Constraints (2016) 21:533–556
DOI 10.1007/s10601-016-9239-9

Theoretical insights and algorithmic tools for decision diagram-based optimization

David Bergman[1] **· Andre A. Cire**[2]

Published online: 3 March 2016

Abstract The use of decision diagrams has recently emerged as a viable general solution approach for solving discrete optimization problems. The decision diagram data structure is used to explicitly represent, either exactly or approximately, the set of feasible solutions to a given problem. Techniques based on decision diagrams have been successfully used on a diverse set of applications, ranging from scheduling to combinatorial optimization, and have often outperformed commercial state-of-the-art constraint programming and integer programming technology. Lacking, however, is a thorough theoretical investigation into the quality of approximate decision diagrams, as well as the development of structured techniques for tightening relaxation bounds provided by approximate decision diagrams, analogously to how cutting-planes are used in integer programming. This paper provides an analysis of the strength of approximate decision diagrams, as well as the description of several bound-tightening procedures for problems with linear objective functions.

Keywords Decision diagrams · Discrete optimization · Constraint satisfaction problems · Constraint optimization problems

✉ David Bergman
david.bergman@business.uconn.edu

Andre A. Cire
acire@utsc.utoronto.ca

[1] School of Business, University of Connecticut One University Place, Stamford, CT 06901, USA

[2] Department of Management, University of Toronto Scarborough 1265 Military Trail, Toronto, ON M1C-1A4, Canada

1 Introduction

A *decision diagram* (DD) is a graphical structure that provides an explicit representation of solutions to a discrete optimization problem. Recently, a number of optimization techniques based on DDs have substantially improved the state of the art in a variety of problems in constraint programming and operations research. Successes of this approach include solving and substantially tightening the best known bounds for maximum cut benchmark problems [6, 11], improving propagation for constraint-based scheduling techniques and closing previously open TSPLIB instances [15], stronger filtering methods for global constraints [4], the development of generic heuristics for binary programs that scale for large problem sizes [7], scalable parallelization techniques [8], improved Lagrangian relaxation methods [9], and many others [5, 7, 10, 18, 20–22]. The combined set of computational tools and theoretical insights provided by DDs is denoted here by *decision diagram-based optimization* (DDO).

At the heart of DDO is the concept of *limited-size approximate DDs*. Approximate DDs have been introduced as a way to cope with the (typical) exponential-size DDs that exactly represent the set of solutions to a problem. Two types of approximations are commonly used: *Relaxed DDs*, which provide relaxation bounds by over-approximating the solution set [10, 19]; and *restricted DDs*, which provide heuristic solutions by under-approximating the solution set [7]. Both give objective function bounds and are combined to develop complete branch-and-bound solution methods [6, 11]. Furthermore, relaxed DDs are also used in constraint programming as a *constraint store* to enhance the communication between global constraints [1], and have shown to speedup search by orders of magnitude on scheduling problems [4, 15]. Despite the recent success, a theoretical investigation into worst-case bounds on the quality of relaxed and restricted DDs and a thorough approach to bound tightening is missing.

In this paper we present three contributions as a first step towards obtaining theoretical insights on the properties of DD-based methods. First, we demonstrate an upper bound of $O(2^{\frac{2}{3}n})$ on the size of the exact DD for *any* binary constraint satisfaction problem. Second, for any pre-specified diagram size, we prove the existence of relaxed BDDs with a certain degree of *quality*. In this context, the quality of a DD is defined as the ratio of the number of feasible solutions to a discrete problem over the number of solutions encoded in a relaxed BDD. We also show a similar result with respect to restricted BDD. Finally, as a third contribution, we show the existence of approximate BDDs (both relaxed and restricted) of limited width having a given objective function bound for any discrete optimization problem.

In addition to the theoretical perspectives, we also provide three algorithmic contributions in the form of tools that improve the bounds of relaxed DDs. This is similar to the concept of *cutting planes* that are crucial to integer programming (IP) solvers, in which linear programming (LP) relaxations bounds are iteratively improved through valid inequalities, such as Gomory-Chvatál cuts. Similarly, we modify relaxed DDs iteratively to remove solutions that are infeasible with respect to some constraint that must hold for all solutions to the discrete optimization problems.

In particular, the first technique described is a longest-path trimming algorithm. The algorithm is shown to be complete and also is proven to increase the width of the DD by at most one in each iteration. Next, the paper describes how one relaxed DD can be used to tighten another relaxed DD, and worst-case bounds on how large the resulting relaxed DD is proven. This algorithm is then used to derive a novel complete IP solution method which is polynomial in n for a fixed number of constraints.

Finally, a *value extraction* algorithm, which removes solutions of a certain value, is discussed. One can, for example, use an LP to obtain a tighter relaxation bound than that provided by the relaxed DD, but then use the bound with the relaxed DD to initialize the value extraction procedure, and hence further tighten the relaxation bounds.

The present paper focuses on the case of linear objective functions. Non-linear objective functions, for example quadratic functions, can also be handled by approximate DDs [6], but the investigation of bound-tightening procedures for such problems is left for future research.

The remainder of the paper is organized as follows. Section 2 provides notation and an introduction to DDs. Section 4 proves a new upper bound on the size of DDs for arbitrary discrete optimization problems 3. Section 4 describes (and proves the correctness and useful properties of) an algorithm that can be used to complement a DD, with results that will be used throughout the remainder of the paper. Section 5 discusses the proofs relating to the existence of approximate DDs with particular bounds and quality, and the proof of the worst-case width reduced BDDs. Finally, Section 6 discusses the bound tightening procedures and a conclusion is provided in Section 7.

2 Fundamentals of decision diagrams for optimization

For the purposes of this work we express general discrete problems as *constraint satisfaction problems* (CSPs). A CSP $\mathcal{P} = \langle X, D, \mathcal{C}\rangle$ is defined as a set of n *variables* $X = \{x_1, \dots, x_n\}$ each associated with a finite *domain* $D(x_j) \subseteq D$, together with a set of *constraints* $\mathcal{C}$ establishing *relations* among the variables. In particular, a constraint $C_i \in \mathcal{C}$ with *arity* k and *scope* $\langle x_{i1}, \dots, x_{ik}\rangle$ can be alternatively viewed as a Boolean function $C_i : D(x_{i1}) \times \cdots \times D(x_{ik}) \to \{0, 1\}$, which evaluates to 1 when it is satisfied, and 0 otherwise. The remainder of the paper assumes all variables domains are *binaries*, i.e. $D = \mathbb{B}$.

A solution $x' \in \mathbb{B}^n$ assigns values to each variable in X and it is said to be *feasible* if $C(x') = 1$ for all $C \in \mathcal{C}$, and *infeasible* otherwise. The set of feasible solutions to a CSP is denoted by $\mathrm{Sol}(\mathcal{P})$. Solving a CSP typically corresponds to finding one feasible solution to the problem, or in some cases enumerating all of $\mathrm{Sol}(\mathcal{P})$.

A *constraint optimization problem* (COP) associates a CSP $\mathcal{P}$ with an *objective function* $f : \mathbb{B}^n \to \mathbb{R}$. The goal for a COP $(\mathcal{P}, f)$ is to find an *optimal solution* in $\mathcal{P}$ that maximizes f, i.e. a solution x' such that $x' \in \mathrm{Sol}(\mathcal{P})$ and $f(x') \geq f(x'')$ for any $x'' \in \mathrm{Sol}(\mathcal{P})$. In the remainder of the paper we assume that the objective function f is *linear*, $f(x) = \sum_{i=1}^{n} c_i x_i$, and that each objective function coefficient c_i is integral. Note that since all variables are binaries, every additively separable objective function can be equivalently expressed as a linear function.

2.1 Decision diagrams

Decision diagrams take many forms in the literature [27]. We will particularly focus on *binary decision diagrams* in the context of this paper. A BDD $B = (n, U, A, \ell, d, v)$ is a *layered-acyclic digraph* composed of node set U and arcs A. The mapping $\ell : U \to \{1, 2, \dots, n + 1\}$ partitions the nodes into $n + 1$ *layers* $L_i := \{u \in U : \ell(u) = i\}, i = 1, 2, \dots, n + 1$. Layers L_1 and L_{n+1} have cardinality 1 with the single nodes in these layers called the *root* $\mathbf{r}$ and *terminal* $\mathbf{t}$, respectively. $d : A \to \{0, 1\}$ defines the *domain* of each arc $a \in A$, and $v : A \to \mathbb{Z}$ defines the *value* of each arc (v may not be defined in which case $v(a) = 0$ for all a). Each arc $a \in A$ has *tail* $t(a)$ and *head* $h(a)$, respectively, where

$t(a), h(a) \in U$ denote the nodes that it connects. It is assumed that $\ell(h(a)) = \ell(t(a)) + 1$ so that each arc connects nodes in adjacent layers. The *width* of a layer $w(L_i)$ is the number of nodes at that layer, $|L_i|$, and the *width* of B is defined by $w(B) := \max_{i \in \{1,\ldots,n+1\}} w(L_i)$. The *size* $|B|$ of a BDD is the number of nodes in the diagram.

A BDD represents a set of binary vectors in the following way. Each arc-specified path $p = (a_1, a_2, \ldots, a_k)$ represents the vector

$$x(p) = (d(a_1), d(a_2), \ldots, d(a_k)) .$$

Any path from $\mathbf{r}$ to $\mathbf{t}$ thereby corresponds to a vector in $\mathbb{B}^n$. Let $\mathcal{H}(B)$ be the set of arc-directed paths from $\mathbf{r}$ to $\mathbf{t}$. Define $\text{Sol}(B)$ as the set of binary vectors (also called *solutions*) corresponding to arc-directed $\mathbf{r} - \mathbf{t}$ paths:

$$\text{Sol}(B) = \{x \in \mathbb{B}^n : x = x(p) \text{ for some } p \in \mathcal{H}(B)\}.$$

The *value* $\text{val}(p)$ of a path p (and of the solution $x(p)$) is $\text{val}(p) = \text{val}(x(p)) = \sum_{a \in p} v(a)$. Let $z(B)$ be the length of the longest path in B and $p(B)$ any path in B realizing this bound.

BDDs can be manipulated through the well-known APPLY operation so as to create new BDDs that are logically related to the previous ones [12]. This function, in general, can handle disjunction/conjunction of BDDs, thereby providing the capability to complement, extract solutions, and many other tasks. With suitable input BDDs, the output of APPLY can be the same as the output of the algorithms described in this paper. Nonetheless, the algorithms presented in this paper work directly on the input BDD and so do not require auxiliary BDDs that are created in the general procedures. Additionally, the bounds on quality of approximate BDDs proven here cannot be easily derived using the APPLY operations alone.

