence-dependent cost and by Castro et al. (2020a) for pick-up and delivery problems.

Several researchers have studied the quality of the DD bounds for sequencing problems and compared them to those obtained from a linear programming (LP) relaxation. Hooker (2017) formalized some of the main components to create relaxed DDs for general sequencing problems and presents preliminary results on the bound quality for different DD construction procedures. van den Bogaerdt and de Weerdt (2018) used this framework to create bounds for the multi-machine scheduling problems with encouraging performance. Similarly, Maschler and Raidl (2018) studied the DD bound quality for a prize-collecting sequencing problem and compared the bounds given by a top-down and an iterative refinement construction scheme. Lastly, Castro et al. (2018, 2019, 2020b) explored different DD-based relaxations for AI planning problems and show encouraging results when comparing the DD dual bounds with those from an LP relaxation.

We note that relaxed DDs and dual bound computation are strongly related to state-space relaxations (Christofides et al. 1981). As we discussed in Sections 2.2 and 3.1, DDs can be seen as compact representations of the state-space of a recursive model and, therefore, relaxed DDs can be interpreted as state-space relaxations. These state-space relaxations are widely used in the vehicle routing literature (Baldacci et al. 2011, 2012) and in other applications, e.g., cutting stock problems (de Lima et al. 2022), to compute stronger bounds than LP relaxations. The main difference between these two techniques is that state-space relaxations are generally built by defining a mapping between the original states and the relaxed space. In contrast, the DD construction procedures are much more flexible since they can dynamically modify the DD to suite the user needs (see Section 4.1).

With this in mind, one key benefit of DDs is that they can be dynamically modified to generate better bounds. The simplest alternative is to increase the width limit. However, larger relaxed DDs require more computational resources, i.e., memory and time for compilation. Further, empirical results show that bound improvements decrease as the DD

width increases (Bergman et al. 2014c, 2016a, Castro et al. 2020b). The question of which DD width will lead to computationally efficient relaxed DDs that provide informative dual bounds is still open.

4.2.2. Lagrangian Bounds. An alternative for computing better dual bounds is to enhance the DD relaxation with dual information from an IP formulation. Consider a minimization problem with feasible set $\mathcal{X} \subseteq \mathbb{Z}^n$, a linear objective function $\boldsymbol{c}^\top \boldsymbol{x}$, and a relaxed DD $\mathcal{D}$ that over-approximates $\mathcal{X}$, i.e., $\mathcal{X} \subseteq \text{Sol}(\mathcal{D})$. The idea is to consider a set of m valid inequalities $A\boldsymbol{x} \leq \boldsymbol{b}$ as penalties to the objective function to avoid paths in $\mathcal{D}$ that are infeasible. The resulting problem is a Lagrangian sub-problem

$$\mathcal{L}(\boldsymbol{\lambda}) = \min\{\boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{\lambda}^\top (A\boldsymbol{x} - \boldsymbol{b}) : \boldsymbol{x} \in \text{Sol}(\mathcal{D})\},$$

where $\boldsymbol{\lambda} \in \mathbb{R}^m_+$ are the Lagrangian penalties.

Since $\text{Sol}(\mathcal{D})$ can be reformulated as a network flow model $\text{NF}(\mathcal{D})$, the theoretical results of Lagrangian duality hold for $\mathcal{L}(\boldsymbol{\lambda})$ (Conforti et al. 2014, Fisher 2004). In particular, $\mathcal{L}(\boldsymbol{\lambda})$ for any $\boldsymbol{\lambda} \in \mathbb{R}^m_+$ is a valid dual bound for the original problem. Then, the Lagrangian dual problem seeks $\boldsymbol{\lambda}$ that gives the tightest dual bounds. In our minimization example, the Lagrangian dual maximizes the sub-problem $\mathcal{L}(\boldsymbol{\lambda})$ and, thus, returns a bound that is equal to or stronger than the DD relaxation:

$$\min\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in \text{Sol}(\mathcal{D})\} \leq \max\{\mathcal{L}(\boldsymbol{\lambda}) : \boldsymbol{\lambda} \in \mathbb{R}^m\}.$$

Bergman et al. (2015b) first proposed Lagrangian duality as a mechanism to enhance DD relaxations. Their approach considers a DD that represents a subset of the constraints of a problem. The Lagrangian procedure introduces dual information for the remaining constraints as penalties in the objective function. The authors tested their procedure over the TSP with encouraging results, where the DD-based Lagrangian relaxation returns significantly tighter bounds than the pure DD relaxation. Hooker (2019) further explored this idea for a family of sequencing problems and presents a detailed experimental evaluation with similar conclusions.

Castro et al. (2020a) employed this procedure for a pick-up and delivery problem and experimented with different DD and Lagrangian relaxations. Their experiments show that the DD-based Lagrangian bounds are affected by the relaxed inequalities and the constraints that are prioritized inside the DD. The authors conjectured that we can get better

bounds by dualizing inequalities that are hard to represent inside the DD and prioritizing the remaining constraints in the DD relaxation.

