On the Modelling of Constraints with Tractable Logical Operators

Ruiwei Wang and Roland H.C. Yap

School of Computing, National University of Singapore
wangruiwei@u.nus.edu, ryap@comp.nus.edu.sg

Abstract

Solving a Constraint Satisfaction Problem (CSP) usually requires a model typically using existing basic constraints. The most flexible form of constraint, ad-hoc (generic) constraints defined with certain constraint representations, such as binary constraint tree (BCT) and decision diagrams, have been proposed where basic constraints in intensional form are insufficient. A modeller may wish to combine basic constraints using logic operators ($\wedge, \vee, \neg$). However, negation, a key logical operator for expressivity is not tractable in many existing constraint representations. This creates a dilemma, for modelling, we would desire more flexibility, but a model whose operations are intractable may in turn be impractical.

In this paper, we give a framework which allows for a tractable negation operator on constraint representations. We apply the framework on the BCT and ordered decision diagram constraints, giving new subforms. These subforms can be strictly more succinct than ordered multi-valued decision diagrams (OMDD), while being as tractable as OMDD for logical combinations. We give applications to show effective propagators from logical combinations and in building large constraint models for configuration problems.

1 Introduction

Constraint Satisfaction Problem (CSP) is a general way of modelling and solving a wide range of combinatorial problems. Like writing programs, to solve a CSP, one has to first model the CSP and it is natural to construct models using logical operators between constraints. In this paper, we focus on ad-hoc constraints (also called generic constraints) as the building blocks of models, because they can easily express any discrete constraint.

Many constraint representations and propagators have been proposed, such as table-like representations (Yap, Xia, and Wang 2020), decision diagrams (Cheng and Yap 2010; Gange, Stuckey, and Szymanek 2011; Perez and Régin 2014) and binary constraint trees (Wang and Yap 2022b). A reason for various representations of ad-hoc constraints is that different representations are natural/efficient for modelling different problems. Indeed when it comes to tractability and representation size (succinctness), the representations can vary considerably (Wang and Yap 2023) and can have practical impact on effective solving and model size.

It is natural to combine constraints into more complex ones by using logical operators (Bacchus and Walsh 2005; Jefferson et al. 2010). Modelling languages such as Minizinc (Nethercote et al. 2007), Essence (Frisch et al. 2008) and CPMpy (Guns 2019) also provide logical operators for modelling constraints. To express all logical operators, a minimal set of logical connectives can be $\{\vee, \neg\}$ or $\{\wedge, \neg\}$ where negation is a key operator for expressivity. However, we can see that tractable negation operator is generally not supported by many useful ad-hoc constraint representations (Wang and Yap 2023). This creates a dilemma, on the one hand, we want more expressiveness in modelling through logical combinations of constraint representations, but on the other hand, how to do so while logical operations are still tractable. Given that there are many constraint representations on which certain logical operators are intractable, care is needed to find the cases where we can have both expressiveness and tractability.

In this paper, we propose a framework, called constraint representation pair, to investigate tractable negation on constraint representations, taking into account the importance of representation size. We study representation size through the notion of succinctness (Darwiche and Marquis 2002; Wang and Yap 2023) which measures the worst-case size. Constraint representation pair gives a way to identify the most succinct representations (i.e. subforms) supporting tractable negation. The ordered multi-valued decision diagram (OMDD) is one of the few existing constraint representations with tractable negation. We identify two subforms of BCT (binary constraint tree) and OMVD (ordered multi-valued variable diagram, a non-deterministic OMDD), respectively called op-BCT and op-OMVD, which can be strictly more succinct than OMDD, while having the same power as OMDD for tractable logical operations. We also close some open questions discussed in (Amilhastre et al. 2014; Wang and Yap 2023). We then show the results can be applied in two problems. Firstly, by using tractable logical operators, we can use op-BCT to model reified constraints. We show that the resulting model is more efficient than one using OMDD constraints. Configuration problems create large models and logical operations occur naturally, typically with OMDD representations. We show that a product configuration benchmark can be modelled much more compactly with op-BCT.

2 Preliminaries

A *Constraint Network (CN)* is a triple (X, D, C) where X is a set of variables, $D(x)$ denotes the domain of a variable $x \in X$ and C is a set of constraints. A *literal* of a variable x is a pair (x, a). A *tuple* over a set of variables S is a set of literals such that every variable in S has exactly one literal in the tuple. Each constraint c is defined with a *scope* $scp(c) \subseteq X$ and a *relation* $rel(c)$ consisting of tuples over $scp(c)$, where $|scp(c)|$ is the *arity* of c, and c is a *binary constraint* if $|scp(c)| = 2$. A *Binary Constraint Network (BCN)* is a CN of which all constraints are binary constraints.

Given variables V and tuple τ, $\tau[V]$ denotes the subset $\{(x, a) \in \tau | x \in V\}$, and $T[V] = \{\tau[V] | \tau \in T\}$ is the *projection* of the tuples T on V. A literal (x, a) is *valid* if $a \in D(x)$. A tuple τ is *valid* if all literals in τ are valid. A tuple τ *satisfies* a constraint c if $\tau[scp(c)] \in rel(c)$. A CN (X, D, C) is *satisfiable* if there is a valid tuple over X satisfying all constraints in C.

A constraint can be modelled with different representations, such as ordinary table (Wang et al. 2016; Demeulenaere et al. 2016), sliced table (Gharbi et al. 2014), basic smart table (Verhaeghe et al. 2017), short support (Jefferson and Nightingale 2013), compressed table (Katsirelos and Walsh 2007), smart table (Mairy, Deville, and Lecoutre 2015), segmented table (Audemard, Lecoutre, and Maamar 2020), semi-MDD (Verhaeghe, Lecoutre, and Schaus 2018), OMDD (Cheng and Yap 2010), smart MDD (Verhaeghe, Lecoutre, and Schaus 2019), OMVD (Amilhastre et al. 2014), deterministic finite automata (Pesant 2004), nondeterministic finite automata (Quimper and Walsh 2006), BCT (Wang and Yap 2022b), and context-free grammar (Quimper and Walsh 2006).

In this paper, we will focus on OMDD, OMVD and BCT, where OMDD is a special case of OMVD. For example, Figure 1a gives an OMDD (it is also an OMVD) modelling the constraint $(x_1 = a_{23} \wedge x_2 = 1 \wedge x_2 = x_3) \vee (x_1 = 1 \wedge x_1 = x_2 \wedge x_3 = a_{12})$, and Figure 2a is a binary constraint tree modelling the constraint $x_1 = y_{12} \wedge x_2 = y_{21}$.

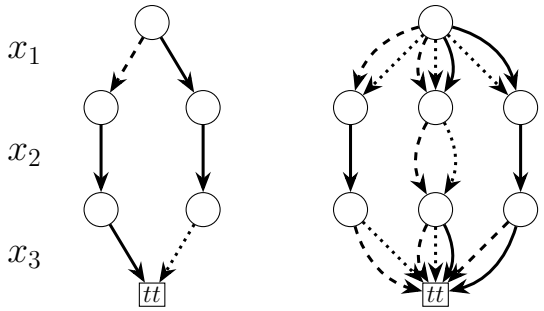
A family of constraints c can be modelled as a representation in polysize if the size of the representation is polynomial in that of the constraints c.