2.2 Exact and approximate BDDs

A BDD B with $n + 1$ layers can be used to encode a set of solutions to a COP $(\mathcal{P}, f)$. In particular, B is an *exact BDD* for $\mathcal{P}$ if $\text{Sol}(B) = \text{Sol}(\mathcal{P})$ and $\forall p \in \mathcal{H}(B), \text{val}(p) = f(x(p))$. In other words, B is exact if there exists a bijection between paths in B and feasible solutions to $\mathcal{P}$, and the length of each path in B coincides with the objective function value of the solution to which it corresponds to. The optimal solutions to $\mathcal{P}$ thereby coincide with the longest $\mathbf{r} - \mathbf{t}$ paths with respect to the arc values $v(a)$.

Although a BDD can compactly represent an exponential number of solutions to a COP, the diagram will in general grow exponentially large as the number of variables increases. One proposed technique for dealing with this issue is the use of *approximate BDDs* which come in two forms: *relaxed BDDs* and *restricted BDDs*. A BDD is *relaxed* if $\text{Sol}(B) \supseteq \text{Sol}(\mathcal{P})$ and, for all $p \in \mathcal{H}(B)$ for which $x(p)$ is feasible, $\text{val}(p) \geq f(x(p))$. A BDD is *restricted* if $\text{Sol}(B) \subseteq \text{Sol}(\mathcal{P})$ and, for all $p \in \mathcal{H}(B)$, $\text{val}(p) \leq f(x(p))$. This enables the identification of *objective function bounds*: the longest path in a relaxed BDD provides an upper bound on the optimal value of $\mathcal{P}$, while the longest path in a restricted BDD provides a lower bound to such value.

As an illustration, consider the following COP, formulated as an integer linear program:

$$\begin{aligned} &\max f(x) = 3x_1 + 4x_2 + 2x_3 + 2x_4 + 7x_5 \\ &\text{subject to} \\ &\quad x_1 + x_2 + x_3 \leq 1 \\ &\quad x_2 + x_3 + x_4 \leq 1 \\ &\quad x_4 + x_5 \leq 1 \\ &\quad x_j \in \{0, 1\} \qquad j = 1, \ldots, 5 \end{aligned} \tag{$\mathcal{P}^1$}$$

Figure 1 depicts (a) an exact BDD, (b) a relaxed BDD, and (c) a restricted BDD for $\mathcal{P}^1$. Arcs a depicted as solid lines have arc domain $d(a) = 1$ and arcs depicted with dashed arcs have arc domain $d(a) = 0$. The arc costs are indicated next to each arc. **r** is drawn on the top and **t** is drawn at the bottom, so arcs are directed downwards.

The path with the longest path in the exact BDD has value 11 and corresponds to solution $(0, 1, 0, 0, 1)$, an optimal solution. In the relaxed BDD, solution $(0, 1, 1, 0, 1)$ (an infeasible solution) has the longest path value among all $\mathbf{r} - \mathbf{t}$ paths and provides a relaxation bound of 13 (an upper bound because the problem is stated as a maximization problem). It can be verified that for each feasible solution the value of the path is an upper bound on the objective function value of the solution it represents. Alternatively, in the restricted BDD, the path with the largest value corresponds to solution $(1, 0, 0, 0, 1)$ and provides a lower bound of 10 on the objective function.

Restricted BDDs [7, 11] are obtained through top-down compilation schemes in which nodes are heuristically deleted during construction to keep the diagram within a limited size. Relaxed BDDs required more involved operations [5, 10, 11]. Namely, compilation methods for relaxed BDDs typically *merge* nodes during a top-down construction in a way that all feasible solutions are preserved, but some infeasible solutions are added to cope with the combinatorial growth.

A novel branching scheme *within* relaxed BDDs, coupled with the bounds provided by relaxed and restricted BDDs, provide the basis for *BDD-based optimization*. The technique has been used to close open instances of benchmark problems for classical COPs and has proven in many cases to outperform state-of-the-art commercial linear integer programming solvers [6].

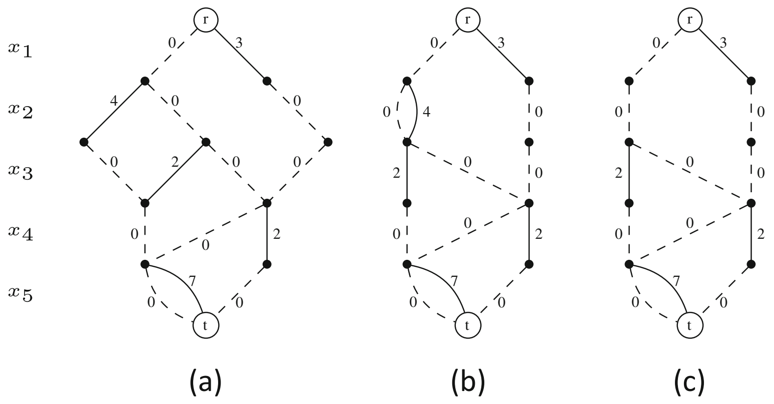

Fig. 1 **a** Exact BDD, **b** relaxed BDD, and **c** restricted BDD for COP $\mathcal{P}^1$

Given nodes $u, v \in U$ in B, let $B[u, v]$ be the BDD with root u and terminal v obtained from B by removing all nodes and arcs not contained in any path from u to v. If B is exact and the layers of $B[u, v]$ are associated with the same variables as B, the set $\text{Sol}(B) \subseteq \mathbb{B}^{\ell(v)-\ell(u)}$ corresponds to the projection of solutions $\text{Sol}(B)$ onto variables $x_{\ell(u)}, \ldots, x_{\ell(v)-1}$. If B is a relaxation or a restriction, then $\text{Sol}(B)$ corresponds to either a superset or subset of such projection, respectively.

Similarly, for any $u \in U$, define $B[u]$ to be the BDD obtained by removing from B all nodes and arcs that do not lie on paths from $\mathbf{r}$ to u or from u to $\mathbf{t}$. Observe that, for any layer L_i, the collections of BDDs $B[u]$, $u \in L_i$, partition the set of solutions $\text{Sol}(B)$ into the sets $\{\text{Sol}(B[u])\}_{u \in L_i}$, since all solutions must necessarily pass through some node in L_i.

B is said to be *reduced* if, for any $u, u' \in U$ with $\ell(u) = \ell(u')$, the condition $\text{Sol}(B[u, \mathbf{t}]) = \text{Sol}(B[u', \mathbf{t}])$ holds if and only if $u = u'$. This implies that the set of solution completions emanating from a node on any given layer of the BDD is unique. Equivalently, a BDD B is reduced if all isomorphic sub-BDDs of B are superimposed, i.e. no two sub-BDDs represent the same set of solutions. It has been shown that, for a given ordering of the variables, there exists a unique reduced BDD for any *Boolean function* — implying the same for any CSP. A BDD B can be reduced in time $O(|B| \cdot log|B|)$ [27]. Reduced BDDs are important from a theoretical and algorithmic point of view because, for a given ordering of the problem variables, this unique BDD is the most compact BDD representation of a set of solutions.

3 Size of exact BDDs

In this section we address the question of determining bounds on the size of an exact BDD for constraint satisfaction problems, which will proven to be important to assess the quality of approximate BDDs. Namely, we wish to establish lower and upper bounds on the width (and hence the size) of an exact BDD for any arbitrary CSP $\mathcal{P}$. Note that bounding the width bounds the size since $|B| = O(n\, w(B))$.

The problem of bounding the size of a BDD is directly linked to that of *variable ordering*, i.e. the bijection between layers L of an exact BDD and the variables X of the CSP it represents. Previous work has shown that different orderings can yield drastically different BDDs for combinatorial optimization problems [5]. Since there is an unique reduced BDD for a particular variable ordering [27], searching for small BDDs amounts to searching for good variable orderings for a problem.

3.1 Lower bounds

We first provide a discussion on the lower bound of the size of a BDD. Previously, Hosaka et al. [23] demonstrated that there are *threshold Boolean functions*[1] for which the size of the BDD has a lower bound of $\Omega(2^{\frac{n}{2}})$. Thus, in general it is not possible to obtain BDDs that are polynomially-sized on n or on the constraint set size, $|\mathcal{C}|$.

Alternatively, this result can also be seen as a consequence of a theorem by Rothvoß [25], who shows that there are 0-1 polytopes which require exponential-size extended

[1]Given scalars a and b, a threshold Boolean function evaluates to 1 if $ax \leq b$ and 0 otherwise.

formulations; i.e., have *exponential extension complexity*. Since an exact BDD for a COP induces an extended formulation by writing the longest path problem from **r** to **t** as an LP model [2], in general there does not exist an ordering of the variables for which the exact BDD has polynomial size on n. Rothvoß later produced explicit models for the matching problem and independent set problem with exponential extension complexity [26]. This implies that, for any variable ordering, the exact BDD for such instances have exponential size as well.

3.2 Upper bounds

A trivial upper bound on the width of a BDD for a CSP is $O(2^n)$, which follows because $w(L_{i+1}) \leq 2 \cdot w(L_i)$ for any layer L_i, $i \leq n$.

We now demonstrate in Theorem 1 a new result that tightens this bound substantially. In particular, the theorem holds for *any* ordering and for any arbitrary CSP.

Theorem 1 *The width w(B) of any reduced BDD B satisfies w(B)* $\leq 2^{\kappa(n)}$,

$$\kappa(n) := \frac{\mathsf{W}(2^{n+1}\ln(2))}{\ln(2)}$$

and $\mathsf{W}(x)$ *is the Lambert W function (or product logarithm function), defined as the inverse function of* $f(x) = xe^x$ *(or, equivalently,* $\mathsf{W}(xe^x) = x$ *).*

Proof Fix layer ℓ. We first claim that $w(L_\ell) \leq \min\left\{2^\ell, 2^{2^{n-\ell+1}}\right\}$. The first term in the minimization holds because each node has at most 2 outgoing arcs, and so $w(L_{\ell+1}) \leq 2 \cdot w(L_\ell)$. Since $w(L_\ell) = 1$, the bound follows.