Tjandraatmadja and van Hoeve (2020) explored DD-based Lagrangian bounds for general ILP problems and proposed a decomposition approach. Instead of representing the full problem with a relaxed DD, the authors considered sub-structures that are known to be easily representable by a DD (e.g., conflict graphs). Thus, their procedure creates a DD for a specific sub-structure and uses Lagrangian duality to penalize the remaining constraints in order to improve the bounds and remove infeasible arcs. The main advantage of this procedure is that it can be used by any ILP model as long as there is a sub-structure that can be efficiently exploited by a DD.

While most works in the literature use Lagrangian duality over a single DD, Bergman et al. (2015a) proposed a Lagrangian decomposition approach to communicate information over multiple DDs. The idea is to have several DDs representing different constraints and use Lagrangian penalties to synchronize their solutions. The authors introduced this technique in the CP literature to improve propagation across multiple DDs. Lange and Swoboda (2021) explored a similar idea for integer linear programming models where each linear constraint is represented with a DD. The authors employed a message-passing algorithm to solve the Lagrangian dual problem and showed that their approach obtains competitive bounds when compare to commercial solvers.

We note that despite the simplicity of this Lagrangian dual bounds, the works employing this procedure are very limited. The main advantage of Lagrangian procedures is that they avoid the construction of huge relaxed DDs to create tight bounds. As pointed out by several authors (Tjandraatmadja and van Hoeve 2020, Castro et al. 2020a), it could be more beneficial to create an exact or relaxed DD that represents a subset of the constraints and introduce the remaining constraints as dual penalties. By doing so, we can exploit problem structures with potentially smaller DDs and obtain better bounds than when we try to represent the complete problem with a large relaxed DD.

4.2.3. Primal bounds. In contrast to relaxed DDs, restricted DDs compute primal bounds and provide feasible solutions for a problem. Bergman et al. (2014d) first introduced this graphical structure as a general procedure to heuristically generate solutions for discrete optimization problems. Their approach computes primal bounds by optimizing the restricted DD using the longest/shortest path procedure for optimization problems (see

Section 4.2.1). Since all the paths represent feasible solutions, an optimal path corresponds to the tightest primal bound given by the restricted DD.

The main advantage of restricted DDs is that they encode a set of feasible solutions, so they can potentially compute stronger primal bounds than other methodologies. For example, Bergman et al. (2014d) showed that DD primal bounds are competitive to the ones provided by IP solvers for set covering and set packing problems. Moreover, we can introduce restricted DDs into search algorithms to obtain stronger primal bounds. For instance, ONeil and Hoffman (2019) created restricted DDs to solve a TSP problem with pick-ups and deliveries in an online setting. The authors used small-width restricted DDs within a branch-and-bound search to find high-quality solutions in a few seconds.

We also note that relaxed DDs can also provide primal bounds. For example, Horn and Raidl (2019) used a limited discrepancy search procedure guided by relaxed DD dual bounds to find feasible solutions for a prize-collecting job sequencing problem. Alternatively, we can extract feasible solutions from a relaxed DD using some heuristic methods that traverse the DD from root to terminal node. However, the procedure might be unsuccessful if we select a partial path that leads to infeasible solutions.

Lastly, the primal bound of a restricted DD can also certify the feasibility of a problem. For instance, Kell and Van Hoeve (2013) used restricted DDs to show the feasibility of multidimensional bin packing problems. If the restricted DD has at least one path, then the problem is feasible. This simple procedure proved feasibility for several instances that IP and CP technologies could not in a given time limit.

4.2.4. Branch-and-Bound Procedures. One of the main advantages of relaxed DDs is that they can provide stronger dual bounds than an LP relaxation (Bergman et al. 2014c). Thus, replacing the LP relaxation with a DD relaxation in a branch-and-bound procedure can be very advantageous.

Bergman et al. (2016b) proposed a general branch-and-bound scheme where relaxed and restricted DDs provide dual and primal bounds, respectively. The main difference with standard LP-based branch-and-bound is that their procedure branches over nodes of a relaxed DD instead of variable-value assignments. Thus, the DD-based branch-and-bound can potentially generate fewer sub-problems since each branching decision fixes the values of multiple variables at a time. Gillard et al. (2021) presented two pruning techniques to

further improve the DD-based branch-and-bound procedure that leverage local relaxed DD bound information to avoid exploring nodes that lead to sub-optimal solutions.

Bergman et al. (2014a) extended the DD-based branch-and-bound procedure for parallel computing, where every core is responsible for a DD sub-problem. Their procedure takes advantage of the flexibility of DDs to efficiently process the sub-problems and communicate bounds between each sub-problem.

González et al. (2020b) proposed a mechanism that integrates IP into the DD-based branch-and-bound. The authors modified the DD-based branch-and-bound of Bergman et al. (2016b) so that relaxed nodes can be either solved by an IP solver or follow the original DD branching mechanism. To identify which node should be solved by an IP solver, the authors implemented a supervised learning technique that chooses nodes during search. This novel DD-based branch-and-bound procedure can be applied to any combinatorial problem and shows promising results in the maximum independent set problem and the quadratic stable set problem (González et al. 2020a).