Logical Operators

Usually, logical operators are expressed by conjunction, disjunction and negation operators. The *conjunction* $c_1 \wedge c_2$ between 2 constraints c_1, c_2 is a constraint such that $scp(c_1 \wedge c_2) = scp(c_1) \cup scp(c_2)$ and $rel(c_1 \wedge c_2)$ is the set of valid tuples τ over $scp(c_1 \wedge c_2)$ such that $\tau[scp(c_1)] \in rel(c_1)$ and $\tau[scp(c_2)] \in rel(c_2)$.

The *disjunction* $c_1 \vee c_2$ between 2 constraints c_1, c_2 is a constraint such that $scp(c_1 \vee c_2) = scp(c_1) \cup scp(c_2)$ and $rel(c_1 \vee c_2)$ is the set of valid tuple τ over $scp(c_1 \vee c_2)$ such that $\tau[scp(c_1)] \in rel(c_1)$ or $\tau[scp(c_2)] \in rel(c_2)$.

The *negation* $\neg c$ of a constraint c is a constraint such that $scp(\neg c) = scp(c)$ and the constraint relation $rel(\neg c)$ is the set of valid tuples τ over $scp(\neg c)$ such that $\tau \notin rel(c)$. In this paper, we will refer to constraint c as the primal and correspondingly $\neg c$ as its dual.



(a) Primal (b) Dual

Figure 1: An *op*-OMVD model for constraint family $\mathbf{F}_3$ (Table 1) on variables $\{x_1, x_2, x_3\}$, domain $\{a_{12}, a_{23}, 1\}$, and dotted, dashed and solid lines denote values a_{12}, a_{23}, 1.

A logical operator is *tractable* on a representation if the time complexity of computing the logical operator is polynomial in the size of its operands (representation size).

Succinctness

A constraint representation R_1 is *at least as succinct as* another representation R_2 (i.e. $R_1 \leq R_2$) if for any constraint c modelled as R_2 with size m, c can be modelled as R_1 with size polynomial in m (Darwiche and Marquis 2002; Wang and Yap 2023). A representation R_1 is *strictly more succinct* than R_2 (i.e. $R_1 < R_2$) if $R_1 \leq R_2$ and $R_2 \not\leq R_1$. R_1 is *incomparable* to R_2 (i.e. $R_1 \neq R_2$) if $R_1 \not\leq R_2$ and $R_2 \not\leq R_1$. R_1 is *equal* to R_2 (i.e. $R_1 = R_2$) if $R_1 \leq R_2$ and $R_2 \leq R_1$. We call the representation R_1 a *subform* of R_2 if R_2 is at least as succinct as R_1 (i.e. $R_2 \leq R_1$).

3 Constraint Representation Pairs

We propose the following framework for identifying constraint representations supporting tractable negation. A *constraint representation pair*,[1] p-R, modelling a constraint c is a pair (A, B) such that A and B respectively model c and its negation $(\neg c)$ as a representation R, where A is called a primal constraint and B is a dual constraint. Further, if R is defined with certain structures, such as variable ordering and constraint graph, then an *ordered constraint representation pair*,[2] *op*-R, is a subform of p-R such that the primal and dual constraints both *follow the same structure* (see later). It is straightforward that p-R is a subform of R, and *op*-R is a subform of p-R, which means $R \leq p\text{-}R \leq op\text{-}R$.

Lemma 1. $R \leq p\text{-}R$ and $p\text{-}R \leq op\text{-}R$.

In addition, the negation of a p-R or *op*-R constraint can be computed by exchanging the primal and dual constraints, i.e. $\neg(A, B) = (B, A)$. Hence, the negation operation is tractable on p-R and *op*-R.

Lemma 2. The negation of a p-R (*op*-R) constraint can be computed in polytime.

For any constraint c and any subform of a representation R supporting tractable negation, if c can be modelled as the

[1] p-R refers to R being a pair.

[2] *op*-R refers to a p-R where A and B follow the same structure.

subform with a size n, then there exists A and B respectively modelling c and its negation $\neg c$ as the representation R with a size polynomial in n. The pair (A, B) is a p-R modelling c with a size polynomial in n, thus, p-R is the most succinct subform of R that supports polytime negation.

Lemma 3. p-R is the most succinct subform supporting polytime negation of a constraint representation R.

Moreover, the conjunction and disjunction operations between p-R (or op-R) constraints can be implemented by the operations on R as follows. Let the constraint representation pairs (A_1, B_1) and (A_2, B_2), respectively, model the constraints c_1 and c_2. The conjunction $(A_1, B_1) \wedge (A_2, B_2)$ is equal to $(A_1 \wedge A_2, B_1 \vee B_2)$.

Lemma 4. $(A_1, B_1) \wedge (A_2, B_2) = (A_1 \wedge A_2, B_1 \vee B_2)$

Proof. $A_1 = \neg B_1$ and $A_2 = \neg B_2$, thus, $A_1 \wedge A_2 = \neg(B_1 \vee B_2)$ and $(A_1 \wedge A_2, B_1 \vee B_2)$ models $c_1 \wedge c_2$. □

Then the disjunction operation is encoded with the negation and conjunction operations, i.e. $(A_1, B_1) \vee (A_2, B_2) = \neg(\neg(A_1, B_1) \wedge \neg(A_2, B_2)) = (A_1 \vee A_2, B_1 \wedge B_2)$.

Corollary 1. $(A_1, B_1) \vee (A_2, B_2) = (A_1 \vee A_2, B_1 \wedge B_2)$

Thereby, if the conjunction and disjunction operations are tractable on R, then they are also tractable on p-R.

Lemma 5. The conjunction and disjunction operations on p-R (op-R) can be computed in polytime if they both can be computed in polytime on a constraint representation R.

Based on Lemmas 2 and 5, we can use representation pair to identify the subforms, on which all logic operators are tractable, of the constraint representations supporting tractable conjunction and disjunction, such as decision diagrams and binary constraint trees.

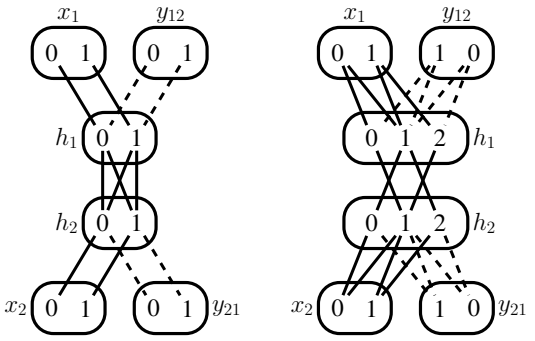
Decision Diagram Pairs

An *ordered multi-valued variable diagram (OMVD)* w.r.t. an ordering O over r variables (Amilhastre et al. 2014) is a rooted DAG which has $r+1$ layers of nodes and a root node in the first layer and a terminal node in the last layer such that each arc pointing from a node in the i^{th} layer of the OMVD is labelled with a value in $D(O_i)$. Each path from the root to the terminal node encodes a tuple in the constraint relation. The *size of an OMVD* is defined as the number of edges and nodes in the OMVD. An *OMDD* (used in MDD constraints) is an OMVD such that the labels of the arcs pointing from the same node are different.