For the second term, more care is needed. First, for $u \in L_j$ and $k \in \{0, 1\}$, define $o(u)_k$ to be the node in layer L_{j+1} which is connected via a k-arc emanating from node u, where $o(u)_k = \hat{0}$ if no k-arc emanates from u. Let $o(u) = (o_0(u), o_1(u))$. Note that, if $o(u) = o(u')$ for any pair of nodes $u, u' \in L_j$, then the BDD is not reduced because these nodes can be merged. Therefore, for the remainder of the proof it is assumed that the BDD is reduced so that the set of vectors $o(u)$ in each layer are unique. The proof proceeds via two claims.

Claim 1 $|L_\ell| \leq |L_{\ell+1}| \cdot (|L_{\ell+1}| + 2)$. Since $o(u)$ is unique for each node $u \in L_\ell$, we count the number of distinct vectors $o(u)$ that are possible in L_ℓ. Note that $o_k(u) \in \{\hat{0}\} \cup L_{\ell+1}$ for each $k \in \{0, 1\}$, hence there are $|L_{\ell+1}| + 1$ options for each coordinate k, with the only unacceptable vector being $\{\hat{0}, \hat{0}\}$. Thus, we have $(|L_{\ell+1}| + 1) \cdot (|L_{\ell+1}| + 1) - 1 = |L_{\ell+1}| (|L_{\ell+1}| + 2)$ possible nodes in layer L_ℓ.

Claim 2 $L_\ell| \leq 2^{2n-\ell+1} - 1$ *for* $\ell \leq n$. The proof proceeds by backward induction on ℓ. For the basis case $\ell = n$, we have that $|L_{n+1}| = 1$, since the BDD ends with the terminal node **t**. From Claim 1, it follows that

$$|L_n| \leq 1 \cdot (1 + 2) = 2^{2n-n+1} - 1 = 3,$$

which is the maximum number of nodes at layer L_n in a reduced BDD. Fix now $\ell' \in \{n - 1, n - 2, \ldots, 1\}$ and by induction hypothesis, suppose the claim is true for

$\ell \in \{n-1, n-2, \dots, \ell'+1\}$. It follows that $|L_{\ell'+1}| \leq 2^{2^{n-(\ell'+1)+1}} - 1 = 2^{2^{n-\ell'}} - 1$. Using Claim 1, we have

$$\begin{aligned}
|L_{\ell'}| \leq |L_{\ell'+1}| \cdot \left(|L_{\ell'+1}| + 2\right) &\leq \left(2^{2^{n-\ell'}} - 1\right) \cdot \left(2^{2n-\ell'} - 1 + 2\right) \\
&= \left(2^{2^{n-\ell'}} - 1\right) \cdot \left(2^{2^{n-\ell'}} + 1\right) \\
&= 2^{2^{n-\ell'}+2^{n-\ell'}} - 1 \\
&= 2^{2^{n-\ell'+1}} - 1,
\end{aligned}$$

as desired. It therefore follows that

$$|L_\ell| \leq \min\left\{2^\ell, 2^{2^{n-\ell+1}}\right\}. \tag{1}$$

To find the layer ℓ^* with the largest possible width, we equate the expressions:

$$2^{\ell^*} = 2^{2^{n-\ell^*+1}}$$

The left-hand side is increasing in ℓ^* and the right-hand side is decreasing in ℓ^*. Thus, the maximum must occur at (approximately) the value ℓ^* such that

$$\ell^* = 2^{n-\ell^*+1},$$

or, equivalently,

$$\ell^* + \log_2(\ell^*) = n + 1.$$

Since $\log_2(\ell^*) = \ln(\ell^*)/\ln(2)$, the expression above can be rewritten as follows:

$$\begin{aligned}
\ell^* + \frac{\ln(\ell^*)}{\ln(2)} &= n + 1 \\
\ell^*\ln(2) + \ln(\ell^*) &= (n+1)\ln(2) \\
e^{\ell^*\ln(2)+\ln(\ell^*)} &= e^{(n+1)\ln(2)} \\
e^{\ell^*\ln(2)}e^{\ln(\ell^*)} &= e^{\ln(2)^{n+1}} \\
\ell^* e^{\ell^*\ln(2)} &= 2^{n+1} \\
\ell^*\ln(2)e^{\ell^*\ln(2)} &= 2^{n+1}\ln(2)
\end{aligned}$$

Finally, from the definition of the Lambert W function,

$$\begin{aligned}
\mathsf{W}\left(\ell^*\ln(2)e^{\ell^*\ln(2)}\right) &= \mathsf{W}\left(2^{n+1}\ln(2)\right) \\
\ell^*\ln(2) &= \mathsf{W}\left(2^{n+1}\ln(2)\right) \\
e^{\ell^*} &= \frac{\mathsf{W}\left(2^{n+1}\ln(2)\right)}{\ln(2)} = \kappa(n).
\end{aligned}$$

This implies that, for any BDD B, $w(B) \leq 2^{\kappa(n)}$. □

The value $k_{w'}$ in the following lemma will be used for proving future existential results.

Lemma 1 *For any CSP and for any variable ordering, the number of layers in the reduced exact BDD having width larger than w' is bounded above by*

$$k_{w'} := n - \log_2(w') - \log_2(\log_2(w')). \tag{2}$$

Proof This follows directly from (1). □

Note that the proof of Theorem 1 gives a construction procedure of a reduced BDD which achieves a width of $2^{\kappa(n)}$, and therefore the upper bound is tight. Starting from the root node and until layer $\lfloor \kappa(n) \rfloor$, construct a node with two outgoing arcs that will be directed to unique nodes on the subsequent layer. Then, from the terminal node up to layer $\lfloor \kappa(n) \rfloor$, build the maximum number of nodes on the previous layer having distinct pairs o as specified in the proof.

However, a question for future research is whether or not the binary set represented by such a BDD can be represented by a another smaller BDD having a different variable ordering. To illustrate this on a classical example, consider the set below, for n even:

$$X'_n = \{x \in \mathbb{B}^n : \; 2^{\frac{n}{2}-1}x_1 + 2^{\frac{n}{2}-2}x_2 + \ldots + 2^0 x_{\frac{n}{2}} + 2^0 x_{\frac{n}{2}+1} + \ldots \\ + 2^{\frac{n}{2}-2}x_{n-1} + 2^{\frac{n}{2}-1}x_n = 2^{\frac{n}{2}} - 1\}.$$

For $n = 6$ the equation on X'_n above becomes $4x_1 + 2x_2 + x_3 + x_4 + 2x_5 + 4x_6 = 7$. Using the lexicographic ordering of the variables, the BDD width for the set X'_n is $2^{\frac{n}{2}}$. This is illustrated in the BDD depicted in Fig. 2(a). If, however, the ordering $x_1, x_n, x_2, x_{n-1}, \ldots, x_{\frac{n}{2}}, x_{\frac{n}{2}+1}$ is used, the BDD has width 2, as depicted in the BDD in Fig. 2(b).

4 Size of complement BDDs

Our next results concern the size of *complement BDDs*. A BDD $\neg B$ is a *complement* of another BDD B when $x \in \text{Sol}(B)$ if and only if $x \notin \text{Sol}(\neg B)$ (i.e., $\text{Sol}(\neg B) = \mathbb{B}^n \setminus \text{Sol}(B)$). In the context of Boolean logic, complementing BDDs is a constant-time operation since

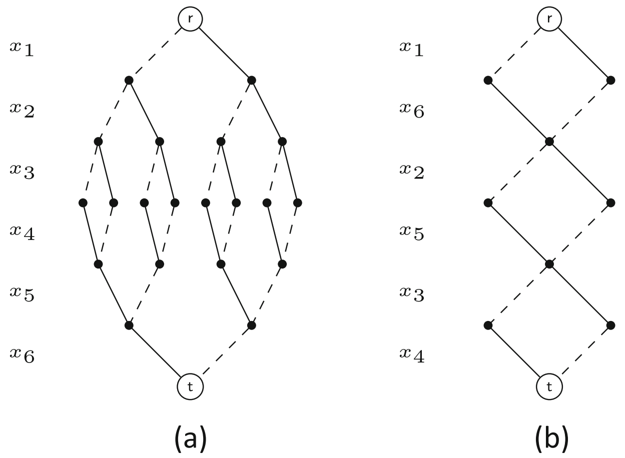

Fig. 2 Two BDDs for the set X'_n

typically BDDs encode both the *true* and *false* assignments of a Boolean formula [12]. In our representation, however, we only encode solutions which evaluate to a *true* state, as in optimization we are primarily interested in operations over the exact or approximate feasible solution set, e.g. to compute bounds on the optimal solution value. Complementing a BDD for a COP is therefore a more involved operation since feasibility of solutions are not explicitly distinguished in the BDD.

It is of important theoretical value to be able to compute and bound the size of complement BDDs. In particular, complement BDDs are used here as a tool to prove existential theorems of Section 5, and can be also applied to negate *table* constraints, typical in constraint programming models [13].

The main result of this section is that, under the same ordering of variables, one can obtain a complement BDD $\neg B$ for a reduced BDD B such that the width of B and $\neg B$ differ by at most one. This implies that, given an exact BDD B for a CSP $\mathcal{P}$, there exists a BDD encoding exactly the infeasible solutions to $\mathcal{P}$ with size $O(|B|)$. We formalize this result in Theorem 2:

Theorem 2 *For any BDD B, there exists a complement BDD $\neg B$ for which*

$$|w(\neg B) - w(B)| \leq 1$$

and hence $||\neg B| - |B|| \leq n$.

Theorem 2 will be proved constructively from the algorithm that follows. A few preliminary definitions are in order. A node $u \in L_j$ is said to be a *tautologous node* if $|\mathrm{Sol}(B[u, \mathbf{t}])| = \mathbb{B}^{n-\ell(u)+1}$, i.e. the partial solutions obtained by $\mathbf{r}-u$ paths can be extended with any 0-1 completion of variables $x_{\ell(u)}, \ldots, x_n$. Correspondingly, a *tautologous path* is a node-specified path p consisting only of tautologous nodes and ending at the terminal node. The left-most node in layer L_5 in each of the BDDs in Fig. 1 is a tautologous node. The paths consisting of this node and $\mathbf{t}$ are examples of tautologous paths.

Lemma 2 *In any reduced BDD, there is at most one tautologous node per layer and these nodes, together, create the only (maximal) tautologous path in B.*

Proof Suppose $u, u' \in L_i$ are both tautologous nodes. Then $\mathrm{Sol}(B[u, \mathbf{r}]) = \mathrm{Sol}(B[u', \mathbf{r}])$, contradicting that B is reduced. Therefore there is at most one tautologous node per layer in B.

