CNF Encodings of Binary Constraint Trees

Ruiwei Wang
School of Computing, National University of Singapore, Singapore

Roland H. C. Yap
School of Computing, National University of Singapore, Singapore

Abstract

Ordered Multi-valued Decision Diagrams (MDDs) have been shown to be useful to represent finite domain functions/relations. For example, various constraints can be modelled with MDD constraints. Recently, a new representation called Binary Constraint Tree (BCT), which is a (special) tree structure binary Constraint Satisfaction Problem, has been proposed to encode MDDs and shown to outperform existing MDD constraint propagators in Constraint Programming solvers. BCT is a compact representation, and it can be exponentially smaller than MDD for representing some constraints. Here, we also show that BCT is compact for representing non-deterministic finite state automaton (NFA) constraints. In this paper, we investigate how to encode BCT into CNF form, making it suitable for SAT solvers. We present and investigate five BCT CNF encodings. We compare the propagation strength of the BCT CNF encodings and experimentally evaluate the encodings on a range of existing benchmarks. We also compare with seven existing CNF encodings of MDD constraints. Experimental results show that the CNF encodings of BCT constraints can outperform those of MDD constraints on various benchmarks.

2012 ACM Subject Classification Computing methodologies → Artificial intelligence; Software and its engineering → Constraint and logic languages

Keywords and phrases BCT, CNF, MDD, NFA / MDD constraint, propagation strength

Digital Object Identifier 10.4230/LIPIcs.CP.2022.40

Funding This work was supported in part by the National Research Foundation Singapore under its AI Singapore Programme [AISG-RP-2018-005] and grant T1 251RES2024.

1 Introduction

Ordered Multi-valued Decision Diagram (MDD) [39] is a compact representation which can be used to encode finite domain functions/relations. Many constraints can be encoded into compact MDD constraints, such as the regular constraints [32], table constraints [10], among and sequence constraints [22]. MDD constraints are also useful to model problems requiring specific constraints which are not readily modelled with existing known constraints [11, 21]. In Constraint Programming (CP) solvers, MDD constraints can be directly handled with MDD Generalized Arc Consistency (GAC) propagators, e.g. the MDDc [10], MDD4R [31], CD [42] and CD^{bs} [43] propagators. Alternatively, MDD constraints can also be solved by SAT solvers by encoding MDD constraints into CNF form [1]. In this way, SAT solvers can directly handle the constraints which can be modelled with MDDs constraints[2, 3, 5].

Binary constraint is also a general representation for constraints. Any non-binary constraint can be transformed into binary constraints through binary encodings such as dual encoding [13], hidden variable encoding [37], double encoding [40] and bipartite encoding [47]. Recently, binary encodings with specialized Arc Consistency (AC) propagators [46, 47] has been shown to outperform the GAC propagators of non-binary table constraints [28, 45, 15, 50]. Similar to MDDs, the binary constraints can also be encoded into CNF with different CNF encodings, such as the log encoding [23, 44, 41], direct encoding [44] and support encoding [24, 19].

Recently, a new representation called Binary Constraint Tree (BCT) [48], which is a set of binary constraints with tree structures (a special binary CSP), has been proposed to encode MDDs. BCT is a compact representation, and it can be exponentially smaller than MDD. In this paper, we also show that non-deterministic finite state automaton (NFA) constraints [33, 9] can be transformed into BCT constraints without exponential blow up but not MDD constraints. Furthermore, a GAC propagator of BCT constraints [48] has been shown to outperform the state-of-the-art MDD GAC propagators. The results in [48] show that BCT has great potential for encoding and reducing MDDs.

In this paper, we investigate how to encode BCT constraints into CNF instances and apply them in SAT solvers. We investigate five CNF encodings of BCT constraints, including the log encoding, direct encoding, support encoding and two new transformations: partial support encoding and minimal support encoding. We tailor three well-known CNF encodings of binary constraints, i.e. the log, direct and support encodings, to handle BCT constraints. In addition, we introduce the partial support encoding and minimal support encoding by eliminating clauses and Boolean variables from the support encoding of BCT constraints. Then we analyze the strength of unit propagation on these 5 CNF encodings of BCT constraints. The support encoding of BCT constraints, which implements propagation completeness [6], can have a greater propagation strength than the other CNF encodings. The partial support encoding and minimal support encoding are more compact than the support encoding but their propagation strength is weaker than the support encoding. The log encoding and direct encoding, which do not implement weak consistency, have the weakest propagation strength. We also compare the five CNF encodings of BCT constraints and seven existing MDD encodings [1] using the Kissat SAT solver [17] on a range of existing benchmarks. Our experimental results show that the CNF encodings of BCT constraints can outperform MDD CNF encodings.

The paper is organized as follows. Section 2 provides the preliminaries. Section 3 shows that BCT can be exponentially smaller than MDD on representing NFA constraints. Sections 4 and 5 respectively introduce CNF encodings of binary constraints and BCT constraints. Experimental results are given in Section 6, and Section 7 concludes.

2 Preliminaries

A CSP P is a pair (X, C) where X is a set of variables, $\mathcal{D}(x)$ is the domain of a variable x, and C is a set of constraints. A variable is a *Boolean variable* if $\mathcal{D}(x) = \{true, false\}$. A *literal* of a variable x is a variable value pair (x, a). A *tuple* over a set of variables $\{x_{i_1}, x_{i_2}, \ldots, x_{i_r}\}$ is denoted by a *set of literals* $\{(x_{i_1}, a_1), (x_{i_2}, a_2), \ldots, (x_{i_r}, a_r)\}$. Each constraint c has a constraint scope $scp(c) \subseteq X$ and a relation $rel(c)$ defined by a set of tuples over $scp(c)$. The arity of c is the number of variables in its scope, i.e. $|scp(c)|$. A constraint c is a *binary constraint* if $|scp(c)| = 2$. A constraint c over Boolean variables $\{x_{i_1}, \cdots, x_{i_r}\}$ is a *clause* if it is a disjunction of a set cl of literals $\{(x_{i_1}, a_1), \cdots, (x_{i_r}, a_r)\}$ such that $a_j \in \{true, false\}$ for $1 \leq j \leq r$, so $rel(c)$ consists of the tuples over $scp(c)$ including at least a literal in cl. A CSP is called a *binary CSP* if it only includes binary constraints. A binary CSP is *normalized* if all constraints have different scopes. A CSP (X, C) is called a *Conjunctive Normal Form (CNF)* if all variables in X are Boolean variables and all constraints in C are clauses.

Given any set of variables V and a tuple τ, we use $\tau[V] = \{(x, a) \in \tau | x \in V\}$ to denote a subset of τ, while $T[V] = \{\tau[V] | \tau \in T\}$ is the *projection* of tuples T on V. In addition, $P|_\tau$ denotes a subproblem $(\{x' | x \in X\}, C)$ of $P = (X, C)$ generated by assigning a tuple τ where $\mathcal{D}(x') = \mathcal{D}(x)$ if τ does not include any literal of x, otherwise $\mathcal{D}(x') = \{(x, a) \in \tau | a \in \mathcal{D}(x)\}$.

Note that $F|_\tau = F$ if $\tau = \emptyset$. A tuple τ is an assignment if $a \in \mathcal{D}(x)$ for all $(x, a) \in \tau$. An assignment τ over X is a solution of P if $\tau[scp(c)] \in rel(c)$ for all constraints $c \in C$. $sol(X, C)$ (or $sol(P)$) denotes all solutions of P. A CSP (X, C) is satisfiable if $Sol(X, C) \neq \emptyset$, otherwise it is unsatisfiable.

A *support of a value* $a \in \mathcal{D}(x)$ on a constraint c is a tuple $\tau \in rel(c)$ such that $(x, a) \in \tau$ and $b \in \mathcal{D}(y)$ for all $(y, b) \in \tau$. A variable $x \in scp(c)$ is *Generalized Arc Consistent* (GAC) on c if a has a support on c for all $a \in \mathcal{D}(x)$. c is GAC if all variables in $scp(c)$ are GAC on c. A CSP (X, C) is GAC if every constraint in C is GAC. For binary CSPs, GAC is also called *Arc Consistency* (AC). For any CNF F, GAC is also called *unit propagation*, where $UP(F)$ is used to denote a CNF generated from F by removing all variable values which are not GAC on F. If $UP(F)$ includes any empty variable domain, F is unsatisfiable.