An *OMVD pair* (p-OMVD) modelling a constraint c is a pair (A, B) of OMVDs where A models c and B models $\neg c$. Then (A, B) is an *ordered OMVD pair* (op-OMVD) if A and B follow the same ordering. An OMVD *follows* an ordering O if its variable ordering is a subsequence of O.

We remark that the CD propagator can be directly used to handle p-OMDD and op-OMVD constraints.

Example 1. Figure 1 shows an op-OMVD modelling the constraint $(x_1 = a_{23} \wedge x_2 = 1 \wedge x_2 = x_3) \vee (x_1 = 1 \wedge x_1 = x_2 \wedge x_3 = a_{12})$ The negation of the constraint is $(x_1 = a_{12} \vee x_2 \in \{a_{12}, a_{23}\} \vee x_3 = a_{23}) \vee (x_1 = a_{23} \wedge x_2 \neq x_3) \vee (x_3 = a_{12} \wedge x_1 \neq x_2) \vee (x_1 = x_2 = x_3 = 1)$. Figures 1a and 1b are respectively the primal constraint (an OMDD) and the dual constraint (an OMVD).



(a) Primal (b) Dual

Figure 2: An op-BCT modelling the constraint $x_1 = y_{12} \wedge x_2 = y_{21}$ with variables $\{x_1, x_2, y_{12}, y_{21}\}$ and domain $\{0, 1\}$, where rounded rectangles denote variables and a line between 2 variable values denotes a tuple in the relation of a constraint over the variables.

Binary Constraint Tree Pairs

A *binary constraint tree (BCT)* is a BCN whose constraint graph is a tree (Wang and Yap 2022b). A constraint c can be encoded as a BCT $P = (X, D, C)$ such that $scp(c) \subseteq X$ and $sol(P)[scp[c]] = rel(c)$. We call the variables $X - scp(c)$ as *hidden variables*. The BCT P *follows a constraint graph* (X, E) if $scp(c) \subseteq X$ and the constraint graph of P is constructed by contracting some edges in E (this is similar to edge contraction in graph minors). The binary constraints in a BCT can be regarded as binary matrices and represented with bit sets. The *size of a BCT* is defined as the number of non-zero words used in the bit sets.

A *BCT pair (p-BCT)* modelling a constraint c is a pair of BCTs (A, B) such that A models c and B models $\neg c$. Then (A, B) is an *ordered BCT pair (op-BCT)* if the BCTs A and B follow the same constraint graph.

In particular, the p-BCT and op-BCT constraints can also be handled with the BCT AC propagator introduced in (Wang and Yap 2022b).

Example 2. Figure 2a gives a BCT encoding a constraint $x_1 = y_{12} \wedge x_2 = y_{21}$ with the domain {0,1} from the family $\mathbf{F}_4$ in Table 1, where x_1, x_2, y_{12}, y_{21} are original variables and h_1, h_2 are hidden variables. Then Figure 2b is a BCT encoding the negation of the constraint $x_1 \neq y_{12} \vee x_2 \neq y_{21}$. Figures 2a and 2b follow the same constraint graph, thus, Figure 2 is an op-BCT.

4 Constraint Families (Counterexamples)

We now introduce 6 constraint families specially designed to study constraint representation pairs on BCT and OMDD (and OMVD). BCT and MDD constraints are known to be good representations for constraints and propagators. The families are constraint models constructed using logical operators on binary inequalities and equalities as the base constraints. The constraint models of the families are shown in Table 1, where the size of a constraint from the families is defined as the number of logical operators used in the ex-

$\mathbf{F}_1$	$\vee_{i=1}^{n-1} \vee_{j=i+1}^{n} x_i = x_j$
$\mathbf{F}_2$	$(\wedge_{i=1}^{n-1} \wedge_{j=i+1}^{n} x_i = y_{ij}) \wedge (\wedge_{i=2}^{n} \wedge_{j=1}^{i-1} x_i \neq y_{ji})$
$\mathbf{F}_3$	$\vee_{i=1}^{n} \vee_{j=1}^{n-1} \vee_{k=j+1}^{n} (x_i = a_{jk} \wedge x_j = x_k \wedge$ $\wedge_{l=1}^{i-1} x_l \in [1,n] \wedge \wedge_{l=i+1}^{n} x_l \in [1,n])$
$\mathbf{F}_4$	$(\wedge_{i=1}^{n} \wedge_{j=1}^{n} x_i = y_{ij} \wedge x_i \in [0,1]) \vee$ $(\wedge_{i=1}^{n} \wedge_{j=1}^{n} x_i = y_{ji} \wedge x_i \in [2,3])$
$\mathbf{F}_5$	$(\wedge_{i=1}^{n} \wedge_{j=1}^{n} x_i = y_{ij} \vee x_i \in [2,3]) \wedge$ $(\vee_{i=1}^{n} \vee_{j=1}^{n} x_i \neq y_{ji} \vee x_i \in [0,1])$
$\mathbf{F}_6$	$\wedge_{i=2}^{n} (x_i = x_{\lfloor \frac{i}{2} \rfloor} \vee x_i \notin [f(i)+1, f(i)+n])$ where $f : [1,n] \to \{0, n, 2n\}$ is a function such that $f(i), f(2i)$ and $f(2i+1)$ are AllDifferent

Table 1: Constraint families for studying representations, where x_i and y_j are variables and a_k are values.

pressions. Note that the logical formulas in Table 1 are polysize in the logical operators used.

We use these families as counterexamples to compare the constraint representations discussed in the paper, and study the tractability of logical operators on the representations. For example, family $\mathbf{F}_4$ can be used to prove the intractability of the disjunction operator on various representations (see Section 6), and resolve several open problems in (Amilhastre et al. 2014; Wang and Yap 2023).

Table 2 shows the size of constraint representations on modelling the 6 families.

Theorem 1. The results in Table 2 hold.

In the rest of the section, we prove Theorem 1. We only need to prove the results of blue cells in Table 2. The other cells can be directly verified with the representation succinctness map given later in Figure 3.

The constraint family $\mathbf{F}_1$ is a disjunction of binary equalities, thus, it can be encoded as polysize OMVD (Wang and Yap 2023). In addition, the negation of $\mathbf{F}_1$ models the AllDifferent constraint (Régin 1994) which cannot be encoded as polysize BCT (Wang and Yap 2023; Wang 2024). Therefore, $\mathbf{F}_1$ cannot be modelled as polysize p-BCT.