DD bounds can also be integrated into other search procedures, such as a standard branch-and-bound (Cire and van Hoeve 2013, Kinable et al. 2017, Castro et al. 2020b, Tjandraatmadja and van Hoeve 2020). The main disadvantage is that the search algorithm might not leverage the DD structure as the specialized DD-based branch-and-bound does. To take advantage of the DD structure it is important to branch on variables following the ordering in the DD. Some authors also note that a depth-first search strategy is preferable for DDs since the branching updates can be done more efficiently by removing arcs in the last branched layer (Cire and van Hoeve 2013, Castro et al. 2020a).

4.3. CP Propagation

As reviewed in Section 3.1, several researchers in the CP community encode global constraints using exact DDs. However, the size of an exact DD grows exponentially with the number of variables, so exact DDs become impractical for large combinatorial structures. Andersen et al. (2007) proposed to represent global constraints with relaxed DDs to avoid the exponential size of exact DDs.

While relaxed DDs are more flexible than exact DDs in terms of memory usage, relaxed DDs do not achieve generalized arc consistency (GAC) in polynomial time as in the case of exact DD. Since some paths in a relaxed DD are infeasible, checking if there exists a feasible solution for a specific variable-value assignment is not a trivial task. Andersen et al.

(2007) proposed a new consistency measure to better analyze the propagation capabilities of a relaxed DD. We say that a relaxed DD $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ achieves *DD consistency* for a global constraint $\mathcal{C}$ if for each arc $a \in \mathcal{A}$ there exists a path p that traverses a and is feasible with respect to $\mathcal{C}$. Intuitively, DD consistency asks for a relaxed DD with no infeasible arcs, which is NP-hard in many cases. Note that identifying all infeasible arcs also results in identifying all infeasible variable-value assignments, thus, DD consistency implies GAC (Andersen et al. 2007).

Researchers have proposed sophisticated propagation mechanisms for different global constraints to achieve DD consistency in polynomial time. A propagation procedure for a relaxed DD is defined by the set of relaxed states and the set of conditions to identify infeasible arcs (see Section 4.1). Andersen et al. (2007) proposed the first relaxed DD propagators for `Linear` and `All-different` constraints. Later, Hadžić et al. (2008b) extended the propagator from linear inequalities to separable inequalities and showed that it achieves DD consistency in polynomial time.

Since the work of Andersen et al. (2007), several papers have introduced DD propagators for other well-known global constraints. Hoda et al. (2010) explored DD propagators for the `Among` and `Element` constraints. The authors also proposed new conditions to identify infeasible arcs for the `All-different` propagator, improving its propagation capabilities. Bergman et al. (2014b) presented a DD propagator for the `Sequence` constraint and show that establishing DD consistency is NP-hard. More recently, Perez and Régin (2017a) created an DD encoding for the `Dispersion` constraint where the DD enforces the mean value constraint and uses a cost-based propagation for the variance restriction.

Besides the encoding of existing global constraints, relaxed DDs are also a building block to create new global constraints. Cire and van Hoeve (2012) introduced a global constraint for disjunctive scheduling based on a relaxed DD. Their DD represents the set of job sequences and considers release times, deadlines, precedence relations, and sequence-dependent set-up times.

Two recent works develop new probabilistic global constraints using DDs. Perez and Régin (2017a) introduced a Probability Mass Function (PMF) constraint where a DD encodes the linear inequality restricting the mean value of the variables and a cost-based propagator to ensure that the probability of every feasible assignment is inside a pre-defined

range. The authors extended this constraint to consider probability distributions given by a Markov chain process (Perez and Régin 2017b).

DDs can also improve the propagation capabilities of a CP solver by sharing information. Hadžić et al. (2009) first studied this idea using compatibility labels between multiple DDs. Their procedure constructs an exact or relaxed DD for each constraint of the problem following the same variable ordering. It then traverses the nodes of each DD to identify compatibility with nodes in different DDs. The authors showed that nodes that do not have any compatibility label can be removed since the solutions traversing that node are infeasible in all other DDs. Bergman et al. (2015a) introduced a different procedure to communicate information between DDs based on Lagrangian decomposition, which we discussed in Section 4.2.2.

While relaxed DDs can model a wide variety of global constraints, there could be alternative procedures that are more suitable for some constraints. Specifically, Andersen et al. (2007) noted that there are polynomial-time algorithms to enforce GAC for some constraints but it could be NP-hard for polynomial-size DDs to do so. A simple example is the `All-different` constraint that achieves GAC in polynomial time by representing the constraint as a matching problem in a bipartite graph. However, GAC is NP-hard in a relaxed DD because the GAC problem reduces to a Hamiltonian path problem.

To summarize, relaxed DDs are attractive alternatives to construct global constraints due to their propagation capabilities and flexibility to represent a wide range of combinatorial structures. However, one of the main practical limitations of relaxed DDs is their implementation, which usually requires the user to develop all the necessary components to build and propagate the DDs. Recent work by Gentzel et al. (2020) presented `Haddock`, a declarative language and architecture for DD compilation. The software supports a wide range of existing DD propagators and can declare multiple DDs within a CP model. Moreover, `Haddock` has comparable performance when compared to dedicated MDD propagators for different constraints.