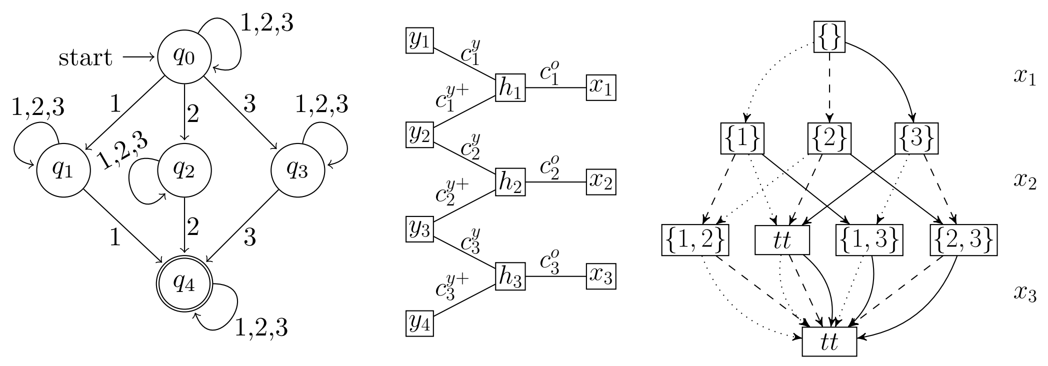
2.1 CNF encoding and unit propagation strength

A *CNF encoding* of a CSP $P = (X, C)$ is a CNF which is equisatisfiable with P, i.e. the CNF is satisfiable iff P is satisfiable. Typically, a CNF encoding consists of a variable encoding (VE) and a constraint encoding (CE) where VE encodes the variables in X as a set φ^X of Boolean variables and each constraint c in C corresponds to a constraint (Boolean function) φ^c over the Boolean variables, while CE encodes the constraint φ^c as a CNF F^c over Boolean variables Y such that $scp(\varphi^c) \subseteq Y$ and $sol(F^c)[scp(\varphi^c)] = rel(\varphi^c)$. There have been many VEs which can be used to transform finite domain variables into Boolean variables, such as direct encoding and log encoding [44] . In addition, there are also various CEs for encoding constraints, e.g. many CNF encodings of MDD constraints [1]. A way to analyze a CE is evaluating the strength of unit propagation on the encoded constraint F^c. We will use the following four levels to classify the strength of unit propagation on encoding F^c:

- F^c implements *weak consistency* if for any tuple τ over $scp(\varphi^c)$ such that $F^c|_\tau$ is unsatisfiable, some variable domains in $UP(F^c|_\tau)$ are empty.
- F^c implements *domain consistency* if for any tuple τ over a subset of $scp(\varphi^c)$ and a literal (v, a) in $UP(F^c|_\tau)$ such that $v \in scp(\varphi^c)$ and all variable domains in $UP(F^c|_\tau)$ are not empty, there is at least a solution of $F^c|_\tau$ including (v, a).
- F^c implements *unit refutation completeness* [14, 1, 25, 26] if for any tuple τ over a subset of Y such that $F^c|_\tau$ is unsatisfiable, some variable domains in $UP(F^c|_\tau)$ are empty.
- F^c implements *propagation completeness* [6, 1, 25, 26] if for any tuple τ over a subset of Y and a literal l in $UP(F^c|_\tau)$ such that all variable domains in $UP(F^c|_\tau)$ are not empty, there is at least a solution of $F^c|_\tau$ including l.

In this classification, propagation completeness is the strongest level, unit refutation completeness is incomparable with domain consistency, and weak consistency is the weakest level. Note that the definition of weak/domain consistency is defined for evaluating the strength of unit propagation and will be used in the rest of this paper.

▶ **Example 1.** Assume the CNF F^c is $(\{x, y, z\}, \{x \vee y, \neg y \vee z, x \vee \neg z\})$ and the scope $scp(\varphi^c)$ is $\{x, y\}$. $(x, false)$ is the only literal in F^c which cannot be extended to a solution of F^c. $(x, false)$ is in $UP(F^c)$, so F^c does not implement propagation completeness. Then x is in $scp(\varphi^c)$, thus, F^c also does not implement domain consistency. For any tuple τ over a subset of $\{x, y, z\}$, if $F^c|_\tau$ is unsatisfiable, $UP(F^c|_\tau)$ has empty variable domains. So F^c implements unit refutation completeness and weak consistency.



(a) NFA: nodes and edges respectively denote states and transitions.

(b) BCT: nodes and edges are variables and constraints.

(c) MDD: dotted (dashed and solid) lines denote the value 1 (2 and 3).

Figure 1 Different representations of a constraint over 3 variables $\{x_1, x_2, x_3\}$.

3 BCT versus MDD on representing NFA constraints

Binary Constraint Tree (BCT) is a compact representation which can be exponentially smaller than Ordered Multi-valued Decision Diagram (MDD). In this section, we show that any NFA constraint can be encoded as a BCT constraint without exponential blow up, where NFA may be exponentially smaller than the corresponding MDD.

A non-deterministic finite state automaton (NFA) is a quintuple $(Q, \sum, \Delta, q_0, Q_t)$ consisting of a finite set Q of states, a finite set $\sum$ of input symbols, a transition function $\Delta : Q \times \sum \to 2^Q$, an initial state $q_0 \in Q$, and a set $Q_t \subseteq Q$ of accepting states, where there is a transition in the NFA from a state $q_i \in Q$ to a state $q_j \in Q$ via a symbol $a \in \sum$ if $q_i \in \Delta(q_j, a)$. A string $a_1...a_r$ is accepted by the NFA if there is a sequence of states, $s_0, s_1, ..., s_r$, such that: $s_0 = q_0$, $s_i \in Q$ and $s_{i+1} \in \Delta(s_i, a_{i+1})$ for $0 \leq i < r$, and $s_r \in Q_t$. A NFA constraint c is a pair (G, O) such that O is an ordering over $scp(c)$, G is a NFA and $rel(c)$ is the set of tuples $\{(O_1, a_1), ..., (O_r, a_r)\}$ over $scp(c)$ such that the string $a_1...a_r$ is accepted by the NFA [33, 9].

▶ **Example 2.** Consider the constraint $\bigvee_{i=1}^{r} \bigvee_{j=i+1}^{r} (x_i = x_j)$ over r variables $\{x_1, ..., x_r\}$ with variable domain $\{1, .., r\}$, which expresses the negation of an *alldifferent* constraint [35]. The size of the negation of the MDDs (Ordered Multi-valued Decision Diagrams) representing alldifferent constraints is exponential in r [4].

We can use a NFA $(\{q_0, ..., q_{r+1}\}, \{1, ..., r\}, \Delta, q_0, \{q_{r+1}\})$ to model the constraint such that $\Delta(q_0, i) = \{q_0, q_i\}$, $\Delta(q_i, i) = \{q_i, q_{r+1}\}$, $\Delta(q_{r+1}, i) = \{q_{r+1}\}$ and $\Delta(q_j, i) = \{q_j\}$ for $1 \leq i \leq r$ and $j \neq 0, i, r+1$. Figure 1a gives a NFA for $r = 3$, and Figure 1c is a MDD modelling the NFA, where every subset of the domain corresponds to a node in the MDD.

▶ **Definition 3** (BCT and BCT constraint [48])**.** *A* Binary Constraint Tree *(BCT) is a normalized binary CSP whose constraint graph is a tree. A* BCT constraint *c is a pair (V, P) such that $P = (X, C)$ is a BCT, $scp(c) = V$, $V \subseteq X$ and $rel(c) = sol(X, C)[V]$, where the variables in $scp(c)$ and $X \setminus scp(c)$ are respectively called the* original *and* hidden *variables.*

We recap BCT, see [48] for more details. BCT is viewed as a single non-binary constraint, the BCT constraint, modelled as a binary CSP with hidden variables. Given the tree structure of the BCT, AC on the BCT can achieve GAC on the BCT constraint. It has been shown in [48] that any MDD constraint can be encoded into a BCT constraint with the same size as the MDD. A BCT can be further optimized with the reduction rules given in [48]. After the reduction, BCT constraints can be much smaller than the corresponding MDD constraints.

3.1 Direct tree binary encoding

For any NFA constraint c^* over r variables, we can encode the states and transitions of the NFA into a sequence of hidden variables, and then representing the NFA constraint c^* as a BCT over the hidden variables such that every tuple in the constraint relation denotes a sequence of r transitions from the initial state to an accepting state in the NFA. The details of the encoding are given in Definition 4.