Let layer L_j be the highest layer (the layer with the lowest index) in B with a tautologous node, and let u be this node. If this is the terminal node then the path consisting only of $\mathbf{t}$ is the one tautologous path in B. Otherwise, by definition, u has two outgoing arcs, one for each arc-domain value, and each node u' for which $(u, u') \in A$ must also be tautologous. There can be at most one such node on layer L_{j+1}. Continuing in this fashion until reaching $\mathbf{t}$ identifies the unique maximal tautologous path. □

Algorithm 1 provides the pseudocode for an algorithm that can be used to complement a *reduced* BDD $B = (n, U, A, \ell, d)$, producing complement BDD $\neg B = (n, U, E, \ell, g)$ (the arc values are omitted and can be calculated after building $\neg B$, if necessary). The algorithm assumes that the tautologous nodes of B are pre-identified, which can be done in $O(|B|)$ by a simple breadth-first search traversal of B starting at $\mathbf{t}$. Namely, the search starts by marking node $\mathbf{t}$ at tautologous, and proceed to visit nodes in the previous layers. A node u is marked tautologous when visited if it has two out-going arcs ending at a tautologous node.

The algorithm works as follows. It initializes the edge set of $\neg B$ with an empty set ($E \leftarrow \emptyset$), which will be associated with arc domains g. Next, it traverses B identifying the tautologous node of a layer L_i as t_i; if such a node does not exist, it artificially creates one to be added to the node set of $\neg B$ later if necessary (lines 6 to 10). The algorithm then traverses the nodes of B in a top-down manner (i.e., from **r** to **t**, layer-by-layer), adding arcs to E according to three cases. If an examined node is tautologous (line 13), the out-going arcs from u are preserved in $\neg B$. If a node u has only one out-going arc a (line 18), an arc with an opposite arc domain is created in $\neg B$ and directed from u to the tautologous node on the next layer. In this case, if the head $h(a)$ of a is not a tautologous node, this arc is preserved and directed to the same node as it is in B. Finally, in the case that the node has two out-going arcs and it is not tautologous (line 26), only those out-going arcs that are directed at non-tautologous nodes are added to $\neg B$. The algorithm ends by adding the tautologous nodes to $\neg B$ which have some paths traversing them.

Algorithm 1 Complement BDD

```
 1: procedure COMPLEMENT($B$) ▷ $B = (n, U, A, \ell, d)$ is a reduced BDD for $\mathcal{P} = \langle X, \mathbb{B}^n, \mathcal{C} \rangle$;
 2:                                                        assumes $0 < |\text{Sol}(B)| < \mathbb{B}^n$
 3:
 4:     $E \leftarrow \emptyset$                                  ▷ Initialization of the edge set of $\neg B$
 5:     $t_{n+1} = \mathbf{t}$                         ▷ Initialize the terminal node as a taut. node
 6:     for $i = 1, \ldots, n$ do
 7:         if $\nexists$ taut. node in $L_i$ then      ▷ ‘taut.’ will be used to abbreviate ‘tautologous’
 8:             create node $t_i, \ell(t_i) = i$             ▷ will be used as taut. node if necessary
 9:         else
10:             label taut. node in $L_i$ as $t_i$
11:     for $i = 1, \ldots, n$ do
12:         for all $u \in L_i$ do
13:             if $u$ is a taut. node then                          ▷ add both out-going arcs to
14:                                                                 taut. node on next layer
15:                 create arc $e' = (u, t_{i+1}), g(e') = 0$
16:                 create arc $e'' = (u, t_{i+1}), g(e'') = 1$
17:                 $E \leftarrow E \cup \{e', e''\}$
18:             else if $u$ has only one out-going arc $a = (u, v) \in A$ then
19:                 create arc $e' = (u, t_{i+1}), g(e') = 1 - d(a)$         ▷ this arc cannot lead to
20:                                                                a feasible solution to $\mathcal{P}$
21:                 $E \leftarrow E \cup \{e'\}$
22:                 if $v$ is not a taut. node then
23:                     create arc $e'' = (u, v), g(e'') = d(a)$       ▷ need to continue exploring
24:                                                                               ▷ this path
25:                     $E \leftarrow E \cup \{e''\}$
26:             else                                                ▷ $u$ has two out-going arcs
27:                 let $a' = (u, v'), a'' = (u, v'')$                ▷ at most one is a taut. node;
28:                                                              otherwise $u$ is a taut. node
29:                 if $v'$ is not a taut. node then
30:                     create arc $e' = (u, v'), g(e') = d(a')$
31:                     $E \leftarrow E \cup \{e'\}$
32:                 if $v''$ is not a taut. node then
33:                     create arc $e'' = (u, v'), g(e'') = d(a'')$
34:                     $E \leftarrow E \cup \{e''\}$
35:     $\neg U \leftarrow U \cup \{t_i : i = 1, \ldots, n+1, \exists\, e \in E$ with $h(e) = t_i\}$ ▷ add to $U$ only those taut.
36:                                                                     nodes that have paths
37:                                                                         directed to them
38:     return $\neg B = (n, \neg U, E, \ell, g)$
```

Theorem 3 *Algorithm 1 is correct and runs in $O(|B|)$ time.*

Proof The proof requires showing

1: an assignment of values to variables corresponding to some path in B does not correspond to any path in $\neg B$, and
2: any assignment of values to variables which does not correspond to a path in B does correspond to a path in $\neg B$.

Take some path $p = (a_1, \ldots, a_n)$ of arcs in B and let $(\mathbf{r} = u_1, u_2, \ldots, u_n, u_{n+1} = \mathbf{t})$ be the nodes on p. For condition **1** it suffices to show that $x(p) \notin \text{Sol}(\neg B)$. Follow p until it hits a tautologous node u_i. By construction, since u_i is the first tautologous node visited, each arc will be preserved in $\neg B$ until u_{i-1}. When processing u_{i-1} the algorithm breaks into cases based on the out-degree of u_{i-1}. In either case, arc x_{i-1} which is directed in B to a tautologous node u_i is re-created in E, but with the opposite arc-domain. Since any solution of values to the first $i-2$ variables can only lead from $\mathbf{r}$ to exactly one node in layer $i-1$, the solution $(x(p)_1, \ldots, x(p)_{i-2})$ *only* appears in $\neg B$ within $\text{Sol}(\neg B[\mathbf{r}, u_{i-1}])$. Since no arc with arc-domain $d(a)$ is added to E out of u_{i-1}, $x(p)$ cannot appear in $\text{Sol}(\neg B)$.

For condition **2**, let x' be an solution for which $x' \notin \text{Sol}(B)$. Starting from $\mathbf{r}$, follow the sequence of arcs dictated by the values of x'; i.e., take the arc a_1 out of $\mathbf{r}$ with arc-domain x'_1 (if it exists), and then take the arc a_2 out of node $h(a_1)$ with arc-domain x'_2 (if it exists), and so on. At some point, there will be a node u^* reached in some layer L_j, $j < n$ for which no arc with arc-domain x'_j has tail u^*, otherwise $x' \in \text{Sol}(B)$.

First note that this sequence of arcs is preserved in E so that the partial solution up to $x'(p)_{i-1}$ corresponds to some path from $\mathbf{r}$ to u^* in $\neg B$. This follows because no node visited in B in this order is a tautologous node and Algorithm 1 duplicates every arc a when $t(a)$ and $h(a)$ are not tautologous.

Consider then when the algorithm processes u^*. Because there is one arc directed out of u^*, in line 18, there is an arc directed out of u^* in E with arc-domain $x'(p)_{i-1}$ — and this arc is directed to the tautologous node on layer L_i. Therefore, the remaining solutions $x'(p)_i, \ldots, x'(p)_n$ must correspond to the arc-domains of some path from t_i to $\mathbf{t}$ in $\neg B$, so that $x'(p) \in \text{Sol}(\neg B)$ and completing the proof of correctness.

Concerning the running time, the algorithm examines each node in B a constant number of times, and each processing also is done in time $O(1)$. □

Returning to the example from Section 2, Fig. 3(a) depicts the exact BDD (with arc costs omitted) for the feasible set, containing 10 paths corresponding to the 10 feasible solutions. Figure Fig. 3(b) depicts the complement BDD, containing the $2^5 - 10 = 32 - 10 = 22$ paths corresponding to the 22 infeasible solutions.

Equipped with Algorithm 1 the proof of Theorem 2 follows.

Proof - Theorem 2 By Theorem 3, Algorithm 1 produces a complement BDD. During the execution of the algorithm, in creating $\neg B$, at most one node per layer (the tautologous node, if it does not exist and needs to be added as prescribed by the algorithm) is added to the node set. Additionally, at most one node is deleted from the node set (if the tautologous node does not have any arcs directed to it). The bounds on the difference of the widths and the sizes of B and $\neg B$ follow. □

Additionally, the complement BDD will be reduced.

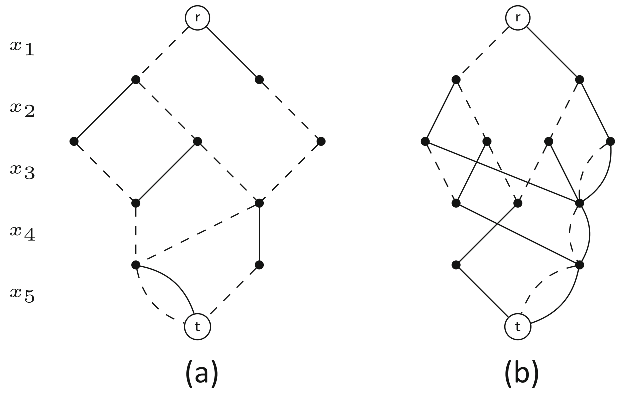

Fig. 3 (a) The exact reduced BDD for COP $\mathcal{P}^1$ and (b) the complement of this BDD

Theorem 4 *The complement BDD created via Algorithm 1 is reduced.*

Proof For $u \in L_j$ and $k \in \{0, 1\}$, recall the definition of $o(u)_k$ as the node in layer L_{j+1} which is connected via a k-arc emanating from node u, where $o(u)_k = \hat{0}$ if no k-arc emanates from u and $o(u) = (o_0(u), o_1(u))$. Note again that if $o(u) = o(u')$ for a pair of nodes $u, u' \in L_j$, then the BDD is not reduced because these nodes can be merged. This condition is sufficient and necessary for a reduced BDD: If no such pair exists, then the set of solution completions emanating from any pair of nodes in a given layer must be distinct.