5. Conclusions and Future Challenges

This paper reviews recent advances using decision diagrams (DDs) to model and solve discrete optimization problems. We describe several procedures that benefit from a graphical representation of the solution set and show how these techniques can be integrated into

other optimization paradigms (e.g., IP and CP). In particular, we distinguish six different ways to employ DDs: modeling, feasibility checking, solution extraction, computing primal and dual bounds, inference and propagation, and solution-space analysis.

There are two key advantages of DDs that explain their success in many of the applications discussed. First, DDs can be efficiently optimized using different linear objective functions. This property is crucial for decomposition techniques since DDs are iteratively used, for example, in cutting plane procedures to solve the separation problem for each new fractional point. Similarly, DDs are computationally efficient when we need to extract information from the diagram multiple times, as in the case of the no-good inference procedures.

The second key advantage of DDs is their versatility when solving optimization problems. For example, we can construct a relaxed DD to obtain a dual bound and have a heuristic procedure that extracts feasible solutions from the DD (i.e., primal bounds). Similarly, we can apply the same DD to generate valid inequalities, prune sub-optimal solutions, and infer no-good assignments for the same problem. This property makes DDs a strong and flexible optimization tool that can benefit from techniques developed in different fields.

There are, however, several DD limitations and challenges. Most importantly, DDs can grow exponentially with the number of variables, which significantly limits their applicability. While this limitation can be partially addressed by employing approximate DDs, the quality of the approximation also depends on the size of the problem and other parametrizations. Other drawbacks include implementation challenges (i.e., lack of general and reliable DD packages for optimization) and modeling limitations (e.g., continuous variables).

While recent works have significantly expanded the use of DDs to solve discrete optimization problems, there are many avenues yet to be explored. In particular, it is still unclear when DDs should be used. As previously mentioned, the size of a DD is a significant limitation, so researchers usually focus on applications that are combinatorially challenging but where the number of variables is small enough that the solution set can be represented with the DD. Sequencing problems are good examples since they usually have weak LP relaxations due to big-M constraints and problems with few variables can be challenging for commercial solvers (Cire and van Hoeve 2013, Kinable et al. 2017, Castro et al. 2020a).

Nonetheless, we know very little about DD representability and how to efficiently construct these diagrams. We believe that it is crucial to understand which combinatorial structures are better suited to DDs so we can exploit them in our implementations. A good example of this idea is the paper by Tjandraatmadja and van Hoeve (2020). The authors proposed a DD approach that can be integrated into an IP solver, but instead of modeling the full problem with a DD they focus solely on conflict graphs, a combinatorial structure that can be efficiently model with DDs. Since representing the full problem with a DD is often impractical, knowing which sub-structures are better suited for DDs can provide a better idea of how to employ DDs in practice.

With this idea in mind, we believe that combining DDs with existing optimization solvers can lead to state-of-the-art performance. Recent papers have shown the potential of this integration, for example, by employing DDs as a component of a decomposition algorithm (e.g., column generation, cutting planes, and Lagrangian relaxation). However, most of these works rely on existing decomposition algorithms and use DDs to efficiently solve combinatorial problems. A promising research direction is to explore new decomposition mechanisms that can leverage DDs to their full potential. DD-based branch-and-bound (Bergman et al. 2016b, González et al. 2020b) is a good example of a methodology specially made for DDs since the branching and bounding algorithms are designed to take advantage of the graphical structure.

Another interesting research direction is to develop algorithms that can leverage information from multiple DDs. The main advantage of this idea is that we can represent larger problems by using multiple DDs that model small portions of them. This idea has been briefly studied in the literature, for example, with Lagrangian decomposition (Bergman et al. 2015a, Lange and Swoboda 2021) and network flow models with linking constraints (Lozano et al. 2020a). However, the use of multiple DDs is still very limited, and, to the best of our knowledge, there are no efficient algorithms to integrate or share information between two or more DDs.

Decision Diagrams are flexible optimization tools that have shown state-of-the-art results solving a wide variety of problems. DDs can be integrated with other optimization paradigms and used in different ways to solve a problem. Nonetheless, we still know very little of these graphical structures and how to properly exploit them for optimization purposes. Thus, there is a wide range of possibilities yet to be explored that can leverage DDs to solve challenging optimization problems.

Acknowledgments

We thank the editors and reviewers whose valuable feedback helped improve the paper. Funding was provided by Agencia Nacional de Investigación y Desarrollo de Chile (Becas Chile) and the Natural Sciences and Engineering Research Council of Canada (Grant Nos. RGPIN-2015-05072, RGPIN-2020-06054, RGPIN-2020-04039).

References

Akers SB (1978) Binary decision diagrams. *IEEE Transactions on computers* C-27(6):509–516.

Amilhastre J, Fargier H, Niveau A, Pralet C (2014) Compiling CSPs: A complexity map of (non-deterministic) multivalued decision diagrams. *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence*, 1–8.

Andersen HR, Hadžić T, Hooker JN, Tiedemann P (2007) A constraint store based on multivalued decision diagrams. *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 118–132.