▶ **Definition 4.** *A* direct tree binary encoding *(DTBE) of a NFA constraint $c^* = (G, O)$ is a BCT $dtbe(c^*)$= $(Y \cup H \cup scp(c^*), \{c^o_1, c^y_1, c^{y+}_1, ..., c^o_r, c^y_r, c^{y+}_r\})$ where*

- *$r = |scp(c^*)|$ and $G = (Q, \sum, \Delta, q_0, Q_t)$ and $Y = \{y_1, ..., y_{r+1}\}$ and $H = \{h_1, ..., h_r\}$;*
- *$scp(c^o_i) = \{O_i, h_i\}$, $scp(c^{y+}_i) = \{y_{i+1}, h_i\}$ and $scp(c^y_i) = \{y_i, h_i\}$;*
- *$\mathcal{D}(y_1) = \{q_0\}$, $\mathcal{D}(y_{r+1}) = Q_t$ and $\mathcal{D}(y_i) = Q$ for $2 \leq i \leq r$;*
- *$\mathcal{D}(h_i)$ is the set of transitions in G for $1 \leq i \leq r$;*
- *$rel(c^y_i) = \{\{(h_i, tr), (y_i, b)\} | tr \in \mathcal{D}(h_i)$, tr is a transition from $b\}$;*
- *$rel(c^o_i) = \{\{(h_i, tr), (O_i, s)\} | tr \in \mathcal{D}(h_i)$, s is the symbol of the transition $tr\}$;*
- *$rel(c^{y+}_i) = \{\{(h_i, tr), (y_{i+1}, b)\} | tr \in \mathcal{D}(h_i)$, tr is a transition to $b\}$.*

We remark that we use the term DTBE for NFA constraints in the same way as the DTBE encoding of MDD in [48], we refer to [48] for more details. Figure 1b shows the constraint graph of a DTBE for a NFA constraint (G, O), where G is given in Figure 1a and O is the variable order $x_1 \leq x_2 \leq x_3$. The states and transitions in the NFA are respectively encoded as the hidden variables $y_1, \cdots, y_4$ and $h_1, \cdots, h_3$, where the states and transitions are encoded as hidden variable values, i.e. $\mathcal{D}(y_1) = \{q_0\}$, $\mathcal{D}(y_2) = \mathcal{D}(y_3) = \{q_0, q_1, q_2, q_3, q_4\}$, $\mathcal{D}(y_4) = \{q_4\}$ and the domain of h_1, h_2, h_3 is the set of all transitions $\{(q_i, a, q_j) | q_j \in \Delta(q_i, a), 0 \leq i \leq 4, 1 \leq a \leq 3\}$ in the NFA. Then binary constraints are used to combine transitions with symbols and states. The binary constraint relations can be constructed based on Definition 4. For example, $rel(c^y_1) = \{\{(y_1, q_0), (h_1, (q_0, a, q_i))\} | i \in \{0, a\}, a \in \{1, 2, 3\}\}$, $rel(c^{y+}_1) = \{\{(y_2, q_i), (h_1, (q_0, a, q_i))\} | i \in \{0, a\}, a \in \{1, 2, 3\}\}$, $rel(c^o_1)$ $= \{\{(x_1, a), (h_1, (q_0, a, q_i))\} | i \in \{0, a\}, a \in \{1, 2, 3\}\}$. In addition, the reduction rules given in [48] can also be directly used to reduce the DTBE encodings of NFA constraints.

▶ **Theorem 5.** *BCT can be exponentially smaller than MDD on representing NFA constraints.*

The proof of Theorem 5 (also Lemma 10 and Propositions 11, 13, 14, 16) can be found in the Appendix. Theorem 5 shows that there exists a family of NFA constraints (Example 2) for which the BCT representation (DTBE) is exponentially smaller than the MDD representation. For example, give the NFA constraint in Example 2 with $r = 15$, the BCT representation (after reduction) has 2K values and 6K tuples which is much smaller than the MDD representation having 33K nodes and 491K edges.

4 CNF encodings for binary constraints

In this section, we introduce three well-known CNF encodings, i.e. log encoding [23, 44, 41], direct encoding [44] and support encoding [24, 19], which are used to transform any binary CSP (X, C) into CNF. These CNF encodings represent each variable x in X as a set of Boolean variables such that every value in $\mathcal{D}(x)$ corresponds to exactly one tuple over the variables. In addition, for each binary constraint $c \in C$, the encodings use clauses to represent the tuples τ over $scp(c)$ such that $\tau \notin rel(c)$ (or $\tau \in rel(c)$). These CNF encodings can be directly used to encode BCT constraints, since any BCT is also a binary CSP.

4.1 Log encoding

Every variable $x \in X$ is represented as $k = \lceil \log_2(d) \rceil$ Boolean variables $V^x = \{v_1^x, ..., v_k^x\}$, where $d = |\mathcal{D}(x)|$. Let T be the set of the first d assignments over B^x in the lexicographic order. Every value a in $\mathcal{D}(x)$ corresponds to a tuple τ_a in T. In addition, if $|\mathcal{D}(x)|$ is not a power of two, the assignments over V^x which are not in T can be excluded by adding the following clause [41] for each literal $(v_i^x, false)$ in the d^{th} (last) tuple τ in T

$$f(v_1^x) \vee \cdots \vee f(v_{i-1}^x) \vee \neg v_i^x \qquad \textit{where} \quad f(v_j^x) = \begin{cases} v_j^x & \text{if } (v_j^x, false) \in \tau \\ \neg v_j^x & \text{if } (v_j^x, true) \in \tau \end{cases}$$

For each constraint $c \in C$ and an assignment $\{(x,a),(y,b)\}$ over $scp(c)$, if the tuple is not in $rel(c)$, then the following clause is added to exclude the tuple

$$\Big(\bigvee_{(v_j^x, true) \in \tau_a \cup \tau_b} \neg v_j^x \Big) \vee \Big(\bigvee_{(v_j^x, false) \in \tau_a \cup \tau_b} v_j^x \Big)$$

▶ **Example 6.** The log encoding of the binary CSP $P = (\{x_1,x_2,x_3,x_4\}, \{x_1 + x_3 \le 5, x_3 = 3 \vee x_4 = 3, x_2 + x_4 \le 5\})$, where variable domains are $\{1,2,3\}$, consists of 8 Boolean variables $\{v_1^{x_1}, v_2^{x_1}, v_1^{x_2}, v_2^{x_2}, v_1^{x_3}, v_2^{x_3}, v_1^{x_4}, v_2^{x_4}\}$ and 10 clauses:

$$\begin{array}{cccc} \neg v_1^{x_1} \vee \neg v_2^{x_1} & \neg v_1^{x_2} \vee \neg v_2^{x_2} & \neg v_1^{x_3} \vee \neg v_2^{x_3} & \neg v_1^{x_4} \vee \neg v_2^{x_4} \end{array}$$

$$\begin{array}{ccc} \neg v_1^{x_1} \vee v_2^{x_1} \vee \neg v_1^{x_3} \vee v_2^{x_3} & \neg v_1^{x_2} \vee v_2^{x_2} \vee \neg v_1^{x_4} \vee v_2^{x_4} & v_1^{x_3} \vee v_2^{x_3} \vee v_1^{x_4} \vee v_2^{x_4} \\ v_1^{x_3} \vee \neg v_2^{x_3} \vee v_1^{x_4} \vee v_2^{x_4} & v_1^{x_3} \vee v_2^{x_3} \vee v_1^{x_4} \vee \neg v_2^{x_4} & v_1^{x_3} \vee \neg v_2^{x_3} \vee v_1^{x_4} \vee \neg v_2^{x_4} \end{array}$$

4.2 Direct encoding

Every variable $x \in X$ is represented as d Boolean variable $B^x = \{v_{a_i}^x | a_i \in \mathcal{D}(x)\}$, where $\mathcal{D}(x) = \{a_1, \cdots, a_d\}$. In addition, an exactly-one constraint over B^x is introduced to guarantee that if a variable in B^x is assigned with *true*, then the other variables in B^x are assigned with *false*. The exactly-one constraint is encoded with *ladder encoding* [20] which includes $d-1$ additional Boolean variables $A^x = \{w_1^x, \cdots, w_{d-1}^x\}$ and a set of $EO(x)$ clauses:

$$\begin{array}{cccc} \neg v_{a_1}^x \vee \neg w_1^x & v_{a_1}^x \vee w_1^x & v_{a_d}^x \vee \neg w_{d-1}^x & \neg v_{a_d}^x \vee w_{d-1}^x \end{array}$$
$$\{w_{i-1}^x \vee \neg w_i^x, v_{a_i}^x \vee w_i^x \vee \neg w_{i-1}^x, \neg v_{a_i}^x \vee \neg w_i^x, \neg v_{a_i}^x \vee w_{i-1}^x | 2 \le i \le d-1\}$$

The latter encoding implements propagation completeness [6, 25]. The clauses in $EO(x)$ can guarantee that every value a_i in $\mathcal{D}(x)$ corresponds to exactly one solution $t(x, a_i)$ of the CNF $(B^x \cup A^x, EO(x))$, where $t(x,a_i) = \{(v_b^x, false) | b \in \mathcal{D}(x), b \neq a_i\} \cup \{(v_{a_i}^x, true)\} \cup \{(w_j^x, true) | 1 \le j < i\} \cup \{(w_j^x, false) | i \le j < d\}$. Then for each $c \in C$ and an assignment $\tau = \{(x, a_i), (y, b_j)\}$ over $scp(c)$ such that $\tau \notin rel(c)$, the clause $\neg v_{a_i}^x \vee \neg v_{b_j}^y$ is added.