Proceed by contradiction and assume that, in the BDD created, there are two nodes $u, u' \in L_j$ for which $o' := o(u) = o(u')$. The vector o' can be one of 9 different configurations: A component k may not have an associated arc (i.e., $o'_k = \hat{0}$), may be directed at the tautologous node (i.e., $o'_k = t_{j+1}$), or may be directed at some other node (i.e., $o_k \in \{v, v'\} \subseteq L_{j+1}$). The 9 possibilities are:

$$P1.(v, \hat{0}) \quad P2.(t_{j+1}, \hat{0}) \quad P3.(\hat{0}, v') \quad P4.(\hat{0}, t_{j+1}) \quad P5.(v, v')$$
$$P6.(v, v) \quad P7.(v, t_{j+1}) \quad P8.(t_{j+1}, v) \quad P9.(t_{j+1}, t_{j+1})$$

The claim is that nodes which satisfy P1–P9 in the input BDD are permuted via a permutation σ in Algorithm 1 into nodes that satisfy P1 - P9 in the output BDD, which implies that there cannot be such a pair u, u'. Otherwise, the nodes $\sigma^{-1}(u_1), \sigma^{-1}(u_2)$ would be two nodes in the original BDD that can be merged. In particular, one can verify that $\sigma(\text{P1}) = \text{P7}$, $\sigma(\text{P2}) = \text{P4}$, $\sigma(\text{P3}) = \text{P8}$, $\sigma(\text{P4}) = \text{P2}$, $\sigma(\text{P5}) = \text{P5}$, $\sigma(\text{P6}) = \text{P6}$, $\sigma(\text{P7}) = \text{P1}$, $\sigma(\text{P8}) = \text{P3}$, and $\sigma(\text{P9}) = \text{P9}$. This implies that if two nodes can be merged in the complement BDD that is created, then they could have been merged in the original BDD as well. □

5 Quality of approximate BDDs

Although approximate BDDs have been useful for solving COPs, a theoretical analysis of their quality is lacking in the literature, with only few works done in domain-specific problems [17]. This section provides, to the best of our knowledge, the first-ever published theorems for the existence of approximate BDDs satisfying two types of quality measures for general discrete problems. Our main goal is to provide a first step towards an analytical treatment of relaxed and restricted BDDs.

There are two ways to define the *quality* of an approximate BDD B for $\mathcal{P}$: (1) the ratio of the number of solutions in B with respect to the number of feasible solutions to $\mathcal{P}$, and (2) the value of the upper and lower bound from relaxed and restricted BDDs, respectively, compared to the optimal solution value. The first measure has direct implications in mathematical and constraint programming methods based on decision diagrams, while the second plays an important role in BDD-based branch-and-bound methods. The following sections provide new results for each quality measure.

5.1 Quality measure in terms of solution ratio

The solution ratio of an approximate BDD indicates the *degree of infeasibility* of the BDD. This is measured by the number of infeasible solutions in a relaxed BDD, and the number of missed feasible solutions in a restricted BDD.

For the case of relaxed BDD, this is a relevant quality measure for hybrid approaches that combine BDDs with mixed-integer linear programming and constraint programming. For example, in DD-based methods for scheduling [15], timetable problems [4], and Lagrangian relaxation [9], the solution process is geared towards branching and eliminating arcs that are *inconsistent*, i.e. do not participate in any feasible solution. The smaller the set of infeasible solutions present in $\mathrm{Sol}(B)$, the fewer the number of arcs that need to be eliminated to find a feasible solution; e.g., in an infeasible CSP, ideally no paths would be present - the more paths, the harder the decision. As for restricted BDDs, less feasible solutions missed result in a more diverse set of solutions in a BDD. This measure is critical, e.g., when the decision maker is interested in having many different alternative solutions to apply.

Another reason for having a tight relaxation and restriction is when the objective function contains stochastic elements. In such a case, each realization of the random coefficients can be perceived as a distinct objective function to be optimized over the BDD. The smaller the infeasibility of a relaxed BDD, the better this bound will be across a wide variety of objective functions. In restricted BDDs, a better solution ratio implies that the bounds provided are potentially stronger for many objective functions as opposed to a few ones.

We now present our results, first focusing on restricted BDDs and then on relaxed BDDs.

Restricted BDDs Define the *quality of a restricted BDD B for a CSP $\mathcal{P}$, $q^{\text{res}}(B)$*, to be the ratio of the number of solutions (or paths) in B to the number of feasible solutions to $\mathcal{P}$ (here assuming $|\mathrm{Sol}(B)| > 1$):

$$q^{\text{res}}(B) = \frac{|\mathrm{Sol}(B)|}{|\mathrm{Sol}(\mathcal{P})|}.$$

The higher the quality, the better the restricted BDD. The existence of restricted BDDs with a specified quality are addressed in Theorem 5. Recall from Lemma 1 that k_i defined in (2) bounds the number of layers that can possibly have width larger than i.

Theorem 5 *Let W be the width of an exact reduced BDD B for COP $\mathcal{P}$ (we can assume that W is the minimum over all orderings of the variables). There exists a restricted BDD*

B' with width $\tilde{w}$ having quality

$$q^{\text{res}}(B') \geq \prod_{j=\tilde{w}}^{W-1} \left(\frac{j}{j+1}\right)^{k_{j+1}},$$

(where it is assumed that if the product on the right-hand side is vacuous the term is 1).

Proof If the exact BDD has width less than or equal to $\tilde{w}$ then $B' = B$ satisfies the condition in the statement of the theorem.

Otherwise, suppose first that $\tilde{w} = W - 1$. The expression in the theorem reduces to

$$q^{\text{res}}(B') \geq \left(\frac{\tilde{w}}{\tilde{w}+1}\right)^{k_{\tilde{w}+1}}.$$

By Lemma 1 there are, at most, $k_{\tilde{w}+1}$ layers with width larger than $\tilde{w}$ (in this case by the assumption exactly equal to W) in B. Consider the top layer L_i with this width and select the node u with the minimum number of paths going through this node; i.e., the node $u \in L_i$ for which $|\text{Sol}(B[u])|$ is minimal. Since each path in B goes through one and only one node on each layer, there exists some node on this layer for which $|\text{Sol}(B[u])| \leq \frac{1}{W} \cdot |\text{Sol}(B)|$. Deleting this node from B creates a restricted BDD B' with one less layer of width W containing at least $\frac{W-1}{W} \cdot |\text{Sol}(B)|$. Performing the same deletion for all remaining layers with width $W = \tilde{w}+1$ results in a restricted BDD with at least $\left(\frac{\tilde{w}}{\tilde{w}+1}\right)^{k_{\tilde{w}+1}} \cdot |\text{Sol}(B)|$ paths, and, therefore, quality at least $\left(\frac{\tilde{w}}{\tilde{w}+1}\right)^{k_{\tilde{w}+1}}$, as desired.

For general $\tilde{w} < W$ one can continue the process, taking the restriction of width $W - 1$ with the desired quality bound and iteratively decreasing the width while only sacrificing a $\frac{w'}{w'+1}$ fraction of the width for each w' between $\tilde{w}$ and W. □

By Theorem 1 we know that $\theta(n) := 2^{\kappa(n)}$ is an upper bound on the width of the exact BDD for any CSP. The following corollary therefore follows:

Corollary 1 *For any COP $\mathcal{P}$ there exists a restricted BDD B' with width $\tilde{w}$ having quality*

$$q^{\text{res}}(B') \geq \prod_{j=\tilde{w}}^{\theta(n)-1} \left(\frac{j}{j+1}\right)^{k_{j+1}}.$$

Relaxed BDDs In terms of relaxations, define the *quality of a relaxed BDD B* for a CSP $\mathcal{P}$, $q^{\text{rel}}(B)$, to be the ratio of the number of feasible solutions to $\mathcal{P}$ to the number of solutions (or paths) in B :

$$q^{\text{rel}}(B) = \frac{|\text{Sol}(\mathcal{P})|}{|\text{Sol}(B)|}.$$

The higher the quality, the better the relaxed BDD.

Let $\alpha(\mathcal{P}) = \frac{|\text{Sol}(\mathcal{P})|}{2^n}$ denote the fraction of the Boolean vectors in $\mathbb{B}^n$ which are feasible to $\mathcal{P}$, and let $\beta(P) = 1 - \alpha(\mathcal{P})$. We now address the measure $q^{\text{rel}}(B)$ in Theorem 6.

Theorem 6 *Let W be the width of an exact reduced BDD B for a CSP $\mathcal{P} = \langle X, \mathbb{B}^n, \mathcal{C}\rangle$ (we can assume that W is the minimum over all orderings of the variables). There exists a* *relaxed BDD B' with width $\tilde{w}$ having quality*

$$q^{\mathrm{rel}}(B') \geq \frac{\alpha(\mathcal{P})}{1-\beta(\mathcal{P})\left(\prod_{j=\tilde{w}-1}^{W}\left(\frac{j}{j+1}\right)^{k_{j+1}}\right)}.$$

Before proceeding with the proof, consider the case when $\alpha(\mathcal{P}) = \frac{1}{2}$. The expression in the theorem becomes

$$q^{\mathrm{rel}}(B') \geq \frac{1}{2-\left(\prod_{j=\tilde{w}-1}^{W}\left(\frac{j}{j+1}\right)^{k_{j+1}}\right)}, \tag{3}$$

where the expression in the subtrahend of the denominator relates to the expression in Theorem 5. This expression will be positive, but as it grows the fraction approaches $\frac{1}{2}$, which is the trivial bound one can obtain by using a BDD representing $\mathbb{B}^n$ (which, when reduced, has a width of 1). The tightening of the bound in Theorem 5 would enable a tightening of this expression as well.