Baldacci R, Mingozzi A, Roberti R (2011) New route relaxation and pricing strategies for the vehicle routing problem. *Operations research* 59(5):1269–1283.

Baldacci R, Mingozzi A, Roberti R (2012) New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS Journal on Computing* 24(3):356–371.

Barnhart C, Johnson EL, Nemhauser GL, Savelsbergh MW, Vance PH (1998) Branch-and-price: Column generation for solving huge integer programs. *Operations research* 46(3):316–329.

Becker B, Behle M, Eisenbrand F, Wimmer R (2005) BDDs in a branch and cut framework. *Proceedings of the International Workshop on Experimental and Efficient Algorithms*, 452–463.

Behle M (2007) *Binary decision diagrams and integer programming*. Ph.D. thesis, Saarland University.

Behle M (2008) On threshold BDDs and the optimal variable ordering problem. *Journal of Combinatorial Optimization* 16(2):107–118.

Behle M, Eisenbrand F (2007) 0/1 vertex and facet enumeration with BDDs. *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments*, 158–165.

Benders J (1962) Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik* 4(1):238–252.

Bergman D, Bodur M, Cardonha C, Cire AA (2021) Network models for multiobjective discrete optimization. *INFORMS Journal on Computing* (In press).

Bergman D, Cardonha C, Mehrani S (2019) Binary decision diagrams for bin packing with minimum color fragmentation. *Proceedings of the International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 57–66.

Bergman D, Cire AA (2016a) Decomposition based on decision diagrams. *Proceedings of the International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 45–54.

Bergman D, Cire AA (2016b) Multiobjective optimization by decision diagrams. *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 86–95.

Bergman D, Cire AA (2016c) Theoretical insights and algorithmic tools for decision diagram-based optimization. *Constraints* 21(4):533–556.

Bergman D, Cire AA (2017) On finding the optimal BDD relaxation. *Proceedings of the International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 41–50.

Bergman D, Cire AA (2018) Discrete nonlinear optimization by state-space decompositions. *Management Science* 64(10):4700–4720.

Bergman D, Cire AA, Sabharwal A, Samulowitz H, Saraswat V, van Hoeve WJ (2014a) Parallel combinatorial optimization with decision diagrams. *Proceedings of the International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, 351–367.

Bergman D, Cire AA, van Hoeve WJ (2014b) MDD propagation for sequence constraints. *Journal of Artificial Intelligence Research* 50(1):697–722.

Bergman D, Cire AA, van Hoeve WJ (2015a) Improved constraint propagation via Lagrangian decomposition. *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 30–38.

Bergman D, Cire AA, van Hoeve WJ (2015b) Lagrangian bounds from decision diagrams. *Constraints* 20(3):346–361.

Bergman D, Cire AA, van Hoeve WJ, Hooker JN (2014c) Optimization bounds from binary decision diagrams. *INFORMS Journal on Computing* 26(2):253–268.

Bergman D, Cire AA, van Hoeve WJ, Hooker JN (2016a) *Decision Diagrams for Optimization* (Springer International Publishing), 1st edition.

Bergman D, Cire AA, van Hoeve WJ, Hooker JN (2016b) Discrete optimization with decision diagrams. *INFORMS Journal on Computing* 28(1):47–66.

Bergman D, Cire AA, van Hoeve WJ, Yunes T (2014d) BDD-based heuristics for binary optimization. *Journal of Heuristics* 20(2):211–234.

Bergman D, Lozano L (2021) Decision diagram decomposition for quadratically constrained binary optimization. *INFORMS Journal on Computing* 33(1):401–418.

Bergman D, van Hoeve WJ, Hooker JN (2011) Manipulating MDD relaxations for combinatorial optimization. *Proceedings of the International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, 20–35.

Bertsekas DP (2017) *Dynamic programming and optimal control*, volume 1 (Athena scientific Belmont, MA), 3rd edition.

Bollig B, Wegener I (1996) Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on computers* 45(9):993–1002.

Bryant RE (1986) Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers* 100(8):677–691.

Bryant RE (1992) Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys* 24(3):293–318.

Cappart Q, Goutierre E, Bergman D, Rousseau LM (2019) Improving optimization bounds using machine learning: Decision diagrams meet deep reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 1443–1451.

Castro MP, Cire AA, Beck JC (2020a) An MDD-based Lagrangian approach to the multi-commodity pickup-and-delivery TSP. *INFORMS Journal on Computing* 32(2):263–278.

Castro MP, Cire AA, Beck JC (2021) A combinatorial cut-and-lift procedure with an application to 0-1 second-order conic programming. *Mathematical Programming* (In press).

Castro MP, Piacentini C, Cire AA, Beck JC (2018) Relaxed decision diagrams for cost-optimal classical planning. *Proceedings of the Workshop on Heuristics and Search for Domain-Independent Planning*, 50–58.

Castro MP, Piacentini C, Cire AA, Beck JC (2019) Relaxed BDDs: An admissible heuristic for delete-free planning based on a discrete relaxation. *Proceedings of the International Conference on Automated Planning and Scheduling*, 77–85.