▶ **Example 7.** The direct encoding of the binary CSP given in Example 6 includes 20 Boolean variables and the following clauses:

$$\{\neg v_1^{x_j} \vee \neg w_1^{x_j}, v_1^{x_j} \vee w_1^{x_j}, v_3^{x_j} \vee \neg w_2^{x_j}, \neg v_3^{x_j} \vee w_2^{x_j} | 1 \leq j \leq 4\}$$
$$\{w_1^{x_j} \vee \neg w_2^{x_j}, v_2^{x_j} \vee w_2^{x_j} \vee \neg w_1^{x_j}, \neg v_2^{x_j} \vee \neg w_2^{x_j}, \neg v_2^{x_j} \vee w_1^{x_j} | 1 \leq j \leq 4\}$$
$$\neg v_3^{x_1} \vee \neg v_3^{x_3} \quad \neg v_3^{x_2} \vee \neg v_3^{x_4} \quad \neg v_1^{x_3} \vee \neg v_2^{x_4} \quad \neg v_1^{x_3} \vee \neg v_1^{x_4} \quad \neg v_2^{x_3} \vee \neg v_1^{x_4} \quad \neg v_2^{x_4} \vee \neg v_2^{x_4}$$

4.3 Support encoding

Every variable $x \in X$ is represented as d Boolean variable $B^x = \{v_{a_i}^x | a_i \in \mathcal{D}(x)\}$, where $\mathcal{D}(x) = \{a_1, \cdots, a_d\}$, and an exactly-one constraint over B^x is used to make sure that each value in $\mathcal{D}(x)$ corresponds to exactly one tuple over B^x. The exactly-one constraint is encoded with $(A^x \cup B^x, EO(x))$, i.e. ladder encoding. In addition, for each value $a \in \mathcal{D}(x)$ and a constraint $c \in C$ such that $scp(c) = \{x, y\}$, a clause $cl(x, a, c)$ is added where

$$cl(x, a, c) = \neg v_a^x \vee (\bigvee_{\{(x,a),(y,b)\} \in rel(c) \wedge b \in \mathcal{D}(y)} v_b^y)$$

By using the clauses $cl(x, c) = \{cl(x, a, c) | c \in C, x \in scp(c), a \in \mathcal{D}(x)\}$, unit propagation on the support encoding of a binary CSP can achieve AC on the binary CSP [19].

5 CNF encoding for BCT constraints

For any BCT constraint (V, P), the CNF encodings of binary constraints can be directly used to encode the BCT constraint, because the BCT P is a binary CSP. In addition, binary constraints are special cases of BCT constraints, i.e. every binary constraint is a BCT constraint without any hidden variables. When comparing the unit propagation, there is a subtlety, the strength of unit propagation on the CNF encodings of BCT constraints can be different from that for binary constraints. In this section, we will discuss the strength of unit propagation when using the log, direct and support encodings to encode BCT constraints as CNF. Afterwards, we will propose two further CNF encodings of BCT constraints by eliminating Boolean variables and clauses from the support encoding of the constraints.

5.1 Encodings from binary constraints

The log encoding of binary constraints implements weak consistency but we highlight that it does not do so for BCT constraints, since BCT constraints can include hidden variables. Assume P is the binary CSP given in Example 6 and F is the log encoding of the BCT constraint $(\{x_1, x_2\}, P)$. Then F does not implement weak consistency. For example, the tuple τ=$\{(v_1^{x_1}, true), (v_2^{x_1}, false), (v_1^{x_2}, true), (v_2^{x_2}, false)\}$ is not included by any solution of F, i.e. $F|_\tau$ is unsatisfiable but $UP(F|_\tau)$ does not include any empty variable domain. Therefore, the log encoding of BCT constraints does not implement weak consistency.

▶ **Proposition 8.** *Log encoding of BCT constraints does not implement weak consistency.*

Similarly, the direct encoding of BCT constraints also does not implement weak consistency. For example, the tuple τ=$\{(v_3^{x_1}, true), (v_3^{x_2}, true)\}$ is not included by any solution of the direct encoding F (given in Example 7) of the BCT constraint $(\{x_1, x_2\}, P)$, i.e. $F|_\tau$ is unsatisfiable, but $UP(F|_\tau)$ does not include any empty variable domain.

▶ **Proposition 9.** *Direct encoding of BCT constraints does not implement weak consistency.*

Unit propagation on the support encoding of a binary CSP can achieve AC on the binary CSP. Further, the constraint graph of a BCT is a tree. Hence, unit propagation on the support encoding of a BCT constraint can achieve GAC on the BCT constraint. For any BCT (X, C), we can set a variable $x \in X$ as the root and construct a tree order O over X such that $O_1 = x$, where O is a *tree order* if for any $j > 1$ and $O_j \in X$, there is exactly one constraint $c \in C$ such that $scp(c) = \{O_i, O_j\}$ and $i < j$. In addition, we use T^O to denote a set of clauses $\bigcup\{cl(O_i, c) | c \in C, scp(c) = \{O_i, O_j\}, i < j\}$ with respect to a tree order O.

▶ **Lemma 10.** *Given a BCT $P = (X, C)$ and a tree order O over X, if a literal $(v_a^{O_1}, true)$ is included in $UP(F)$ and all variable domains in $UP(F)$ are not empty, there is $\tau \in sol(P)$ such that $(O_1, a) \in \tau$ and $(v_b^x, true)$ is included in F for all $(x, b) \in \tau$, where $F = (A, C^A)|_{\tau'}$ and $B^x \subseteq A$ for all $x \in X$ and $T^O \subseteq C^A$ and τ' is a tuple over a subset of A.*

We now show the support encoding of BCT constraints implements propagation completeness.

▶ **Proposition 11.** *The support encoding $F = (A \cup B, T \cup E)$ of BCT constraints (V, P) implements propagation completeness, where P is a BCT (X, C) and $A = \bigcup_{x \in X} A^x$ and $B = \bigcup_{x \in X} B^x$ and $T = \{cl(x, c) | c \in C, x \in scp(c)\}$ and $E = \bigcup_{x \in X} EO(x)$.*

5.2 Partial support encoding

We now introduce a new CNF encoding of BCT constraints, called partial support encoding, by eliminating clauses from the support encoding of the BCT constraints. Give any BCT $P = (X, C)$ and variables $V \subseteq X$, the *partial support encoding* of the BCT constraint (V, P) is a CNF $F = (A^V \cup B^V \cup B^H, T \cup E^V)$, where $A^V = \bigcup_{x \in V} A^x$ and $B^V = \bigcup_{x \in V} B^x$ and $B^H = \bigcup_{h \in X \setminus V} B^h$ and $T = \{cl(x, c, a) | c \in C, x \in scp(c), a \in \mathcal{D}(x)\}$ and $E^V = \bigcup_{x \in V} EO(x)$. Partial support encoding has the same Boolean variables and clauses as the support encoding of (V, P), except that the clauses in $EO(h)$ and the Boolean variables in A^h are removed from the support encoding of (V, P) for any hidden variables h in $X \setminus V$.

For each solution τ of P, we can construct a solution of F, e.g. $(\bigcup_{(x,a) \in \tau} t(x, a)) \cup \{(v_a^x, true) | (x, a) \in \tau\} \cup \{(v_a^x, false) | (x, a) \notin \tau, a \in \mathcal{D}(x)\}$. Conversely, every solution τ of F corresponds to at least one solution of P (see Lemma 10), since variable domains in $UP(F|_\tau)$ are not empty. However, the strength of unit propagation on the partial support encoding is weaker than that on the support encoding for BCT constraints. Partial supporting encoding implements domain consistency and unit refutation completeness but not propagation completeness.

For the partial support encoding F of a BCT $(\{x_1, x_2\}, P)$ where P is the binary CSP (BCT) given in Example 6, the literal $(v_3^{x_3}, false)$ cannot be extended to any solution of the CNF $F|_\tau$ but $(v_3^{x_3}, false)$ is in $UP(F|_\tau)$, where $\tau = \{(v_3^{x_4}, false), (v_1^{x_3}, false), (v_2^{x_3}, false), (v_3^{x_1}, false)\}$ is a tuple over the Boolean variables $\tau = \{v_3^{x_4}, v_1^{x_3}, v_2^{x_3}, v_3^{x_1}\}$. Therefore, partial support encoding of BCT constraint does not implement propagation completeness.

▶ **Proposition 12.** *Partial support encoding for BCT constraints does not implement propagation completeness.*

Partial support encoding implements unit refutation completeness (based on Lemma 10), thus, it also implements weak consistency.