Lemma 6. $\mathbf{F}_1$ can be modelled as a polysize OMVD but not polysize p-BCT.

The AllDifferent constraint over n variables $\{x_1, \cdots, x_n\}$ can be decomposed into $\frac{n(n-1)}{2}$ binary inequalities. $\mathbf{F}_2$ encodes each binary inequality $x_i \neq x_j$ as $x_i = y_{ij} \wedge x_j \neq y_{ij}$ with a hidden variable y_{ij}. The projection of $\mathbf{F}_2$ on the variables $\{x_1, \cdots, x_n\}$ models the AllDifferent constraints, thus, $\mathbf{F}_2$ cannot be modelled as polysize BCT.

Lemma 7. There is no polysize BCT modelling $\mathbf{F}_2$.

We can get an OMDD modelling $\mathbf{F}_1$ by eliminating and merging some nodes in the OMDD modelling $\mathbf{F}_3$, thus, there is no polysize OMDD encoding $\mathbf{F}_3$.

Lemma 8. $\mathbf{F}_3$ cannot be modelled as a polysize OMDD.

Proof. Let A be an OMDD modelling $\mathbf{F}_3$ w.r.t. any variable ordering O. By eliminating the nodes and edges which cannot reach the terminal node via an edge labelled with a value in $\{a_{jk} | 1 \leq j < k < n\}$ in the n^{th} layer, we can get an OMDD modelling $\vee_{j=1}^{n-2} \vee_{k=j+1}^{n-1} (x_n = a_{jk} \wedge x_j = x_k \wedge (\wedge_{l=1}^{n-1} x_l \in [1,n]))$. Then merging all nodes in the n^{th} layer as a terminal node can get an OMDD modelling $\mathbf{F}_1$. So there is no polysize OMDD modelling $\mathbf{F}_3$. □

	$\mathbf{F}_1$	$\mathbf{F}_2$	$\mathbf{F}_3$	$\mathbf{F}_4$	$\mathbf{F}_5$	$\mathbf{F}_6$
BCT	✓	•	✓	•	✓	✓
p-BCT	•	•	✓	•	✓	✓
op-BCT	•	•	✓	•	•	✓
OMVD	✓	•	✓	•	✓	•
p-OMVD	•	•	✓	•	✓	•
op-OMVD	•	•	✓	•	•	•
OMDD	•	•	•	•	•	•

Table 2: Representation sizes of modelling the constraint families given in Table 1, where • (✓) denotes a family cannot (can) be modelled as the representation in polysize.

Lemma 9. $\mathbf{F}_3$ can be modelled as a polysize op-OMVD.

Proof. The constraint family $\mathbf{F}_3$ is a disjunction of ternary constraints. In addition, the negation of $\mathbf{F}_3$ is equal to $(\vee_{i=1}^{n-1} \vee_{j=i+1}^{n} x_i \notin [1,n] \wedge x_j \notin [1,n]) \vee (\vee_{i=1}^{n} \vee_{j=1}^{n-1} \vee_{k=j+1}^{n} x_i = a_{jk} \wedge x_j \neq x_k) \vee (\wedge_{i=1}^{n} x_i \in [1,n])$, which is a disjunction of binary constraints and $\wedge_{i=1}^{n} x_i \in [1,n]$. Binary/ternary constraints and $\wedge_{i=1}^{n} x_i \in [1,n]$ can be encoded as polysize OMVD w.r.t. any variable ordering. So $\mathbf{F}_3$ can be modelled as polysize op-OMVD □

The constraint families $\mathbf{F}_4, \mathbf{F}_5$ and $\mathbf{F}_6$ are combinations of some constraints which can be obtained by setting the variable domain of the families to certain values. For example, by setting the variable domain of $\mathbf{F}_4$ to the values [0,1] (and [2,3]), we can get the constraint $\wedge_{i=1}^{n} \wedge_{j=1}^{n} x_i = y_{ij}$ (and $\wedge_{i=1}^{n} \wedge_{j=1}^{n} x_i = y_{ji}$).

Lemma 10. $\mathbf{F}_4$ cannot be modelled as a polysize BCT.

Proof. Let $A = (H \cup V \cup X, C)$ be any BCT encoding $\mathbf{F}_4$ where H is the hidden variables, and $X = \{x_1, \cdots, x_n\}$ and $V = \{y_{ij} | 1 \leq i, j \leq n\}$ are the original variables.

From Lemma 1 in (Wang and Yap 2023), there is a variable v such that eliminating v partitions the constraint graph of A into connected components (CCs) where each CC includes at most $\frac{n}{2}$ variables in X. Correspondingly, for each $x_i \in X$, there are at least $\frac{n-2}{2}$ variables $x_j \in X$ such that x_i, x_j are in different CCs, thus, x_i, y_{ij} or x_j, y_{ij} are in different CCs. Hence, there are $\frac{n^2-2n}{2}$ equalities $x_i = y_{ij}$ or $x_i = y_{ji}$ connecting variables in different CCs.

By the pigeonhole principle, there must be $\frac{n-2}{4}$ variables S in X such that one of the following two cases holds: (i) every $x_i \in S$ is equal to a variable y_{ij} in different CC; (ii) each $x_i \in S$ equals to a variable y_{ji} in different CC.

In case (i), for any tuples τ_1, τ_2 in $rel(A)$ such that the variables X are assigned with values in $[0,1]$, if $\tau_1[S] \neq \tau_2[S]$, then the variable v must be assigned with different values in the tuples τ_1, τ_2, otherwise for any CC with variables y_{CC}, $(\tau_1 \setminus \tau_1[y_{CC}]) \cup \tau_2[y_{CC}]$ is a tuple in $rel(A)$ (a contradiction). So v has at least $2^{\frac{n-2}{4}}$ values.

Similarly, in case (ii), v also has at least $2^{\frac{n-2}{4}}$ values. So there is no polysize BCT encoding $\mathbf{F}_4$. □

Lemma 11. $\mathbf{F}_5$ cannot be modelled as a polysize *op*-BCT.

Proof. Let (A, B) be a polysize *op*-BCT modelling $\mathbf{F}_5$ with the variable domain $[0, 3]$. By removing the tuples having values in $[2, 3]$ from A, we get a BCT A_1 encoding $\wedge_{i=1}^{n-1} \wedge_{j=i+1}^{n} x_i = y_{ij} \wedge x_i \in [0, 1]$.

The negation of $\mathbf{F}_5$ is $(\vee_{i=1}^{n-1} \vee_{j=i+1}^{n} x_i \neq y_{ji} \wedge x_i \in [0, 1]) \vee (\wedge_{i=1}^{n} \wedge_{j=1}^{n} x_i = y_{ji} \wedge x_i \in [2, 3])$. Then eliminating the tuples having values in $[0, 1]$ from B, we get a BCT B_1 encoding $\wedge_{i=1}^{n} \wedge_{j=1}^{n} x_i = y_{ji} \wedge x_i \in [2, 3]$.