Castro MP, Piacentini C, Cire AA, Beck JC (2020b) Solving delete free planning with relaxed decision diagram based heuristics. *Journal of Artificial Intelligence Research* 67(1):607–651.

Cheng KC, Yap RH (2008) Maintaining generalized arc consistency on ad hoc r-ary constraints. *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 509–523.

Cheng KC, Yap RH (2010) An MDD-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. *Constraints* 15(2):265–304.

Christofides N, Mingozzi A, Toth P (1981) State-space relaxation procedures for the computation of bounds to routing problems. *Networks* 11(2):145–164.

Cire AA, Diamant A, Yunes T, Carrasco A (2019) A network-based formulation for scheduling clinical rotations. *Production and Operations Management* 28(5):1186–1205.

Cire AA, Hooker JN (2014) The separation problem for binary decision diagrams. *Proceedings of the International Symposium on Artificial Intelligence and Mathematics*, 1–7.

Cire AA, van Hoeve WJ (2012) MDD propagation for disjunctive scheduling. *Proceedings of the International Conference on Automated Planning and Scheduling*, 11–19.

Cire AA, van Hoeve WJ (2013) Multivalued decision diagrams for sequencing problems. *Operations Research* 61(6):1411–1428.

Conforti M, Cornuéjols G, Zambelli G (2014) *Integer Programming* (Springer International Publishing), 1st edition.

Cornuéjols G (2008) Valid inequalities for mixed integer linear programs. *Mathematical Programming* 112(1):3–44.

Davarnia D (2021) Strong relaxations for continuous nonlinear programs based on decision diagrams. *Operations Research Letters* 49(2):239–245.

Davarnia D, van Hoeve WJ (2021) Outer approximation for integer nonlinear programs via decision diagrams. *Mathematical Programming* 187(1):111–150.

de Lima VL, Alves C, Clautiaux F, Iori M, de Carvalho JMV (2022) Arc flow formulations based on dynamic programming: Theoretical foundations and applications. *European Journal of Operational Research* 296(1):3–21.

de Uña D, Gange G, Schachte P, Stuckey PJ (2019) Compiling CP subproblems to MDDs and d-DNNFs. *Constraints* 24(1):56–93.

de Weerdt M, Baart R, He L (2021) Single-machine scheduling with release times, deadlines, setup times, and rejection. *European Journal of Operational Research* 291(2):629–639.

Dijkstra EW, et al. (1959) A note on two problems in connexion with graphs. *Numerische mathematik* 1(1):269–271.

Fisher ML (2004) The Lagrangian relaxation method for solving integer programming problems. *Management science* 50(12):1861–1871.

Frohner N, Raidl GR (2019a) Merging quality estimation for binary decision diagrams with binary classifiers. *Proceedings of the International Conference on Machine Learning, Optimization, and Data Science*, 445–457.

Frohner N, Raidl GR (2019b) Towards improving merging heuristics for binary decision diagrams. *Proceedings of the International Conference on Learning and Intelligent Optimization*, 30–45.

Gange G, Stuckey PJ, Szymanek R (2011) MDD propagators with explanation. *Constraints* 16(4):407.

Gange G, Stuckey PJ, Van Hentenryck P (2013) Explaining propagators for edge-valued decision diagrams. *Proceedings of International Conference on Principles and Practice of Constraint Programming*, 340–355.

Gentzel R, Michel L, van Hoeve WJ (2020) Haddock: A language and architecture for decision diagram compilation. *International Conference on Principles and Practice of Constraint Programming*, 531–547 (Springer).

Gillard X, Coppé V, Schaus P, Cire AA (2021) Improving the filtering of branch-and-bound MDD solver. *Proceedings of the International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 231–247.

González JE, Cire AA, Lodi A, Rousseau LM (2020a) BDD-based optimization for the quadratic stable set problem. *Discrete Optimization* (In press).

González JE, Cire AA, Lodi A, Rousseau LM (2020b) Integrated integer programming and decision diagram search tree with an application to the maximum independent set problem. *Constraints* 25(1-2):23–46.

Guo C, Bodur M, Aleman DM, Urbach DR (2021) Logic-based Benders decomposition and binary decision diagram based approaches for stochastic distributed operating room scheduling. *INFORMS Journal on Computing* 33(4):1551–1569.

Hadžić T, Hooker JN (2006) Postoptimality analysis for integer programming using binary decision diagrams. Technical report, Carnegie Mellon University.

Hadžić T, Hooker JN (2007) Cost-bounded binary decision diagrams for 0-1 programming. *Proceedings of the International Conference on Integration of AI and OR Techniques in Constraint Programming*, 84–98.

Hadžić T, Hooker JN, OSullivan B, Tiedemann P (2008a) Approximate compilation of constraints into multivalued decision diagrams. *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 448–462.

Hadžić T, Hooker JN, Tiedemann P (2008b) Propagating separable equalities in an MDD store. *Proceedings of the International Conference on Integration of AI and OR Techniques in Constraint Programming*, 318–322.