▶ **Proposition 13.** *The partial support encoding $F = (A^V \cup B^V \cup B^H, T \cup E^V)$ of a BCT constraint (V, P) implements unit refutation completeness where $P = (X, C)$.*

In addition, from Lemma 10, we can also get that partial support encoding implements domain consistency.

▶ **Proposition 14.** *The partial support encoding $F = (A^V \cup B^V \cup B^H, T \cup E^V)$ of a BCT constraint (V, P) implements domain consistency where $P = (X, C)$.*

5.3 Minimal support encoding

We now give a more compact CNF encoding of BCT constraints called minimal support encoding. Give any BCT $P = (X, C)$ and variables $V \subseteq X$, the *minimal support encoding* of the BCT constraint (V, P) with respect to a tree order O over X is a CNF $F = (A^V \cup B^V \cup B^H, T^O \cup E^V)$, where $P = (X, C)$ and $A^V = \bigcup_{x \in V} A^x$ and $B^V = \bigcup_{x \in V} B^x$ and $B^H = \bigcup_{h \in X \setminus V} B^h$ and $E^V = \bigcup_{x \in V} EO(x)$ and $O_1 \in V$. Minimal support encoding has the same Boolean variables as the partial support encoding but the minimal support encoding does not include the clauses $cl(O_j, c)$ for any binary constraint $c \in C$ between 2 variables $O_i, O_j \in X$ such that $i < j$.

The strength of unit propagation on minimal support encoding is weaker than that on partial support encoding. For the minimal support encoding F of a BCT $(\{x_1, x_2\}, P)$ with respect to a tree order $x_1 < x_3 < x_4 < x_2$ and a tuple $\tau = \{(v_3^{x_1}, true)\}$ where P is the binary CSP given in Example 6, the literal $(v_3^{x_4}, true)$ cannot be extended to a solution of the CNF $F|_\tau$ but $(v_3^{x_4}, true)$ is included in $UP(F|_\tau)$. So the minimal support encoding does not implement domain consistency and propagation completeness.

▶ **Proposition 15.** *Minimal support encoding for BCT constraints does not implement domain consistency and propagation completeness.*

In addition, the minimal support encoding is stronger than the log and direct encodings for BCT constraints. From Lemma 10, we can get that the minimal support encoding implements unit refutation completeness and weak consistency.

▶ **Proposition 16.** *The minimal support encoding F of a BCT constraint (V, P) with respect to a tree order O implements unit refutation completeness where $x = O_1$ and $P = (X, C)$.*

Table 1 summarizes the strength of unit propagation on all five CNF encoding of BCT constraints. The support encoding of BCT constraints, which implements propagation completeness, is the strongest encoding. Then the partial support encoding implementing domain consistency is stronger than the log, direct and minimal support encodings. In addition, the log and direct encodings of BCT constraints are weaker than the minimal support encoding, since the log and direct encodings of BCT constraints do not implement weak consistency.

Table 1 Strength of Unit Propagation on various encodings of BCT constraints. The label ✓(✗) denotes that the CNF encodings (does not) implement a unit propagation strength level.

	Log	Direct	Minimal Support	Partial Support	Support
Weak consistency	✗	✗	✓	✓	✓
Domain consistency	✗	✗	✗	✓	✓
Unit refutation completeness	✗	✗	✓	✓	✓
Propagation completeness	✗	✗	✗	✗	✓

6 Experiments

We evaluate our five CNF encodings of BCT constraints, i.e. the log encoding, direct encoding, support encoding, PS (partial support) encoding and MS (minimal support) encoding, with seven existing CNF encodings [1] of MDD constraints, i.e. the Min (minimal), GMin (GenMiniSAT), Tes (Tseitin), BaP (Basic path), LevP (level path), NNFP (NNF path) and ComP (complete path) encodings. We employ the Kissat SAT solver [17] in default configuration to solve the resulting CNF. We also test a BCT GAC propagator [48] in the Abscon solver [29],[1] where the Abscon solver uses the binary branching MAC and geometric restart strategy[2], lexical value heuristic and five choices of variable heuristics (Lexical, DDeg [38], WDeg [7], Activity [30] and Impact [34]). We highlight the Abscon results are to compare CNF encodings with a SAT solver with a GAC propagator in a CP solver.

Experiments were run on a 3.20GHz Intel i7-8700 machine. Solving time is limited to 10 minutes per instance and memory to 12G. We tested the encodings with existing benchmarks also used by other papers [18, 9], and instances from the 2019 XCSP competition[3] and Minizinc challenge 2021.[4]

Tables 2, 3, 4 and 5 show the average solving times (in seconds) of the CNF encodings and the Abscon solver, and if there are some instances which cannot be solved in 10 minutes, then the tables give the number of unsolvable instances, i.e. the number of time-out or memory-out instances. The "Inst." and "# I" columns respectively give the names of different instance series and the numbers of CSP instances used. The "# Sol" row shows the total number of instances solved in 10 minutes. The best results of all methods are in bold, and the underlined results are the best results of the CNF encodings. The "Itime" row is the average initialization time (in seconds) of different methods and includes the encoding times. In addition, the "Itime" row also shows the number of memory-out instances if the CNF encoding runs out of memory during initialization. For different benchmarks, the Abscon solver results use the best variable heuristic from the 5 heuristics.

Figures 2a, 3a, 4a and 5a compare the performance profiles [16] of the methods VB-BCT, VB-MDD and VB-Abscon, where VB-BCT and VB-MDD respectively denote the virtual best CNF encoding of BCT constraints and MDD constraints respectively. VB-Abscon is the Abscon solver using the virtual best variable heuristic. The y-axis is the percentage of instances solved and the x-axis is the ratio of the solving time of a method to the virtual best method (time ratio). In the figures, we remove (i) the trivial instances which can be solved by all methods (VB-BCT, VB-MDD, and VB-Abscon) in 2 seconds and (ii) the instances which cannot be solved by any method in 10 minutes. Figures 2b, 3b, 4b and 5b use scatter plots to compare the solving times of various CNF encodings. In the figures, each dot denotes a CSP instance of a series, and the dot shapes correspond to instance series. In order to use logarithmic scales, the time on the x/y-axis is set as (1 + solving time). Figures 2c, 3c, 4c and 5c compare the number of clauses of different CNF encodings given by the x and y-axis.

[1] We have used the CP BCT propagator from [48] implemented in Abscon.
[2] The initial $cutoff = 10$ and $\rho = 1.1$. For each restart, *cutoff* is the allowed number of failed assignments and *cutoff* increases by $(cutoff \times \rho)$ after restart.
[3] `http://www.cril.univ-artois.fr/XCSP19/`
[4] `https://www.minizinc.org/challenge.html`

6.1 Benchmark Series 1: NFA

We use the 6 NFA series which are also used in [9]. These NFA benchmarks are modelled with NFA constraints. The NFA constraints can be transformed into BCT constraints and also MDD constraints. The direct tree binary encoding introduced in Section 3 are used to encode NFA constraints as BCT constraints, and then the BCT constraints are further reduced with the reduction rules proposed in [48]. In addition, the automaton library dk.brics.automaton[5] is used to minimize and transform any NFA into a DFA, where the DFA is directly expanded into the corresponding (quasi-reduced [1]) MDD.

Table 2 NFA benchmarks.

		BCT					MDD							Abscon
Inst.	#I	Log	Direct	Support	PS	MS	Min	GMin	Tes	BaP	LevP	NNFP	ComP	(DDeg)
NFA-50	14	1 out	166.42	11.62	7.76	<u>4.86</u>	7.91	2 out	3 out	102.18	126.79	109.96	107.57	**1.21**
NFA-36	18	16 out	10 out	61.69	30.40	<u>19.53</u>	1 out	15 out	15 out	11 out	11 out	11 out	13 out	**4.98**
NFA-34	13	89.63	14.57	2.20	1.38	<u>1.25</u>	1.78	16.56	24.90	19.59	25.25	19.16	21.17	**0.54**
NFA-54	15	15 out	14 out	179.23	91.95	<u>69.11</u>	2 out	15 out	15 out	15 out	15 out	15 out	15 out	**11.62**
NFA-57	15	14 out	10 out	1 out	55.42	<u>27.07</u>	1 out	13 out	15 out	10 out	11 out	11 out	12 out	**6.40**
NFA-60	15	14 out	6 out	30.13	16.80	<u>10.13</u>	25.79	12 out	15 out	2 out	3 out	3 out	2 out	**2.50**
#Sol	90	30	50	89	<u>**90**</u>	<u>**90**</u>	86	33	27	52	50	50	48	**90**
Itime	90	0.51	0.38	0.25	0.23	0.23	290.65	292.62	297.76	2 out	6 out	12 out	13 out	2.10