Disjunction is tractable on the BCTs following the same graph (Wang and Yap 2023), thus, $A_1 \vee B_1$ is a polysize BCT modelling $\mathbf{F}_4$ (a contradiction). So there is no polysize *op*-BCT modelling $\mathbf{F}_5$. □

Lemma 12. There is a polysize p-OMVD encoding $\mathbf{F}_5$.

Proof. The constraint graph of $\wedge_{i=1}^{n} \wedge_{j=1}^{n} x_i = y_{ij} \vee x_i \notin [2, 3]$ is separated stars, so it can be encoded as a polysize OMVD w.r.t. the variable ordering $x_1 < y_{11} < \cdots < y_{1n} < \cdots < x_n < y_{n1} < \cdots < y_{nn}$. Then $\vee_{i=1}^{n} \vee_{j=1}^{n} x_i \neq y_{ji} \vee x_i \in [0, 1]$ is a disjunction of binary constraint which can be encoded as a polysize OMVD w.r.t. any variable ordering. In addition, conjunction is tractable on OMVD with the same variable ordering (Amilhastre et al. 2014), thus, $\mathbf{F}_5$ can be encoded as polysize OMVD.

Similarly, the negation of $\mathbf{F}_5$ can be encoded as a polysize OMVD w.r.t. the variable ordering $x_1 < y_{11} < \cdots < y_{n1} < \cdots < x_n < y_{1n} < \cdots < y_{nn}$.

So $\mathbf{F}_5$ can be encoded as polysize p-OMVD. □

Lemma 13. $\mathbf{F}_6$ cannot be modelled as a polysize OMVD.

Proof. WLOG, we assume $n = 2^{h+1} - 1$. The constraint graph of $\mathbf{F}_6$ is a complete binary tree of height h, which means the pathwidth of $\mathbf{F}_6$ is $\lceil \frac{h}{2} \rceil$, see (Scheffler 1989) and Lemma 2.1 in (Groenland et al. 2023). Therefore, for any OMVD A modelling $\mathbf{F}_6$ w.r.t. any variable ordering O, there must be $2 \leq l \leq n$ such that existing $\frac{h}{2}$ binary equalities crossing the l^{th} layer in the OMVD.

By the pigeonhole principle, there are at least $\frac{h}{6}$ variables S in $\{x_1, \cdots, x_n\}$ such that for all variables $x_i \in S$, $f(i)$ is equal to the same $k \in \{0, n, 2n\}$, and $x_i, x_{\lfloor \frac{i}{2} \rfloor}$ are separated by the l^{th} layer. This means for all x_i, x_j in S, $j \neq \lfloor \frac{i}{2} \rfloor$, in addition, for any tuples $\tau_1, \tau_2 \in rel(A)$ and $x \in \{x_1, \cdots, x_n\}$ such that $x \in [k + 1, k + n]$ and $\tau_1[S] \neq \tau_2[S]$, the paths encoding τ_1, τ_2 must pass different nodes in the l^{th} layer of A. Therefore, the l^{th} layer of A has at least $n^{\frac{h}{6}} = n^{O(log(n))}$ nodes. So there is no polysize OMVD encoding $\mathbf{F}_6$. □

Lemma 14. There is a polysize *op*-BCT encoding $\mathbf{F}_6$.

Proof. The constraint graph of the constraint family $\mathbf{F}_6$ is a tree, so it can be encoded as polysize BCT. In addition, the negation of $\mathbf{F}_6$ is a disjunction of binary constraints which can be encoded as polysize BCT with any constraint graph including hidden variables. Therefore, $\mathbf{F}_6$ can be encoded as polysize *op*-BCT. □

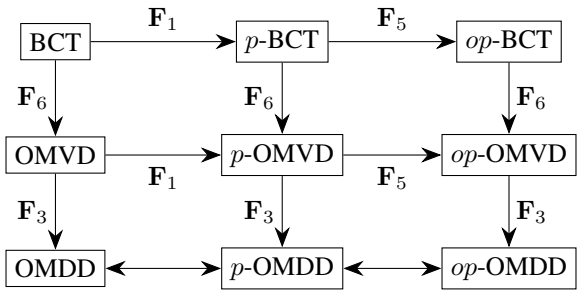


Figure 3: Succinctness of constraint representations. An arc from R_1 to R_2 denotes $R_1 \leq R_2$. An arc labelled with a family F from R_1 to R_2 means F can be encoded as polysize R_1 but not R_2. In addition, if there is a path from R_1 to R_2, then $R_1 \leq R_2$, otherwise $R_1 \not\leq R_2$.

5 Succinctness of Representations

Following (Wang and Yap 2023), we use succinctness to compare the compactness of constraint representations (note that succinctness evaluates constraint representations from a worst-case perspective). Figure 3 shows the succinctness of the constraint representations discussed in the paper. We remark (not shown) that all these representations are more succinct than the table constraint. In the figure, the constraint representation pairs p-BCT, *op*-BCT, p-OMVD and *op*-OMVD can reach OMDD but not vice versa, therefore, they are all strictly more succinct than OMDD. The constraint families on the arcs in Figure 3 are used to separate constraint representations, e.g. the arc from *op*-OMVD to *op*-OMDD is labelled by $\mathbf{F}_3$, which means the family $\mathbf{F}_3$ can be encoded as polysize *op*-OMVD but not polysize OMDD (see Table 2). In this section, we will prove that the results shown in Figure 3 hold.

For any constraint representation R, the representation pair p-R is a subform of R, and *op*-R is a subform of p-R (Lemma 1), therefore, BCT $\leq$ p-BCT $\leq$ *op*-BCT, and OMVD $\leq$ p-OMVD $\leq$ *op*-OMVD. Then constraint representation pair identifies the most succinct subform supporting tractable negation (Lemma 3). Hence, if a representation R_2 is a subform of another representation R_1, then p-R_2 is also a subform of p-R_1.

Lemma 15. If $R_1 \leq R_2$, then p-$R_1 \leq p$-R_2.

Proof. Let (A_2, B_2) be any p-R_2 constraint. $R_1 \leq R_2$, thus, there are A_1, B_1 modelling A_2, B_2 as polysize R_1, i.e. the size of the p-R_1 constraint (A_1, B_1) is polynomial in that of the p-R_2 constraint (A_2, B_2). So p-$R_1 \leq p$-R_2. □

The negation on OMDD can be computed in polytime without changing variable ordering, thus, p-OMDD and *op*-OMDD have the same succinctness as OMDD.

Lemma 16. OMDD = p-OMDD = *op*-OMDD.