Hadžić T, O'Mahony E, O'Sullivan B, Sellmann M (2009) Enhanced inference for the market split problem. *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence*, 716–723.

Hadžić T, Subbarayan S, Jensen RM, Andersen HR, Møller J, Hulgaard H (2004) Fast backtrack-free product configuration using a precompiled solution space representation. *Proceedings of the International Conference on Economic, Technical and Organizational aspects of Product Configuration Systems* 131–138.

Haus UU, Michini C (2017) Compact representations of all members of an independence system. *Annals of Mathematics and Artificial Intelligence* 79(1-3):145–162.

Haus UU, Michini C, Laumanns M (2017) Scenario aggregation using binary decision diagrams for stochastic programs with endogenous uncertainty. arXiv:1701.04055.

Hoda S, Van Hoeve WJ, Hooker JN (2010) A systematic approach to MDD-based constraint programming. *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 266–280.

Hooker JN (2013) Decision diagrams and dynamic programming. *Proceedings of the International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, 94–110.

Hooker JN (2017) Job sequencing bounds from decision diagrams. *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 565–578.

Hooker JN (2019) Improved job sequencing bounds from decision diagrams. *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 268–283.

Horn M, Maschler J, Raidl GR, Rönnberg E (2021) A*-based construction of decision diagrams for a prize-collecting scheduling problem. *Computers & Operations Research* 126:105–125.

Horn M, Raidl GR (2019) Decision diagram based limited discrepancy search for a job sequencing problem. *Computer Aided Systems Theory – EUROCAST*, 344–351.

Horn M, Raidl GR (2021) A*-based compilation of relaxed decision diagrams for the longest common subsequence problem. *Proceedings of the International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 72–88.

Hosseininasab A, van Hoeve WJ (2021) Exact multiple sequence alignment by synchronized decision diagrams. *INFORMS Journal on Computing* 33(2):721–738.

Jung V, Régin JC (2021) Checking constraint satisfaction. *Proceedings of the International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 332–347.

Karahalios A, van Hoeve WJ (2021) Variable ordering for decision diagrams: A portfolio approach. Technical report, Carnegie Mellon University.

Kell B, Sabharwal A, van Hoeve WJ (2015) BDD-guided clause generation. *Proceedings of the Integration of AI and OR Techniques in Constraint Programming*, 215–230.

Kell B, Van Hoeve WJ (2013) An MDD approach to multidimensional bin packing. *Proceedings of the International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, 128–143.

Kinable J, Cire AA, van Hoeve WJ (2017) Hybrid optimization methods for time-dependent sequencing problems. *European Journal of Operational Research* 259(3):887–897.

Kowalczyk D, Leus R (2018) A branch-and-price algorithm for parallel machine scheduling using ZDDs and generic branching. *INFORMS Journal on Computing* 30(4):768–782.

Lange JH, Swoboda P (2021) Efficient message passing for 0–1 ILPs with binary decision diagrams. *Proceedings of the International Conference on Machine Learning*, 6000–6010.

Latour AL, Babaki B, Dries A, Kimmig A, Van den Broeck G, Nijssen S (2017) Combining stochastic constraint optimization and probabilistic programming. *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 495–511.

Latour ALD, Babaki B, Nijssen S (2019) Stochastic constraint propagation for mining probabilistic networks. *Proceedings of the International Joint Conference on Artificial Intelligence*, 1137–1145.

Lozano L, Bergman D, Smith JC (2020a) On the consistent path problem. *Operations Research* 68(6):1913–1931.

Lozano L, Magazine MJ, Polak GG (2020b) Decision diagram-based integer programming for the paired job scheduling problem. *IISE Transactions* 53(6):671–684.

Lozano L, Smith JC (2018) A binary decision diagram based algorithm for solving a class of binary two-stage stochastic programs. *Mathematical Programming* (In press).

Martin RK (1987) Generating alternative mixed-integer programming models using variable redefinition. *Operations Research* 35(6):820–831.

Maschler J, Raidl GR (2018) Multivalued decision diagrams for a prize-collecting sequencing problem. *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling*, 375–397.

Mehrani S, Cardonha C, Bergman D (2021) Models and algorithms for the bin-packing problem with minimum color fragmentation. *INFORMS Journal on Computing* (In Press).

Minato S (1993) Zero-suppressed BDDs for set manipulation in combinatorial problems. *Proceedings of the International Design Automation Conference*, 272–277.

Morrison DR, Sewell EC, Jacobson SH (2016) Solving the pricing problem in a branch-and-price algorithm for graph coloring using zero-suppressed binary decision diagrams. *INFORMS Journal on Computing* 28(1):67–82.

Nadarajah S, Cire AA (2020) Network-based approximate linear programming for discrete optimization. *Operations Research* 68(6):1767–1786.

Nishino M, Yasuda N, Minato S, Nagata M (2015) BDD-constrained search: A unified approach to constrained shortest path problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 1219–1225.

ONeil RJ, Hoffman K (2019) Decision diagrams for solving traveling salesman problems with pickup and delivery in real time. *Operations Research Letters* 47(3):197–201.