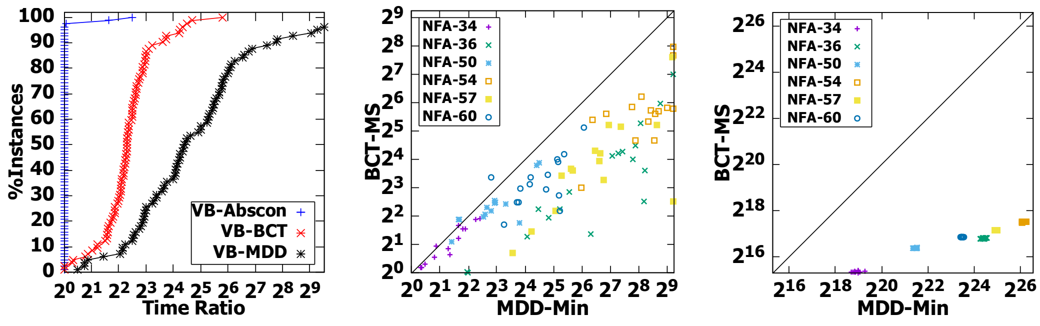
Table 2 shows the average result of the NFA instances. The NFA constraints used in the instances are much smaller than the corresponding MDDs, and the resulting encodings of BCT constraints also fit in memory. However, for MDDs being larger, the MDD CNF encodings BaP, LevP, NNFP and ComP run out of memory (memory-out). For these encodings, there are 2, 6, 12 and 13 memory-out NFA instances in the NFA-54 series, respectively. We remark that if an CNF encoding becomes too large for an instance, it simply cannot be used. The average initialization time of encoding MDD constraints are much larger than that of encoding BCT constraints, e.g. the Itime of Min is 290 seconds but that of MS is less than 1 second. The CNF encodings of BCT constraints are much faster than those of MDD constraints. The MS and PS encodings can solve all 90 NFA instances in 10 minutes but the best CNF encoding of MDD constraints, i.e. the Min encoding, only solves 86 instances. The best result for this series is the Abscon propagator with the DDeg variable heuristic.

Figure 2a shows that the best overall result for the NFA instances is the VB-Abscon propagator followed by the CNF encodings where VB-BCT overall outperforms VB-MDD on solving the NFA instances. The MS encoding is more compact than the Min encoding, where Min has the best CNF encoding result of MDD constraints for these NFA benchmarks. For example, the number of clauses in MS can be up to 400 times less than that in Min (Figure 1c gives the overall comparison). Correspondingly, MS has potential to be faster than Min. Figure 1b shows that MS is faster than Min on almost all tested NFA instances.

6.2 Benchmark Series 2: Pentominoes

We use all 192 Pentominoes instances from the the pentominoes generator website.[6] Some of the Pentominoes instances were also used in the Minizinc challenge 2021. The instances are separated into 4 series, P-5, P-10, P-15 and P-20, where P-k denotes the instances using

[5] `http://www.brics.dk/automaton/`
[6] `https://github.com/zayenz/minizinc-pentominoes-generator`



(a) Virtual best comparison. **(b)** BCT-MS vs MDD-Min. **(c)** The number of clauses.

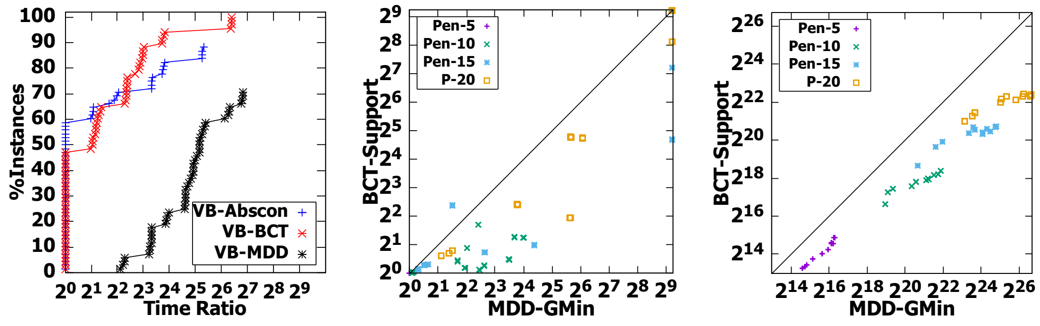
Figure 2 NFA benchmarks.

a $k \times k$ board. The constraints used in these benchmarks are represented with regular expressions (see [27] for more details). We use the dk.brics.automaton library to encode any regular expression into a DFA, and then directly expand the DFA into a MDD. The enocoding from [48] is used to transform any MDD constraint into a BCT constraint.

Table 3 Pentominoes benchmarks.

		BCT					MDD							Abscon
Inst.	#I	Log	Direct	Support	PS	MS	Min	GMin	Tes	BaP	LevP	NNFP	ComP	(Lex.)
P-5	48	0.27	**<0.01**	**<0.01**	**<0.01**	**<0.01**	0.05	0.01	0.06	0.02	0.02	0.02	0.02	0.02
p-10	48	154.70	32.04	**0.60**	0.76	3.62	72.86	4.94	31.65	32.08	17.73	19.42	18.28	9.95
P-15	48	36 out	24 out	**16 out**	20 out	20 out	24 out	24 out	24 out	24 out	24 out	24 out	28 out	20 out
P-20	48	44 out	24 out	16 out	16 out	16 out	24 out	20 out	32 out	36 out	36 out	24 out	28 out	**12 out**
#Sol	192	112	144	**160**	156	156	144	148	136	132	132	144	136	**160**
Itime	192	4.37	3.94	3.39	3.38	3.36	1.23	1.52	2.20	2.48	2.62	2.56	2.73	5.43

Table 3 gives the average result of the Pentominoes instances. The BCT support, PS and MS CNF encodings significantly outperform the MDD CNF encodings. The support encoding is faster or solves more instances than the other CNF encodings on all 4 series. For the MDD CNF encodings, GMin has the best overall result, but the support encoding of BCT constraints can solve 12 more instances than GMin. In addition, the CNF encodings of BCT constraints can be competitive with the Abscon solver. The support encoding gives the best performance on 3 out of 4 Pentominoes series.



(a) Virtual best comparison. **(b)** BCT-Support vs MDD-GMin. **(c)** The number of clauses.

Figure 3 Pentominoes benchmarks.

Figure 3a shows the overall result on Pentominoes. VB-BCT is the best on more than 40% instances, and it can solve 5% more instances than VB-Abscon. Different CNF encodings of BCT constraints solve different instances, thus, VB-BCT can solve more instances than VB-Abscon. From Figure 3c, we can see that the number of clauses of GMin can be much more (up to 20 times more) than that of the support encoding. In addition, GMin is also slower than the support encoding on almost all instances (see Figure 3b).

6.3 Benchmark Series 3: Nurse scheduling

We use four different models of the nurse scheduling problem, namely, N-1, N-2, N-3 and N-4. The nurse scheduling problems come from [8, 18, 48], where nurses are assigned with a day shift, evening shift, night shift or day off for each day. The models have a cardinality [36] constraint per shift and a regular constraint per nurse. The cardinality constraints are used to guarantee that there are enough nurses to meet a demand of each shift. Each model has its own regular constraints as follows:

- In N-1, the model uses regular constraints to restrict that for each 7 days, a nurse work 1 or 2 night shifts, 1 or 2 evening shifts, 1 to 5 day shifts and 2 to 5 days off.
- In N-2, a nurse works 1 or 2 night shifts every 7 days, and 1 or 2 days off every 5 days.
- In N-3, a nurse works 1 or 2 night shifts every 9 days, and 2 or 3 days off every 7 days.
- In N-4, a nurse works 1 or 2 night shifts every 11 days, and 3 or 4 days off every 9 days.

All models restrict that a nurse can only work a second shift after 12 hours of the first. The cardinality and regular constraints are encoded as MDD constraints, and then the MDD constraints are transformed into BCT constraints. For each model, we use 50 instances from the N30 series[7] where the number of nurses of an instance is set to the maximum number of the nurse demand for a day.

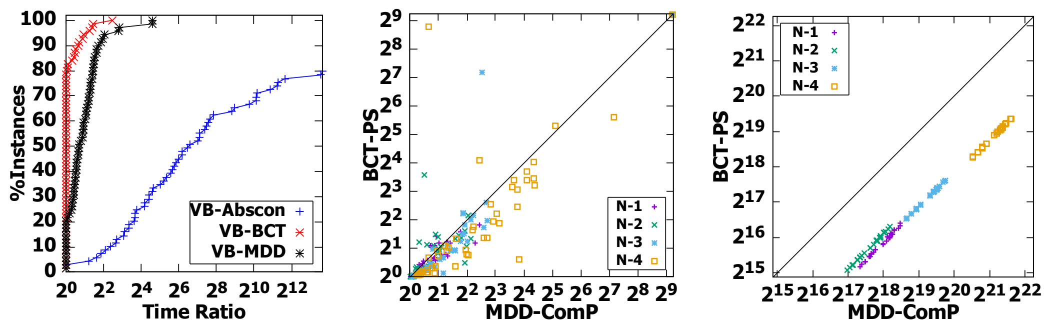
Table 4 Nurse scheduling benchmarks.