Proof. As shown in (Amilhastre et al. 2014), for any OMDD A w.r.t. a variable ordering O, it is tractable to compute an OMDD B encoding $\neg A$ w.r.t. O, where the pair (A, B) is a polysize *op*-OMDD encoding A. So *op*-OMDD $\leq$ OMDD,

	$A \wedge B$	$A \vee B$	$\neg A$	$\bigwedge S$	$\bigvee S$
op-BCT	• $\mathbf{F}_2$	• $\mathbf{F}_4$	✓	• $\mathbf{F}_2$	• $\mathbf{F}_4$
op-OMVD	• $\mathbf{F}_2$	• $\mathbf{F}_4$	✓	• $\mathbf{F}_2$	• $\mathbf{F}_4$
p-BCT	• $\mathbf{F}_2$	• $\mathbf{F}_4$	✓	• $\mathbf{F}_2$	• $\mathbf{F}_4$
p-OMVD	• $\mathbf{F}_2$	• $\mathbf{F}_4$	✓	• $\mathbf{F}_2$	• $\mathbf{F}_4$
OMDD	• $\mathbf{F}_2$	• $\mathbf{F}_4$	✓	• $\mathbf{F}_2$	• $\mathbf{F}_4$
OMVD	• $\mathbf{F}_2$	• $\mathbf{F}_4$	• $\mathbf{F}_1$	• $\mathbf{F}_2$	• $\mathbf{F}_4$
BCT	• $\mathbf{F}_2$	• $\mathbf{F}_4$	• $\mathbf{F}_1$	• $\mathbf{F}_2$	• $\mathbf{F}_4$

Table 3: Tractability of logical operators, where • (✓) denotes a logical operator on a representation is not tractable (is tractable). The results in the blue cells resolve some open problems in (Amilhastre et al. 2014; Wang and Yap 2023).

which means OMDD, *p*-OMDD and *op*-OMDD have the same succinctness. □

Then every OMDD w.r.t. an variable ordering O is also an OMVD w.r.t. O, thus, *op*-OMVD $\leq$ *op*-OMDD $\leq$ OMDD.

Lemma 17. *op*-OMVD $\leq$ OMDD.

In addition, two OMVDs following the same ordering can be encoded as two BCTs following the same constraint graph by the direct binary tree encoding from (Wang and Yap 2022a,b). Hence, we can get that *op*-BCT $\leq$ *op*-OMVD.

Lemma 18. *op*-BCT $\leq$ *op*-OMVD.

With the lemmas, and also the results shown in Table 2, we can prove the results in Figure 3.

Theorem 2. The results of Figure 3 hold.

Proof. The arcs between BCT, OMVD and OMDD have been proved in (Amilhastre et al. 2014; Wang and Yap 2023). The other arcs in Figure 3 can be directly verified with Lemmas 1, 15, 16, 17 and 18. Then the labels on the arcs, i.e. the constraint families $\mathbf{F}_1$, $\mathbf{F}_3$, $\mathbf{F}_5$ and $\mathbf{F}_6$, can be verified with the results shown in Table 2. □

6 Tractability of Logical Operators

We then study the tractability of the conjunction, disjunction and negation operators. The results are shown in Tables 3 and 4, where $A \wedge B$ ($A \vee B$) denotes the conjunction (disjunction) between 2 constraints, $\neg A$ is the negation operator, and $\wedge S$ ($\vee S$) means the conjunction (disjunction) of a set S of constraints. Table 3 gives the results of representations with arbitrary structures, and Table 4 is the results of representations following same structure. We remark that *p*-OMDD and *op*-OMDD have the same succinctness as OMDD, thus, they also support the same tractable logical operators.

Representations With Arbitrary Structures

If NP $\neq$ P, the conjunction is NP-hard and intractable on OMDD with arbitrary variable orderings (Amilhastre et al. 2014; Wang and Yap 2023), thus, it is also NP-hard and intractable on the constraint representation pairs more succinct than OMDD. We show a stronger result using Table 2 that conjunction cannot be computed in polytime on the representations with arbitrary structures.

	$A \wedge B$	$A \vee B$	$\neg A$	$\bigwedge S$	$\bigvee S$
op-BCT	✓	✓	✓	• $\mathbf{F}_2$	• $\mathbf{F}_1$
op-OMVD	✓	✓	✓	• $\mathbf{F}_2$	• $\mathbf{F}_1$
OMDD	✓	✓	✓	• $\mathbf{F}_2$	• $\mathbf{F}_1$
OMVD	✓	✓	• $\mathbf{F}_1$	• $\mathbf{F}_2$	✓
BCT	✓	✓	• $\mathbf{F}_1$	• $\mathbf{F}_2$	✓

Table 4: Tractability of logical operators between representations following the same structure.

Lemma 19. The conjunction between two OMDD (BCT, *p*-BCT, *op*-BCT, OMVD, *p*-OMVD or *op*-OMVD) constraints cannot be computed in polytime.

Proof. Based on Table 2, the family $\mathbf{F}_2$ cannot be encoded as the representations in polysize. $\mathbf{F}_2$ is a conjunction $(\wedge_{i=1}^{n} c_i^1) \wedge (\wedge_{i=2}^{n} c_i^2)$ where c_i^1 equals to $\wedge_{j=i+1}^{n} x_i = y_{ij}$ and c_i^2 is $\wedge_{j=1}^{i-1} x_i \neq y_{ji}$. Then $\wedge_{i=1}^{n} c_i^1$ can be encoded a polysize OMDD A w.r.t. the ordering $x_1 < y_{12} < \cdots < y_{1n} < \cdots < x_{n-1} < y_{(n-1)n}$, as c_i^1 has a star structure. Similarly, $\wedge_{i=2}^{n} c_i^2$ can be encoded as a polysize OMDD B w.r.t. the variable ordering $x_2 < y_{21} < \cdots < x_n < y_{n1} \cdots < y_{n(n-1)}$. There is no polysize BCT encoding $A \wedge B$, so conjunction cannot be computed in polytime on the representations. □

The tractability of disjunction on OMDD, OMVD and BCT constraints are also open problems discussed in (Amilhastre et al. 2014; Wang and Yap 2023). We resolve them with the results given in Table 2.

Lemma 20. The disjunction between two OMDD (BCT, *p*-BCT, *op*-BCT, OMVD, *p*-OMVD or *op*-OMVD) constraints cannot be computed in polytime.

Proof. $\mathbf{F}_4$ cannot be encoded as polysize BCT (see Table 2). Then $\mathbf{F}_4$ is a disjunction $c_1 \vee c_2$ where c_1 is equal to $\wedge_{i=1}^{n} \wedge_{j=1}^{n} x_i = y_{ij}$ and c_2 is $\wedge_{i=1}^{n} \wedge_{j=1}^{n} x_i = y_{ji}$. Similar to the proof of Lemma 19, c_1, c_2 have a star structure and can be encoded as polysize OMDD, thus, the disjunction operator is not tractable on the representations. □

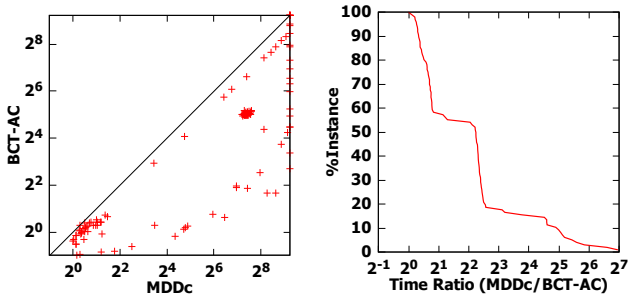
Based on Lemmas 2, 19 and 20, we then show that the results in Table 3 hold.