Parjadis A, Cappart Q, Rousseau LM, Bergman D (2021) Improving branch-and-bound using decision diagrams and reinforcement learning. *Proceedings of the International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 446–455.

Perez G, Régin JC (2014) Improving GAC-4 for table and MDD constraints. *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 606–621.

Perez G, Régin JC (2015a) Efficient operations on MDDs for building constraint programming models. *Proceedings of the International Joint Conference on Artificial Intelligence*.

Perez G, Régin JC (2015b) Relations between MDDs and tuples and dynamic modifications of MDDs based constraints. arXiv:1505.02552.

Perez G, Régin JC (2016) Constructions and in-place operations for MDDs based constraints. *Proceedings of the International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 279–293.

Perez G, Régin JC (2017a) MDDs are efficient modeling tools: An application to some statistical constraints. *Proceedings of the International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 30–40.

Perez G, Régin JC (2017b) MDDs: sampling and probability constraints. *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 226–242.

Perez G, Régin JC (2017c) Soft and cost MDD propagators. *Proceedings of the AAAI Conference on Artificial Intelligence*, 3922–3928.

Perez G, Régin JC (2018) Parallel algorithms for operations on multi-valued decision diagrams. *Proceedings of the AAAI Conference on Artificial Intelligence*, 6625–6632.

Ploskas N, Laughman C, Raghunathan AU, Sahinidis NV (2019) Heat exchanger circuitry design by decision diagrams. *Proceedings of the International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 461–471.

Raghunathan AU, Bergman D, Hooker JN, Serra T, Kobori S (2018) Seamless multimodal transportation scheduling. arXiv:1807.09676.

Riascos-Álvarez LC, Bodur M, Aleman DM (2020) A branch-and-price algorithm enhanced by decision diagrams for the kidney exchange problem. arXiv:2009.13715.

Römer M, Cire AA, Rousseau LM (2018) A local search framework for compiling relaxed decision diagrams. *Proceedings of the International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 512–520.

Rossi F, Van Beek P, Walsh T (2006) *Handbook of constraint programming* (Elsevier), 1st edition.

Roy P, Perez G, Régin JC, Papadopoulos A, Pachet F, Marchini M (2016) Enforcing structure on temporal sequences: the allen constraint. *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 786–801.

Salemi H, Davarnia D (2021) On the structure of decision diagram-representable mixed integer programs with application to unit commitment. Optimization Online.

Schiex T, Verfaillie G (1994) Nogood recording for static and dynamic constraint satisfaction problems. *International Journal on Artificial Intelligence Tools* 3(02):187–207.

Serra T, Hooker J (2020) Compact representation of near-optimal integer programming solutions. *Mathematical Programming* 182(1):199–232.

Serra T, Raghunathan AU, Bergman D, Hooker JN, Kobori S (2019) Last-mile scheduling under uncertainty. *Proceedings of the International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 519–528.

Subbarayan S (2008) Efficient reasoning for nogoods in constraint solvers with BDDs. *Proceedings of the International Symposium on Practical Aspects of Declarative Languages*, 53–67.

Suzuki H, Ishihata M, Minato S (2018) Exact computation of strongly connected reliability by binary decision diagrams. *International Conference on Combinatorial Optimization and Applications*, 281–295 (Springer).

Suzuki H, Minato S (2018) Fast enumeration of all pareto-optimal solutions for 0-1 multi-objective knapsack problems using ZDDs. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 101(9):1375–1382.

Tjandraatmadja C, van Hoeve WJ (2019) Target cuts from relaxed decision diagrams. *INFORMS Journal on Computing* 31(2):285–301.

Tjandraatmadja C, van Hoeve WJ (2020) Incorporating bounds from decision diagrams into integer programming. *Mathematical Programming Computation* 13:225–256.

van den Bogaerdt P, de Weerdt M (2018) Multi-machine scheduling lower bounds using decision diagrams. *Operations Research Letters* 46(6):616–621.

van Hoeve WJ (2020) Graph coloring lower bounds from decision diagrams. *Proceedings of the International Conference on Integer Programming and Combinatorial Optimization*, 405–418.

van Hoeve WJ (2021) Graph coloring with decision diagrams. *Mathematical Programming* 1–44.

Verhaeghe H, Lecoutre C, Schaus P (2018) Compact-MDD: Efficiently filtering (s)MDD constraints with reversible sparse bit-sets. *Proceedings of the International Joint Conference on Artificial Intelligence*, 1383–1389.

Verhaeghe H, Lecoutre C, Schaus P (2019) Extending compact-diagram to basic smart multi-valued variable diagrams. *Proceedings of the International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 581–598.

Vion J, Piechowiak S (2018) From mdd to bdd and arc consistency. *Constraints* 23(4):451–480.

Wegener I (2000) *Branching programs and binary decision diagrams: theory and applications* (Society for Industrial and Applied Mathematics), 1st edition.

Wolsey LA, Nemhauser GL (1999) *Integer and combinatorial optimization* (John Wiley & Sons), 1st edition.

Xue Y, van Hoeve WJ (2019) Embedding decision diagrams into generative adversarial networks. *Proceedings of the International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 616–632.