Proof of Theorem 6

Let B be an exact BDD for $\mathcal{P}$ with width W. Consider the COP $\mathcal{P}' = \langle X, \mathbb{B}^n, \neg\mathcal{C}\rangle$ where $\neg\mathcal{C}$ is any collection of constraints for which a solution $x \in \mathbb{B}^n$ satisfies $\mathcal{C}$ if and only if it does not satisfy $\neg\mathcal{C}$. Let $\hat{B}$ be the complement of B created via Algorithm 1. $\neg B$ is an exact BDD for $\mathcal{P}'$, with width at most $W+1$. By Theorem 5 there exists a restricted BDD $\bar{B}$ for $\mathcal{P}'$ with $q^{\mathrm{res}}(\bar{B})$ with width at most $\tilde{w}-1$ satisfying

$$q^{\mathrm{res}}\left(\bar{B}\right) = \frac{|\mathrm{Sol}(\bar{B})|}{|\mathrm{Sol}(\mathcal{P}')|} \geq \prod_{j=\tilde{w}-1}^{W}\left(\frac{j}{j+1}\right)^{k_{j+1}}.$$

Define B' as the complement of $\bar{B}$. $\bar{B}$ is a restricted BDD for $\mathcal{P}'$ so that $\mathrm{Sol}(\bar{B}) \subseteq \mathrm{Sol}(\mathcal{P}')$. Together with the facts that $\mathrm{Sol}(B') = \mathbb{B}^n\backslash\mathrm{Sol}(\bar{B})$ (because they are complements) and that $\mathrm{Sol}(\mathcal{P}) = \mathbb{B}^n\backslash\mathrm{Sol}(\mathcal{P}')$, we conclude that B' is a relaxation for $\mathcal{P}$.

Additionally, by Theorem 2, since $w(\bar{B}) \leq \tilde{w}-1$ it follows that $w(B') \leq \tilde{w}$. What remains to be shown is that B' satisfies inequality (3). This follows from

$$\begin{aligned}
q^{\mathrm{rel}}\left(B'\right) &= \frac{|\mathrm{Sol}(\mathcal{P})|}{|\mathrm{Sol}(B')|} && (4)\\
&= \frac{\alpha(\mathcal{P})2^n}{2^n - |\mathrm{Sol}\left(\bar{B}\right)|} && (5)\\
&= \frac{\alpha(\mathcal{P})}{1-\left(\frac{|\mathrm{Sol}(\bar{B})|}{2^n}\right)} && (6)\\
&= \frac{\alpha(\mathcal{P})}{1-\beta(\mathcal{P})\left(\frac{|\mathrm{Sol}(\bar{B})|}{|\mathrm{Sol}(\mathcal{P}')|}\right)} && (7)\\
&= \frac{\alpha(\mathcal{P})}{1-\beta(\mathcal{P})q^{\mathrm{res}}(\bar{B})} && (8)\\
&\geq \frac{\alpha(\mathcal{P})}{1-\beta(\mathcal{P})\left(\prod_{j=\tilde{w}-1}^{W}\left(\frac{j}{j+1}\right)^{k_{j+1}}\right)}, && (9)
\end{aligned}$$

as desired, completing the proof. □

An analogous, general result to Theorem 1 follows. Recall from such theorem that $\theta(n) := 2^{\kappa(n)}$ is an upper bound on the width of any BDD.

Corollary 2 *For any CSP $\mathcal{P}$ there exists a relaxed BDD B' with width $\tilde{w}$ having quality*

$$q^{\text{rel}}(B') \geq \frac{\alpha(\mathcal{P})}{1} - \beta(\mathcal{P}) \left(\prod_{j=\tilde{w}-1}^{\theta(n)} \left(\frac{j}{j+1} \right)^{k_{j+1}} \right).$$

5.2 Quality measure in terms of bounds

The value of the bound provided by approximate BDDs is perhaps most critical. In standard uses of relaxations, such as in optimization, objective function bounds are utilized for pruning search and proving optimality of incumbent solutions. For restricted BDDs, the bound corresponds to the value of a feasible solution found; the better the bound, the better the incumbent solution that is identified.

It is therefore of interest to identify the size of the approximate BDDs needed to obtain objective function bounds of a given value, even though this is a much more challenging task. We provide initial results in Proposition 1 for restricted BDDs and in Theorem 7 for relaxed BDDs.

For restricted BDDs, we have the following simple result.

Proposition 1 *Let z^* be the optimal value for COP $\mathcal{P}$ with objective function $f(x) = \sum_{i=1}^{n} c_i x_i$. There exists a restricted BDD B with $w(B) = 1$ for which the longest path in the BDD, $z(B)$, satisfies that $z(B) = z^*$.*

Proof Let x^* be an optimal solution to $\mathcal{P}$. The BDD B representing this single solution, containing one node u_i per layer L_i together with arcs (u_i, u_{i+1}) having arc-domain x_i^* and arc-value $c_i x_i^*$, is such a desired BDD. □

Unlike in the case of restricted BDDs, proving the existence of a relaxed BDD for which the upper bound provided by the relaxation is at least some specified (valid) bound is non-trivial—a restricted BDD need only represent a single feasible solution while for a relaxed BDD B, a super-set of the feasible solutions needs to be present in $\text{Sol}(B)$. We show a theorem proving the existence of BDDs achieving any given valid objective function bound with a limited width:

Theorem 7 *Let z^{UB} be an upper bound on the optimal value for COP $\mathcal{P} = \langle X, \mathbb{B}^n, \mathcal{C} \rangle$ with objective function $f(x) = \sum_{i=1}^{n} c_i x_i$ to be maximized. There exists a relaxed BDD B with $w(B) \leq z^{\text{UB}} + 1$ for which the longest path in the BDD, $z(B)$, satisfies that $z(B) \leq z^{\text{UB}}$.*

Proof Consider the CSP $\mathcal{P}' = \left\langle X, \mathbb{B}^n, \{\sum_{i=1}^{n} c_i x_i \leq z^{\text{UB}}\} \right\rangle$. Exact BDDs for binary optimization problems with a single linear constraint have been investigated by Behle [3]. In particular, even if $c_i < 0$ for some i, an exact reduced BDD for $\mathcal{P}'$ (with a suitable variable ordering) will have width bounded above by $z^{\text{UB}} + 1$ (see, e.g., page 128 in [3]). Assign arc-lengths $v(a)$ to be $c_{\ell(t(a))}$ (the objective function coefficient in $\mathcal{P}$ of the variable associated with the tail of a). Since for every path p in B the condition $\sum_{i=1}^{n} c_i x(p)_i \leq z^{\text{UB}}$

holds, the longest path in B is bounded by z^{UB}. Furthermore, B is a relaxed BDD because $\text{Sol}(B)$ contains every vector x in $\mathbb{B}^n$ for which $f(x) \leq z^{\text{UB}}$ and z^{UB} is an upper bound on the optimal value of $\mathcal{P}$ so that each solution in $\text{Sol}(\mathcal{P})$ must be in $\text{Sol}(B)$. □

As a corollary to Theorem 7, consider COPs with objective function $\sum_{i=1}^{n} x_i$. Such objective function appear frequently in the literature on combinatorial optimization because they are useful in modeling problems where a maximum (or minimum) number of items need to be selected, such as in the case of set covering, set packing, maximum clique, or the maximum independent set.

Corollary 3 *Let z^{UB} be an upper bound on the optimal value for COP $\mathcal{P} = \langle X, \mathbb{B}^n, \mathcal{C} \rangle$ with objective function $f(x) = \sum_{i=1}^{n} x_i$ to be maximized. There exists a relaxed BDD B with $w(B) \leq n$ for which $z(B) \leq z^{\text{UB}}$.*

Proof An upper bound of n is trivially valid for classes of COPs with this objective function. The result therefore follows from a direct application of Theorem 7. □

6 Bounds tightening techniques for relaxed BDDs

This section provides three techniques for improving upon objective function bounds provided by relaxed BDDs for COPs formulated with linear objective functions. The techniques can be seen as parallels to pure cutting-plane algorithms developed in the context of linear programming (LP) relaxations. Each of the techniques discussed can be used to iteratively improve the objective function bound until either an optimal feasible solution is found, or the problem is proved to be infeasible. In practice, they can be used within branch-and-bound solution methods to improve bounds during search, much like cutting planes are used in branch-and-cut methods for integer programming (IP).

For each technique, it is assumed that a relaxed BDD B has been constructed for COP $\mathcal{P} = \langle X, \mathbb{B}^n, \mathcal{C} \rangle$ (letting $n = |X|$ and $m = |\mathcal{C}|$), and that the objective function f to be maximized is *linear* — $f(x) = \sum_{i=1}^{n} c_i x_i$ (this is easily generalizable to *additively separable objective functions*). It is always safe to assume that a relaxed BDD exists. In particular, one can create the *full* BDD consisting of one node u_i per layer L_i and arc set $A = \cup_{i=1}^{n} \{a_i^0, a_i^1\}$ where for $k = 0, 1$, a_i^k connects nodes u_i and u_{i+1} and has arc-domain $d(a_i^k) = k$ and arc-value $v(a_i^0) = 0$, $v(a_i^1) = c_i$. Constructed in this way, B is a relaxed BDD since $\text{Sol}(B) = \mathbb{B}^n$ and the longest $\mathbf{r} - \mathbf{t}$ path will have length $\sum_{i=1}^{n} \max\{c_i, 0\}$, a trivial upper bound on the optimal solution value. It is therefore assumed that a relaxed BDD B^0 is provided.

6.1 Longest path trimming

A simple, complete algorithm for solving $\mathcal{P}$ is to iteratively *trim* infeasible paths from B^0 creating a sequence of relaxed BDDs $B^0, B^1, B^2, \ldots$ until a feasible (and hence optimal) solution is identified. The procedure *longest path trimming* (LPT) proceeds as follows. First, find a longest path p^0 in B^0. Since B^0 is a relaxed BDD, if $x(p^0)$ is feasible, $x(p^0)$ must also be optimal. If $x(p^0)$ is infeasible, the solution $x(p^0)$ is *trimmed* from B^0 to create another BDD B^1 in such a way that $\text{Sol}(B^1) = \text{Sol}(B) \backslash x(p^0)$. This results in a *tighter* relaxation for $\mathcal{O}$. Continuing in this fashion, finding the longest path p^k in B^k, checking for feasibility of $x(p^k)$, and eliminating it from the solution set of B^k if infeasible to create B^{k+1} will converge to the optimal solution since each BDD created is a relaxed BDD for $\mathcal{P}$. The pseudocode for the algorithm is presented in Algorithm 2.