Theorem 3. The results in Table 3 hold.

Proof. The negation operator is tractable on OMDD but it is not tractable on OMVD and BCT (Wang and Yap 2023). The constraint representation pairs support tractable negation (see Lemma 2). Based on Lemmas 19 and 20, the disjunction and conjunction are intractable on representations. So the results in Table 3 hold. □

Representations Following the Same Structure

The disjunction and conjunction are tractable between two BCT (OMVD or OMDD) constraints following the same structures (Amilhastre et al. 2014; Wang and Yap 2023). Hence, it follows that the disjunction and conjunction operators are tractable between two *op*-BCT (or *op*-OMVD) constraints following the same structures (see Lemma 5).



(a) Time (in seconds) (b) Speedup

Figure 4: Results of constraint reification benchmarks.

Instance	OMDD		*op*-BCT				
			Primal		Dual		Total
	#node	#edge	#hval	#word	#hval	#word	#word
small	2810	3201	178	480	1198	1481	**1961**
medium	9333	10K	331	684	1948	2435	**3119**
big	691K	846K	1870	3811	3280	10K	**14K**
master	633K	824K	4729	6273	1687	2811	**9084**
megane	326K	439K	9161	29K	2997	5820	**34K**

Table 5: Product configuration benchmarks with MinFill variable ordering. #node and #edge denote the number of nodes and edges in OMDD. #hval is the number of hidden variable values in *op*-BCT, and #word captures the size of the BCT representation in non-zero words.

Lemma 21. The conjunction and disjunction between two *op*-BCT (or *op*-OMVD) constraints following the same structure are tractable.

The family $\mathbf{F}_1$ ($\mathbf{F}_2$) cannot be modelled as polysize *op*-BCT and *op*-OMVD but it is a disjunction (conjunction) of binary constraints that can be encoded as polysize *op*-BCT and *op*-OMVD w.r.t. the same structure. So it is intractable to compute the disjunction (conjunction) of a set of *op*-BCT and *op*-OMVD constraints.

Lemma 22. The conjunction and disjunction of a set of *op*-BCT (or *op*-OMVD) constraints following the same structure are intractable.

Then Lemma 2 shows negation is tractable on *op*-BCT (*op*-OMVD). So we can prove the results in Table 4 hold.

Theorem 4. The results in Table 4 hold.

Proof. The results of OMDD, OMVD and BCT are shown in (Amilhastre et al. 2014; Wang and Yap 2023). Based on Lemmas 2, 21 and 22, the results also hold for *op*-BCT and *op*-OMVD. So the results in Table 4 hold. □

7 Practical Applications

We apply the logical operators of *op*-BCT and OMDD to model reified constraints and product configuration problems. We implement the logical operators for *op*-BCT and OMDD, and using them to construct BCT and MDD constraints in the models below. For the implementation of *op*-BCT, we use rules from (Wang and Yap 2022b) to do reduction, and for OMDD, we follow (Perez and Régin 2015). The MDDc GAC propagator (Cheng and Yap 2010) and BCT AC propagator (Wang and Yap 2022b) in the Abscon solver (Merchez, Lecoutre, and Boussemart 2001) are used to handle the MDD and BCT constraints. Experiments were conducted on a 3.20GHz Intel i7-8700 machine with Ubuntu 12.3 and 16GB of RAM. Timeout is 600 seconds and instances are run once.

Constraint Reification Benchmarks

Constraint reification is useful for combining constraints, where a reified constraint $x \Leftrightarrow c$ is equal to $(x \wedge c) \vee (\neg x \wedge \neg c)$. If x and c can be modelled as compact *op*-BCT (or OMDD), then, the logical operators of *op*-BCT (or OMDD) can be used to model the reified constraint.

We use 7 series of instances from CSP competition 2009[3] to evaluate *op*-BCT and OMDD on modelling reified constraints: bddSmall, dag-half, lexVg, ukVg, wordsVg, TSP-20 and TSP-25. The CSP model evaluated is: (i) for each constraint c_i in an instance, a refined constraint $y_i \Leftrightarrow c_i$ is used to combine c_i with a variable y_i; and (ii) a sum constraint $y_1 + \cdots + y_n = k$ is used to model that there are exactly k satisfied constraints. Experiments use $k = n - 1$. In total, there are 122 instances after removing the trivial instances can be solved by both methods in 1 second, where the search heuristic is WDeg (Boussemart et al. 2004).

Figure 4a compares the running time of the MDDc and BCT AC propagators, where each dot in the figure denotes an instance. Most dots are below the diagonal line, showing the BCT AC propagator is faster than MDDc on the most instances. Then Figure 4b evaluates time ratio on the instances solved by both methods within the timeout. Each dot (α, β) in Figure 4b denotes that on β percent instances, the ratio of "the running time of MDDc" to "the running time of BCT AC propagator" is at least α. The BCT AC propagator can be up to 124 times faster than MDDc. In 54% of instances, the BCT AC propagator is at least 4 times faster than MDDc.

This experiment shows that the propagator from constraints constructed with logical expressions over *op*-BCT is effective compared with MDDc propagator. We remark that OMDD is the most well known representation to capture structure in an ad-hoc constraint, and MDDc is also available in various CSP solvers.

Product Configuration Benchmarks

To investigate the total representation size of a model created from logical operators, we evaluate *op*-BCT on models from product configuration instances (Bessiere, Fargier, and Lecoutre 2016). Compared to OMDD, *op*-BCT can have a more flexible structure. We can construct a constraint graph for *op*-BCT based on the tree decomposition generated by eliminating variables in any variable ordering. The MinFill ordering is used in the experiment.

Table 5 shows the sizes of OMDD and *op*-BCT on modelling the product configuration instances. As these are different representations, the numbers cannot be compared directly, however the size of OMDD is mostly captured by

[3] www.cril.univ-artois.fr/CSC09

(#node,#edge) and *op*-BCT by (#hval,#word) of the primal and dual constraints. We can see that *op*-BCT can be much more compact than the OMDD with the same variable ordering (i.e. the MinFill ordering).

8 Conclusion

This paper presents a framework using (ordered) constraint pair which can be used to study when tractable negation becomes possible. Expressing constraints with logical operators, which implicitly includes a particular representation, is a common way of modelling constraint problems. However, negation is known to pose problems as it is not tractable for certain representations. This paper shows how we can gain improved expressivity with representations that can support $(\wedge, \vee, \neg)$ tractably. Naturally, this is not always possible. We investigate the BCT and OMVD representations on our framework, as they can support $(\wedge, \vee)$ tractablely (with the same structure) and already have good succinctness compared to other representations of ad-hoc constraints. We show that *op*-BCT and *op*-OMVD is tractable under logical operators following the same structure while being more succinct than OMDD. While the main contribution of this paper are the theoretical results on tractability and succinctness, we also present preliminary experimental results to show the potential of the tractable representations with efficient and compact models.