		BCT					MDD							Abscon
Inst.	#I	Log	Direct	Support	PS	MS	Min	GMin	Tes	BaP	LevP	NNFP	ComP	(Act.)
N-1	50	1 out	12.88	**0.30**	0.49	3.10	2 out	0.50	0.84	1.09	1.29	0.77	0.64	7 out
N-2	50	11.34	1.86	**0.44**	0.80	0.84	1 out	0.48	0.63	2.95	0.83	0.71	0.72	14 out
N-3	50	2 out	2 out	**1 out**	**1 out**	2 out	4 out	2 out	3 out	**1 out**	**1 out**	**1 out**	**1 out**	13 out
N-4	50	50 out	10 out	3 out	**2 out**	5 out	5 out	4 out	5 out	4 out	**2 out**	3 out	**2 out**	15 out
#Sol	200	147	188	196	**197**	193	188	194	192	195	**197**	196	**197**	151
Itime	200	2.44	1.76	0.57	0.56	0.56	0.45	0.47	0.47	0.49	0.52	0.50	0.51	0.82

Table 4 shows that CNF encodings can overall outperform the Abscon solver for the nurse scheduling instances. The Abscon solver only solves 151/200 instances within timeout but the CNF encodings can solve 197/200 instances. In our detailed results, the CNF encodings of BCT constraints are faster than the CNF encodings of MDD constraints on most instances. For example, the PS encoding is faster than each CNF encoding of MDD constraints on most nurse scheduling instances.

From Figure 4a, we see that VB-BCT is the fastest method on more than 80% instances. VB-BCT can solve 20% more instances than VB-Abscon. The CNF encodings of BCT constraints have better performance than those of MDD constraints. For example, the number of clauses of the PS encoding is around 4 times less than that of the ComP encoding

[7] `https://www.projectmanagement.ugent.be/nsp.php`



(a) Virtual best comparison. **(b)** BCT-PS vs MDD-ComP. **(c)** The number of clauses.

Figure 4 Nurse scheduling benchmarks.

(shown in Figure 4c), and the PS encoding can be faster than ComP on more than 85% instances (see Figure 4b), where ComP is the best CNF encoding of MDD constraints for the nurse scheduling instances.

6.4 Benchmark Series 4: XCSP

We use five instance series from the XCSP website[8] as they are BDD/MDD instances: bdd-15, bdd-18, mdd-p5 (MDD-half), mdd-p7 (MDD-0.7) and mdd-p9 (MDD-0.9). Some of these instances were also used in the 2019 XCSP competition. The instances bdd-15 and bdd-18 are introduced in [12], and then the instances mdd-p5, mdd-p7 and mdd-p9 are introduced in [10, 49], where mdd-pk is a MDD with sharing probability $\frac{k}{10}$ (see [10, 49] for more details).

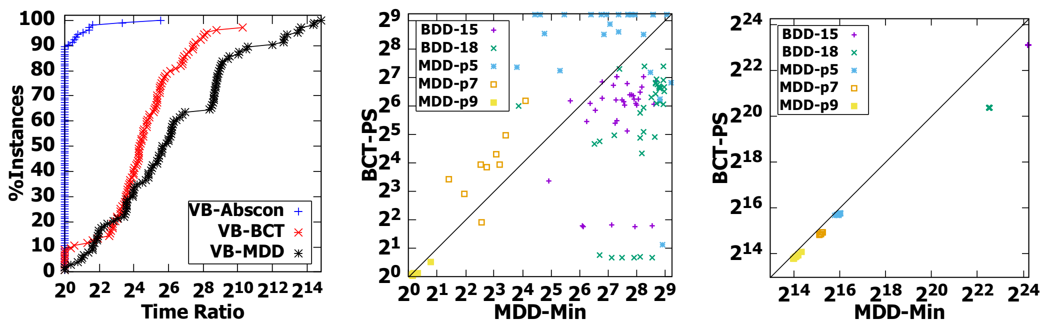
Table 5 XCSP benchmarks.

		BCT					MDD							Abscon
Inst.	#	Log	Direct	Support	PS	MS	Min	GMin	Tes	BaP	LevP	NNFP	ComP	(Act.)
bdd-15	35	35 out	222.99	106.44	67.67	33.05	192.29	152.37	16 out	22 out	29 out	28 out	33 out	**2.42**
bdd-18	35	409.58	129.94	79.64	68.77	22.86	332.11	26 out	29 out	22 out	30 out	25 out	31 out	**0.77**
mdd-p5	25	22 out	23 out	13 out	14 out	13 out	1 out	14 out	16 out	17 out	17 out	19 out	17 out	**73.06**
mdd-p7	9	95.46	68.41	24.73	20.38	23.28	6.82	21.98	44.68	50.12	55.29	39.54	39.58	**1.82**
mdd-p9	10	2.16	0.59	0.19	0.11	0.35	0.18	0.27	0.73	1.08	0.76	0.48	0.46	**0.06**
#Sol	114	57	91	101	100	101	113	74	53	53	38	42	33	**114**
Itime	114	3.96	2.69	1.72	1.64	1.62	1.79	2.21	2.61	2.89	3.03	3.30	3.43	2.09

Table 5 shows that the Abscon solver using the Activity heuristic is the fastest overall for these instances. The Abscon solver can solve all instances while the CNF encodings are time-out on some instances. The CNF encodings of BCT and MDD constraints perform better on different instances. On the bdd-15, bdd-18 and mdd-p9 series, the PS encoding is faster than the CNF encodings of MDD constraints while Min is the best CNF encoding on the mdd-p5 and mdd-p7 series.

Figure 5a shows that VB-Abscon and VB-BCT is the best method on around 90% and 10% instances, respectively. In addition, VB-BCT can be faster than VB-MDD on more than 80% instances. Figure 5b shows the differences between instances, PS is faster than Min on almost all instances in the bdd-15 and bdd-18 series but the opposite happens on the

[8] `http://xcsp.org`



(a) Virtual best comparison. **(b)** BCT-MS vs MDD-Min. **(c)** The number of clauses.

Figure 5 XCSP benchmarks.

mdd-p5 and mdd-p7 instances, where Min is the best CNF encoding of MDD constraints for these instances. From Figure 5c, we can see that the number of clauses of the PS encoding can be 2-5 times less than that of Min on the bdd-15 and bdd-18 series.

We summarize experiments on all four benchmark series. While there is some initialization and encoding time for all methods, this is overall less significant than the solving time (there are many timeouts for some methods). The initialization time becomes significant when the encoding becomes large, e.g. in the NFA instances, the encoding cost becomes significant in the MDD CNF encodings with some being memory-out. Overall across all four problem series, BCT CNF encodings generally outperform MDD CNF encodings. As with the MDD CNF encoding experiments in [1] where they found performance was mixed between CNF encodings and their propagator comparison, we also find that for some problems the BCT CNF encoding is the best while for other problems the BCT propagator in Abscon is the best. Still BCT CNF encoding is overall competitive or best for many instances and increases the flexibility and choices in solving of BCT (and NFA/MDD) constraints.

7 Conclusion

Binary Constraint Tree (BCT) is more compact than Ordered Multi-valued Decision Diagram (MDD). We show that BCT can be exponentially smaller than MDD when representing NFA constraints. We investigate CNF encodings on BCT constraints which allow solving of BCT constraints with SAT solvers. At the same time, we show this can improve CNF encodings of MDD constraints. We tailor three well-known CNF encodings of binary constraints, i.e. the log encoding, direct encoding and support encoding, to encode BCT constraints. Then we propose two new CNF encodings, partial support encoding and minimal support encoding, which give smaller CNF encodings of BCT constraints. We study and compare the strength of unit propagation on these five CNF encodings of BCT constraints. Our experimental results study our CNF encodings of BCT constraints and also compare with seven existing CNF encodings of MDD constraints on a range of existing benchmarks. Experimental results show that the CNF encodings of BCT constraints can outperform those of MDD constraints. Our results show that solving of BCT constraints as well as NFA/MDD constraints is promising on SAT solvers.

References

1 Ignasi Abío, Graeme Gange, Valentin Mayer-Eichberger, and Peter J Stuckey. On CNF encodings of decision diagrams. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 1–17. Springer, 2016.

2 Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, and Valentin Mayer-Eichberger. A new look at BDDs for pseudo-boolean constraints. *Journal of Artificial Intelligence Research*, 45:443–480, 2012.