Algorithm 2 Longest Path Trimming (LPT)

```
1:  procedure LPT($B, \mathcal{P}$)                    ▷ $B$ is a relaxed BDD for $\mathcal{P}$
2:      let $p$ be a longest path in $B$
3:      if $x(p)$ is feasible then
4:          return $x(p)$                              ▷ $x(p)$ is an optimal solution
5:      else
6:          if $|\mathrm{Sol}(B)| = 1$ then
7:              return problem is infeasible           ▷ $x(p)$ is only solution and infeasible
8:          else
9:              TRIM($B, x(p)$)                        ▷ remove $x(p)$ from Sol($B$)
10:             LPT($B, \mathcal{P}$)                  ▷ re-curse
```

Line 9 in Algorithm 2 calls for the trimming operation, which will keep all solutions in B intact but $x(p)$. The pseudocode is provided in Algorithm 3.

Algorithm 3 Trim solution $x(p)$ from Sol(B)

```
1:  procedure TRIM($B, x(p)$)
2:      $u' \leftarrow \mathbf{r}$
3:      for $i = 1, \ldots, n-1$ do
4:          let $a^* = (u', u'')$ be the $(x(p)_i)$-arc directed out of $u'$
5:          if $\exists\, a \in A\backslash\{a^*\}$ with $h(a) = u''$ then     ▷ enter loop only if there are
6:                                                                    other arcs directed at $u''$
7:              create node $v, \ell(v) = i+1$          ▷ all arcs, besides $a^*$, direct into $u''$
8:                                                    will be re-directed to $v$, and all arcs
9:                                                    all arcs out of $u''$ will be duplicated
10:             $U \leftarrow U \cup \{v\}$
11:             for all $a^{\mathrm{out}}$ with $t\left(a^{\mathrm{out}}\right) = u''$ do
12:                 create arc $\tilde{a} = (v, h\left(a^{\mathrm{out}}\right)), d(\tilde{a}) = d(a^{\mathrm{out}})$
13:                 $A \leftarrow A \cup \{\tilde{a}\}$
14:             for all $a^{\mathrm{in}} \in A\backslash\{a^*\}$ with $h(a^{\mathrm{in}}) = u''$ do
15:                 $h(a^{\mathrm{in}}) \leftarrow v$
16:         $u' \leftarrow u''$
17:     let $a^* = (u', \mathbf{t})$ be the arc directed out of $u'$ with arc domain $x(p)_n$
18:     $A \leftarrow A\backslash\{a^*\}$
```

The algorithm starts from the root and iteratively traverses the BDD path p by following the arc domains in $x(p)$. Whenever a node u'' is identified on this path that has arcs directed into u'' other than the arc a^* in p on the corresponding layer, a new node v on is created. All arcs directed out of u'' are duplicated, except with tail v. Additionally, every arc with tail u'', besides a^*, is redirected to have tail v.

Theorem 8 *Algorithm 3 is correct (i.e.,* Sol(B) *contains all solutions but $x(p)$ at the end of the algorithm), runs in time $O(|B|)$, and increases the width of the BDD by at most one.*

Proof Whenever a node is split, upon creation of a new node v, no solutions are added or deleted from Sol(B). This is because the solution $x(p)$ remains intact, while all other arcs directed to u'' are redirect through v, with the exact same completions; i.e., v is created so that $\mathrm{Sol}(B[v, \mathbf{t}])$ is equivalent to $\mathrm{Sol}(B[u'', \mathbf{t}])$.

This implies that until the last line of the algorithm, the solution set remains unchanged. At the culmination, the tail of a^*, node u', has only one arc directed out of it. This is also true for all nodes in path p—by construction there is only *one* arc-directed path going through the nodes along p. This, therefore, is the only solution that is eliminated upon the removal of a^*.

For the running time, note that every node and arc are accessed at most twice, and since $|A| \leq 2|U|$ the overall run time is $O(|B|)$.

Finally, since the algorithm can only increase the node count by at most one node per layer (when node v is created, if necessary), the width can grow by at most 1. □

One thing to note is that after the execution of the algorithm, there may be *dangling* nodes—nodes containing no out-going arcs. There can be deleted with single bottom-up pass of the BDD, but it is not necessary for the proof of correctness.

It follows that LPT is a complete algorithm for any COP, much like the parallel of pure-cutting plane methods in IP.

Theorem 9 *The longest path trimming algorithm, presented in Algorithm 2, is a complete algorithm for COPs.*

Note that the algorithm can be enhanced by, instead of finding the single longest path in the relaxed BDD, finding the k-longest paths, for some value k. This can be done in time proportional to $O(k \cdot n \cdot (m + n\log(n)))$ [16] and for each of the paths checking feasibility. If no feasible solutions are found, eliminating them from the BDD will increase the width by at most k, and then again find the k longest paths. If a feasible solution exists among the solutions corresponding to the k longest paths, the minimum among the set will correspond to the optimal solution.

6.2 Separating inequalities

IP solution methodology relies heavily on the identification of *valid cuts*, that is, inequalities of the form $ax \leq b$ which are valid for the convex hull of integer feasible points of the problem. In this section we describe a similar cutting plane technique to BDDs: For any relaxed BDD B, this technique separates all solutions which violate a (general) constraint C from B.

Given a COP $\mathcal{P} = \langle X, \mathbb{B}^n, \mathcal{C}\rangle$, a BDD B is *exactly consistent* with respect to a constraint $C \in \mathcal{C}$ if every solution $x \in \mathrm{Sol}(B)$ satisfies C. Thus, the question we address is how to modify B to make it exactly consistent for C. To this end, recall that the *scope* of C, henceforth denoted as $s(C)$, is the set of variables in X that are involved in the constraint. Additionally, define $s^{\mathrm{f}}(C)$ and $s^{\ell}(C)$ to be indices of the first and last variables in X that participate in C, respectively.

Our main result is presented in Theorem 10, which requires a well-known result by Bryant [12] stated in Lemma 3. It concern *intersection BDDs*—a BDD whose solution set coincides with the intersection of the solution sets of two input BDDs.

Lemma 3 *Let* B^1, B^2 *be BDDs, each with* $n+1$ *layers. There exists a BDD* B^{int} *with* $\mathrm{Sol}(B^{\mathrm{int}}) = \mathrm{Sol}(B^1) \cap \mathrm{Sol}(B^2)$ *and for which* $w(L_i^{\mathrm{int}}) \leq w(L_i^1) \cdot w(L_i^2)$, *where* $\{L_i^{\mathrm{int}}\}_{i=1}^{n+1}$

are the layers of B^{int} and $\{L_i^k\}_{i=1}^{n+1}$, $k = 1, 2$ are the layers of B^k. Additionally, B^{int} can be constructed in time $O(nw(B^1)w(B^2))$.

Theorem 10 *Let $\mathcal{P} = \langle X, \mathbb{B}^n, \mathcal{C}\rangle$ be a COP and C any constraint in $\mathcal{C}$. Suppose B is a relaxed BDD for $\mathcal{P}$. There exists a relaxed BDD B' for $\mathcal{P}$ which is exactly consistent with respect to C satisfying*

$$w(L'_i)\begin{cases} = w(L_i), & i \le s^{\mathrm{f}}(C), i > s^{\ell}(C) \\ \le w(L_i)\cdot 2^{i-s^{\mathrm{f}}(C)}, & s^{\mathrm{f}}(C) < i \le s^{\mathrm{f}}(C) + (s^{\ell}(C) - s^{\mathrm{f}}(C)+1)\cdot \\ & \qquad \big(\ln(3)\big/\ln(2)+\ln(3)\big) \\ \le w(L_i)\cdot 3^{s^{\ell}(C)-i}, & s^{\mathrm{f}}(C) + (s^{\ell}(C) - s^{\mathrm{f}}(C)+1)\cdot \\ & \qquad \big(\ln(3)\big/\ln(2)+\ln(3)\big) \le i \le s^{\ell}(C), \end{cases}$$

where $\{L_i\}_{i=1}^{n+1}$ are the layers of B and $\{L'_i\}_{i=1}^{n+1}$ are the layers of B'.

Proof Consider the COP $\mathcal{P}' = \langle X, \mathbb{B}^n, \{C\}\rangle$. Inequality (1) in Section 3.2 implies that in the exact reduced BDD $\tilde{B}$ for $\mathcal{P}'$, the width of the layers, denoted by $\tilde{L_i}$, satisfy

$$w(\tilde{L}'_i)\begin{cases} = 1, & i \le s^{\mathrm{f}}(C), i > s^{\ell}(C) \\ \le 2^{i-s^{\mathrm{f}}(C)}, & s^{\mathrm{f}}(C) < i \le s^{\mathrm{f}}(C) + (s^{\ell}(C) - s^{\mathrm{f}}(C)+1)\cdot \\ & \qquad \big(\ln(3)\big/\ln(2)+\ln(3)\big) \\ \le 3^{s^{\ell}(C)-i}, & s^{\mathrm{f}}(C) + (s^{\ell}(C) - s^{\mathrm{f}}(C)+1)\cdot \\ & \qquad \big(\ln(3)\big/\ln(2)+\ln(3)\big) \le i \le s^{\ell}(C). \end{cases}$$

This follows because outside of the range of indices in the scope of the constraint, C does not constrain the variables. Also, the intersection of B and $\tilde{B}$ results in a BDD B' with a width bounded by the product of the widths of the input BDDs, according to Lemma 3. This BDD must have its solution set contained in the solution set of $\tilde{B}$ (the exact BDD for $\mathcal{P}'$) and does not eliminate any other solution in B. Hence, it must be a relaxed BDD for $\mathcal{P}$, which is exactly consistent to C. □

Thus, a constraint can be separated from a BDD B by simply constructing a BDD B' representing the set of solutions satisfied by C and intersecting it with B. Moreover, the theorem can be simplified to the following lemma, which restates the theorem only based on the maximum width.

Corollary 4 *Let $\mathcal{P} = \langle X, \mathbb{B}^n, \mathcal{C}\rangle$ be a COP and C any constraint in $\mathcal{C}$. Suppose B is a relaxed BDD for $\mathcal{P}$. There exists a relaxed BDD B' for $\mathcal{P}$ satisfying*

$$w(B') \le w(B)\cdot\theta\left(s^{\ell}(C) - s^{\mathrm{f}}(C) + 1\right)$$

For a linear inequality $ax \le b$, a stronger results follows because the exact BDD for such constraint has width at most $b+1$.