This paper proposes a new direction to have both modelling expressiveness as well as tractability. It opens a study of existing and new representations where perhaps we can have more expressiveness and tractability.

Acknowledgements

This work has been supported by grants A-8000896-00-00 and A-8002255-00-00.

References

Amilhastre, J.; Fargier, H.; Niveau, A.; and Pralet, C. 2014. Compiling CSPs: A complexity map of (non-deterministic) multivalued decision diagrams. *International Journal on Artificial Intelligence Tools*, 23(04): 1460015.

Audemard, G.; Lecoutre, C.; and Maamar, M. 2020. Segmented Tables: An Efficient Modeling Tool for Constraint Reasoning. In *ECAI 2020*, 315–322. IOS Press.

Bacchus, F.; and Walsh, T. 2005. Propagating logical combinations of constraints. In *International Joint Conference on Artificial Intelligence*, 35–40.

Bessiere, C.; Fargier, H.; and Lecoutre, C. 2016. Computing and restoring global inverse consistency in interactive constraint satisfaction. *Artificial Intelligence*, 241: 153–169.

Boussemart, F.; Hemery, F.; Lecoutre, C.; and Sais, L. 2004. Boosting systematic search by weighting constraints. In *European Conference on Artificial Intelligence*.

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

Darwiche, A.; and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17: 229–264.

Demeulenaere, J.; Hartert, R.; Lecoutre, C.; Perez, G.; Perron, L.; Régin, J.-C.; and Schaus, P. 2016. Compact-Table: efficiently filtering table constraints with reversible sparse bit-sets. In *International Conference on Principles and Practice of Constraint Programming*, 207–223.

Frisch, A. M.; Harvey, W.; Jefferson, C.; Martinez-Hernandez, B.; and Miguel, I. 2008. Essence: A constraint language for specifying combinatorial problems. *Constraints*, 13(3): 268–306.

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

Gharbi, N.; Hemery, F.; Lecoutre, C.; and Roussel, O. 2014. Sliced Table Constraints: Combining Compression and Tabular Reduction. In *Proceedings of the 11th International Conference on Integration of Artificial Intelligence and Operations Research techniques in Constraint Programming*, 120–135.

Groenland, C.; Joret, G.; Nadara, W.; and Walczak, B. 2023. Approximating pathwidth for graphs of small treewidth. *ACM transactions on algorithms*, 19(2): 1–19.

Guns, T. 2019. Increasing modeling language convenience with a universal n-dimensional array, CPpy as python-embedded example. In *Proceedings of the 18th workshop on Constraint Modelling and Reformulation at CP (Modref 2019)*.

Jefferson, C.; Moore, N. C.; Nightingale, P.; and Petrie, K. E. 2010. Implementing logical connectives in constraint programming. *Artificial Intelligence*, 174(16-17): 1407–1429.

Jefferson, C.; and Nightingale, P. 2013. Extending simple tabular reduction with short supports. In *International Joint Conferences on Artificial Intelligence*, 573–579.

Katsirelos, G.; and Walsh, T. 2007. A compression algorithm for large arity extensional constraints. In *International Conference on Principles and Practice of Constraint Programming*, 379–393.

Mairy, J.-B.; Deville, Y.; and Lecoutre, C. 2015. The Smart Table Constraint. In *International Conference on the Integration of AI and OR Techniques in Constraint Programming*, 271–287.

Merchez, S.; Lecoutre, C.; and Boussemart, F. 2001. Abscon: A prototype to solve CSPs with abstraction. In *International Conference on Principles and Practice of Constraint Programming*, 730–744. Springer.

Nethercote, N.; Stuckey, P. J.; Becket, R.; Brand, S.; Duck, G. J.; and Tack, G. 2007. MiniZinc: Towards a standard CP modelling language. In *International Conference on Principles and Practice of Constraint Programming*, 529–543.

Perez, G.; and Régin, J.-C. 2014. Improving GAC-4 for table and MDD constraints. In *International Conference on Principles and Practice of Constraint Programming*, 606–621.

Perez, G.; and Régin, J.-C. 2015. Efficient operations on MDDs for building constraint programming models. In

Twenty-Fourth International Joint Conference on Artificial Intelligence, 374–380.

Pesant, G. 2004. A regular language membership constraint for finite sequences of variables. In *International Conference on Principles and Practice of Constraint Programming*, 482–495.

Quimper, C.-G.; and Walsh, T. 2006. Global grammar constraints. In *International Conference on Principles and Practice of Constraint Programming*, 751–755.

Régin, J.-C. 1994. A filtering algorithm for constraints of difference in CSPs. In *AAAI Conference on Artificial Intelligence*, 362–367.

Scheffler, P. 1989. *Die Baumweite von Graphen als ein Maß für die Kompliziertheit algorithmischer Probleme*, volume 4. Akademie der Wissenschaften der DDR, Karl-Weierstrass-Institut für Mathematik.

Verhaeghe, H.; Lecoutre, C.; Deville, Y.; and Schaus, P. 2017. Extending Compact-Table to basic smart tables. In *International Conference on Principles and Practice of Constraint Programming*, 297–307.

Verhaeghe, H.; Lecoutre, C.; and Schaus, P. 2018. Compact-MDD: Efficiently Filtering (s)MDD Constraints with Reversible Sparse Bit-sets. In *International Joint Conference on Artificial Intelligence*, 1383–1389.

Verhaeghe, H.; Lecoutre, C.; and Schaus, P. 2019. Extending Compact-Diagram to Basic Smart Multi-Valued Variable Diagrams. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 581–598.

Wang, R. 2024. Encoding Constraints as Binary Constraint Networks Satisfying BTP. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 8172–8181.

Wang, R.; Xia, W.; Yap, R. H. C.; and Li, Z. 2016. Optimizing Simple Tabular Reduction with a Bitwise Representation. In *International Joint Conference on Artificial Intelligence*, 787–795.

Wang, R.; and Yap, R. H. C. 2022a. CNF encodings of binary constraint trees. In *International conference on principles and practice of constraint programming*.

Wang, R.; and Yap, R. H. C. 2022b. Encoding Multi-valued Decision Diagram Constraints as Binary Constraint Trees. In *AAAI National Conference on Artificial Intelligence*, 3850–3858.

Wang, R.; and Yap, R. H. C. 2023. The expressive power of ad-hoc constraints for modelling CSPs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 4104–4114.

Yap, R. H. C.; Xia, W.; and Wang, R. 2020. Generalized Arc Consistency Algorithms for Table Constraints: A Summary of Algorithmic Ideas. In *AAAI Conference on Artificial Intelligence*, 13590–13597.