Corollary 5 *Let $\mathcal{P} = \langle X, \mathbb{B}^n, \mathcal{C}\rangle$ be a COP and $ax \le b$ a valid inequality for $\mathcal{P}$. Suppose B is a relaxed BDD for $\mathcal{P}$. There exists a relaxed BDD B' for $\mathcal{P}$ satisfying that*

$$w(B') \le w(B)\cdot(b+1)$$

This leads to a novel exact algorithm for binary IPs. A *binary* IP is a COP with a linear objective function for which $D = \mathbb{B}^n$ and $\mathcal{C}$ consists of a set of linear inequalities $Ax \leq b$ with A an $m \times n$ matrix and b an m-vector. It is assumed for this paper that all inputs are integral. The theorem proves the existence of an algorithm for solving binary IPs that is only linearly dependent on n.

Theorem 11 *An IP problem can be solved in pseudo-polynomial time $O(nC(mb))$, where C is a function that is independent of n.*

Proof For each linear inequality $a^i x \leq b_i$ in $Ax \leq b$, let $B^i, i = 1, \ldots, m$, be an exact BDD for the binary IP consisting of the single constraint $a^i x \leq b_i$. Notice that the width of B^i is bounded by $b_i + 1$, and therefore each BDD can be constructed in time $O(nb\log(b))$. Thus, the entire construction takes $O(nmb\log(b))$. Intersecting the BDDs one-by-one will result in an exact BDD for the original problem. This BDD will be bounded by $\prod_{i=1}^{m}(b_i+1)$. By Lemma 3, each intersection is linear in n, and since there are $m - 1$ intersections, the running time follows. □

We note that there is a stream of research on parametrized complexity for 0/1 integer linear programming (see for example [24]). The results typically fix n and seek worst case bounds on running time. The result proven here take the alternative viewpoint—for fixed m and b, the class of 0/1 integer linear programming can be solved in time linear in n.

6.3 Value Extraction

Value extracting is a technique introduced by Bergman et al. [10], that can be used to iteratively decrease a relaxation bound provided by a relaxed BDD, until a feasible solution is found. It can be perceived as a generalization of LPT in that solutions are taken from the relaxed BDD until a feasible solution is identified, and was introduced for the *set covering* problem where it was shown to outperform state-of-the-art commercial IP solver on randomly generating instances with limited bandwidth.

The algorithm starts with a relaxed BDD B for COP $\mathcal{P}$ with objective function f. Let $B^{ax=b}$ be the exact BDD for the COP $\mathcal{P}[ax = b] := \langle X, \mathbb{B}^n, ax = b\rangle$. Let the upper bound provided by the relaxed BDD be z^{UB}.

Consider the BDD B^{int}, the intersection between B and $B^{f(x)=z^{\text{UB}}}$. First, $w\left(B^{f(x)=z^{\text{UB}}}\right) \leq z^{\text{UB}} + 1$ and therefore $w(B^{\text{int}}) \leq w(B) \cdot \left(z^{\text{UB}} + 1\right)$ and so for any upper bound the width of this BDD will be bounded by the upper bound times the width of the relaxation BDD.

Second, if a feasible solution exists in B^{int}, it must be optimal. Otherwise, the upper bound can be reduced to $z^{\text{UB}} - 1$. Deciding whether or not a feasible solution exists in B^{int} is precisely the task that a number of previous techniques were designed to solve [4, 14, 15, 20, 21]. In the case a set covering, Bergman et al. [10] proposed a specialized filtering algorithm that, when used in the value extraction framework, was able to provide tighter objective function bounds that LP root-node relaxation values augmented with cutting planes.

One observation is that, unlike LPT, value extraction can be instantiated with *any* bound. Suppose, for example, that for a given binary IP, the root node LP relaxation bound is much stronger than the relaxation bound obtained from a relaxed BDD. In this case, the value extraction procedure can be initialized with the tighter LP bound.

7 Conclusions

This paper presents theoretical insights into the use of BDDs for constraint satisfaction/optimization problems. A worst-case bound on the size of exact BDDs is proven together with several results regarding the quality, both in terms of objective function bounds and degree of infeasibility, of limited-width relaxed and restricted decision diagrams. Finally, several bound tightening procedures for relaxed decision diagrams are described and their various merits discussed.

Acknowledgments We greatly appreciated the input of all the anonymous reviewers. This research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), in the form of a Discovery Grant.

References

1. Andersen, H.R., Hadzic, T., Hooker, J.N., & Tiedemann, P. (2007). A constraint store based on multivalued decision diagrams. In Bessière, C. (Ed.), *Principles and practice of constraint programming (CP 2007). Lecture notes in computer science*, (Vol. 4741, pp. 118–132): Springer.
2. Becker, B., Behle, M., Eisenbrand, F., & Wimmer, R. (2005). BDDS in a branch and cut framework. In Nikoletseas, S. (Ed.), *Experimental and efficient algorithms, proceedings of the 4th international workshop on efficient and experimental algorithms (WEA 05). Lecture notes in computer science*, (Vol. 3503, pp. 452–463): Springer.
3. Behle, M. (2007). On threshold BDDs and the optimal variable ordering problem, In *COCOA'07: Proceedings Of the 1st international conference on combinatorial optimization and applications*, (pp. 124-135). Berlin, Heidelberg: Springer.
4. Bergman, D., Cire, A.A., & van Hoeve, W.J. (2014). MDD Propagation for sequence constraints. *Journal of Artificial Intelligence Research*, *50*, 697–722.
5. Bergman, D., Cire, A.A., van Hoeve, W.J., & Hooker, J.N. (2014). Optimization bounds from binary decision diagrams. *INFORMS Journal on Computing*, *26*(2), 253–268.
6. Bergman, D., Cire, A.A., van Hoeve, W.J., & Hooker, J.N. (2015). Discrete optimization with decision diagrams. *INFORMS Journal on Computing*. to appear.
7. Bergman, D., Cire, A.A., van Hoeve, W.J.J., & Yunes, T.H. (2014). BDD-Based heuristics for binary optimization. *Journal of Heuristics*, *20*(2), 211–234.
8. Bergman, D., Ciré, A.A., Sabharwal, A., Samulowitz, H., Saraswat, V.A., & van Hoeve, W.J. (2014). Parallel combinatorial optimization with decision diagrams. In Simonis, H. (Ed.), *Integration of AI and OR Techniques in Constraint Programming - 11th International Conference, CPAIOR 2014, Cork, Ireland, May 19-23, 2014. Proceedings. Lecture Notes in Computer Science*, (Vol. 8451, pp. 351–367) Springer. doi:10.1007/978-3-319-07046-9_25.
9. Bergman, D., Cire, A.A., & van Hoeve, W.J. (2015). Lagrangian bounds from decision diagrams. *Constraints*, *20*(3), 346–361.
10. Bergman, D., van Hoeve, W.J., & Hooker, J.N. (2011). Manipulating MDD relaxations for combinatorial optimization. In Achterberg, T., & Beck, J.C. (Eds.), *CPAIOR. Lecture notes in computer science*, (Vol. 6697, pp. 20–35): Springer.
11. Bergman, D. (2013). New techniques for discrete optimization. Ph.D. thesis, Carnegie Mellon University.
12. Bryant, R.E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, *C-35*, 677–691.
13. Cheng, K.C., & Yap, R.H. (2010). An mdd-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. *Constraints*, *15*(2), 265–304.
14. Cire, A.A., & van Hoeve, W.J. (2012). MDD Propagation for disjunctive scheduling, In *Proceedings of the twenty-second international conference on automated planing and scheduling (ICAPS)*, (pp. 1–1): AAAI Press.
15. Cire, A.A., & van Hoeve, W.J. (2013). Multivalued decision diagrams for sequencing problems. *Operations Research*, *61*(6), 1411–1428.
16. Eppstein, D. (1998). Finding the k shortest paths. *SIAM J. Comput.*, *28*(2), 652–673.

17. Gopalan, P., Klivans, A., Meka, R., Stefankovic, D., Vempala, S., & Vigoda, E. (2011). An fptas for #knapsack and related counting problems, In *IEEE 52nd annual symposium on Foundations of computer science (FOCS), 2011*, (pp. 817–826).
18. Hadzic, T., & Hooker, J.N. (2007). Cost-bounded binary decision diagrams for 0-1 programming. In Loute, E., & Wolsey, L. (Eds.), *Proceedings of the international workshop on integration of artificial intelligence and operations research techniques in constraint programming for combinatorial optimization problems (CPAIOR 2007). Lecture notes in computer science*, (Vol. 4510, pp. 84–98): Springer.
19. Hadzic, T., Hooker, J.N., O'Sullivan, B., & Tiedemann, P. (2008). Approximate compilation of constraints into multivalued decision diagrams. In Stuckey, P.J. (Ed.), *Principles and practice of constraint programming (CP 2008). Lecture notes in computer science*, (Vol. 5202, pp. 448–462): Springer.
20. Hadzic, T., Hooker, J.N., & Tiedemann, P. (2008). Propagating separable equalities in an MDD store. In Perron, L., & Trick, M.A. (Eds.), *Proceedings of the international workshop on integration of artificial intelligence and operations research techniques in constraint programming for combintaorial optimization problems (CPAIOR 2008). Lecture notes in computer science*, (Vol. 5015, pp. 318–322): Springer.
21. Hoda, S., Hoeve, W.J.V., & Hooker, J.N. (2010). A systematic approach to MDD-based constraint programming, In *Proceedings of the 16th international conference on principles and practices of constraint programming. Lecture notes in computer science*, (Vol. 6308, pp. 266–280): Springer.
22. Hooker, J.N. (2013). Decision diagrams and dynamic programming. In Gomes, C.P., & Sellmann, M. (Eds.), *CPAIOR. Lecture notes in computer science*, (Vol. 7874, pp. 94–110): Springer.
23. Hosaka, K., Takenaga, Y., Kaneda, T., & Yajima, S. (1997). Size of ordered binary decision diagrams representing threshold functions. *Theoretical Computer Science*, *180*(1-2), 47–60.
24. Lokshtanov, D. (2009). New methods in parameterized algorithms and complexity. Ph.D. thesis, University of Bergen.
25. Rothvoß, T. (2011). Some 0/1 polytopes need exponential size extended formulations. arXiv:abs/1105.0036.
26. Rothvoß, T. (2014). The matching polytope has exponential extension complexity, In *Proceedings of the 46th annual ACM symposium on theory of computing*, (pp. 263–272). New York, NY, USA: STOC '14, ACM.
27. Wegener, I. (2000). Branching programs and binary decision diagrams: theory and applications. SIAM monographs on discrete mathematics and applications. Society for Industrial and Applied Mathematics.