3 Ignasi Abío and Peter J Stuckey. Encoding linear constraints into SAT. In *International Conference on Principles and Practice of Constraint Programming*, pages 75–91. Springer, 2014.

4 Jérôme Amilhastre, Hélene Fargier, Alexandre Niveau, and Cédric Pralet. Compiling CSPs: A complexity map of (non-deterministic) multivalued decision diagrams. *International Journal on Artificial Intelligence Tools*, 23(04):1460015, 2014.

5 Miquel Bofill, Jordi Coll, Josep Suy, and Mateu Villaret. Compact MDDs for pseudo-boolean constraints with at-most-one relations in resource-constrained scheduling problems. In *International Joint Conference on Artificial Intelligence*, pages 555–562, 2017.

6 Lucas Bordeaux and Joao Marques-Silva. Knowledge compilation with empowerment. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 612–624. Springer, 2012.

7 Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *European Conference on Artificial Intelligence*, 2004.

8 Sebastian Brand, Nina Narodytska, Claude-Guy Quimper, Peter Stuckey, and Toby Walsh. Encodings of the Sequence constraint. In *International conference on principles and practice of constraint programming*, pages 210–224. Springer, 2007.

9 Kenil C.K. Cheng, Wei Xia, and Roland H.C. Yap. Space-time tradeoffs for the regular constraint. In *International Conference on Principles and Practice of Constraint Programming*, pages 223–237. Springer, 2012.

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

11 Kenil C.K. Cheng and Roland H.C. Yap. Applying ad-hoc global constraints with the case constraint to still-life. *Constraints*, 11(2-3):91–114, 2006.

12 Kenil C.K. Cheng and Roland H.C. Yap. Maintaining generalized arc consistency on ad-hoc n-ary boolean constraints. In *17th European Conference on Artificial Intelligence*, pages 78–82, 2006.

13 Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38(3):353–366, 1989.

14 Alvaro Del Val. Tractable databases: How to make propositional unit resolution complete through compilation. In *International Conference on Principles of Knowledge Representation and Reasoning*, pages 551–561. Elsevier, 1994.

15 Jordan Demeulenaere, Renaud Hartert, Christophe Lecoutre, Guillaume Perez, Laurent Perron, Jean-Charles Régin, and Pierre Schaus. Compact-Table: efficiently filtering table constraints with reversible sparse bit-sets. In *International Conference on Principles and Practice of Constraint Programming*, pages 207–223, 2016.

16 Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

17 Armin Biere Katalin Fazekas Mathias Fleury and Maximilian Heisinger. CaDiCaL, KISSAT, PARACOOBA, PLINGELING and TREENGELING entering the SAT competition 2020. *SAT COMPETITION*, 2020:50, 2020.

18 Graeme Gange, Peter J Stuckey, and Radoslaw Szymanek. MDD propagators with explanation. *Constraints*, 16(4):407, 2011.

19 Ian P Gent. Arc consistency in SAT. In *European Conference on Artificial Intelligence*, pages 121–125, 2002.

20 Ian P Gent and Peter Nightingale. A new encoding of AllDifferent into SAT. In *International Workshop on Modelling and Reformulating Constraint Satisfaction*, pages 95–110, 2004.

21 Rebecca Gentzel, Laurent Michel, and Willem Jan van Hoeve. Haddock: A language and architecture for decision diagram compilation. In *nternational Conference on Principles and Practice of Constraint Programming*, pages 531–547. Springer, 2020.

22 Willem-Jan van Hoeve, Gilles Pesant, Louis-Martin Rousseau, and Ashish Sabharwal. Revisiting the sequence constraint. In *International conference on principles and practice of constraint programming*, pages 620–634. Springer, 2006.

23 Kazuo Iwama and Shuichi Miyazaki. SAT-variable complexity of hard combinatorial problems. In *IFIP World Computer Congress*, 1994.

24 Simon Kasif. On the parallel complexity of discrete relaxation in constraint satisfaction networks. *Artificial Intelligence*, 45(3):275–286, 1990.

25 Petr Kucera and Petr Savický. Propagation complete encodings of smooth DNNF theories. *CoRR*, abs/1909.06673, 2019. `arXiv:1909.06673`.

26 Petr Kučera and Petr Savickỳ. Bounds on the size of PC and URC formulas. *Journal of Artificial Intelligence Research*, 69:1395–1420, 2020.

27 Mikael Zayenz Lagerkvist. *Techniques for efficient constraint propagation*. PhD thesis, KTH, 2008.

28 Christophe Lecoutre. STR2: optimized simple tabular reduction for table constraints. *Constraints*, 16(4):341–371, 2011.

29 Sylvain Merchez, Christophe Lecoutre, and Frédéric Boussemart. Abscon: A prototype to solve CSPs with abstraction. In *International Conference on Principles and Practice of Constraint Programming*, pages 730–744. Springer, 2001.

30 Laurent Michel and Pascal Van Hentenryck. Activity-based search for black-box constraint programming solvers. In *International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming*, 2012.

31 Guillaume Perez and Jean-Charles Régin. Improving GAC-4 for table and MDD constraints. In *International Conference on Principles and Practice of Constraint Programming*, pages 606–621. Springer, 2014.

32 Gilles Pesant. A regular language membership constraint for finite sequences of variables. In *International conference on principles and practice of constraint programming*, pages 482–495. Springer, 2004.

33 Claude-Guy Quimper and Toby Walsh. Global grammar constraints. In *International conference on principles and practice of constraint programming*, pages 751–755, 2006.

34 Philippe Refalo. Impact-based search strategies for constraint programming. In *International Conference on Principles and Practice of Constraint Programming*, 2004.

35 Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In *National Conference on Artificial Intelligence*, 1994.

36 Jean-Charles Régin. Generalized arc consistency for global cardinality constraint. *National Conference on Artificial Intelligence*, pages 209–215, 1996.

37 Francesca Rossi, Charles J. Petrie, and Vasant Dhar. On the equivalence of constraint satisfaction problems. In *European Conference on Artificial Intelligence*, pages 550–556, 1990.

38 Barbara M Smith and Stuart A Grant. Trying harder to fail first. In *European Conference on Artificial Intelligence*, 1998.

39 Arvind Srinivasan, Timothy Ham, Sharad Malik, and Robert K Brayton. Algorithms for discrete function manipulation. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 92–95, 1990.

40 Kostas Stergiou and Toby Walsh. Encodings of non-binary constraint satisfaction problems. In *AAAI Conference on Artificial Intelligence*, pages 163–168, 1999.

41 Allen Van Gelder. Another look at graph coloring via propositional satisfiability. *Discrete Applied Mathematics*, 156(2):230–243, 2008.

42 Hélene Verhaeghe, Christophe Lecoutre, and Pierre Schaus. Compact-MDD: Efficiently filtering (s)MDD constraints with reversible sparse bit-sets. In *International Joint Conference on Artificial Intelligence*, pages 1383–1389, 2018.

43 Hélène Verhaeghe, Christophe Lecoutre, and Pierre Schaus. Extending Compact-Diagram to basic smart multi-valued variable diagrams. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 581–598. Springer, 2019.

44 Toby Walsh. SAT v CSP. In *International Conference on Principles and Practice of Constraint Programming*, pages 441–456, 2000.

45 Ruiwei Wang, Wei Xia, Roland H. C. Yap, and Zhanshan Li. Optimizing simple tabular reduction with a bitwise representation. In *International Joint Conference on Artificial Intelligence*, pages 787–795, 2016.

46 Ruiwei Wang and Roland H. C. Yap. Arc consistency revisited. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 599–615, 2019.

47 Ruiwei Wang and Roland H. C. Yap. Bipartite encoding: A new binary encoding for solving non-binary csps. In *International Joint Conference on Artificial Intelligence*, pages 1184–1191, 2020.

48 Ruiwei Wang and Roland H. C. Yap. Encoding multi-valued decision diagram constraints as binary constraint trees. In *AAAI Conference on Artificial Intelligence*, 2022.

49 Wei Xia and Roland H. C. Yap. Optimizing STR algorithms with tuple compression. In *International Conference on Principles and Practice of Constraint Programming*, pages 724–732, 2013.

50 Roland H. C. Yap, Wei Xia, and Ruiwei Wang. Generalized arc consistency algorithms for table constraints: A summary of algorithmic ideas. In *AAAI Conference on Artificial Intelligence*, pages 13590–13597, 2020.

A Appendix: Proofs

▶ **Theorem 5.** *BCT can be exponentially smaller than MDD on representing NFA constraints.*

Proof. The DTBE of a r arity NFA constraints has $3r+1$ variables and $3r$ binary constraints. The hidden variable domains include at most $max(sn, tn)$ values, where sn and tn are the number of states and transitions in the NFA. In addition, each binary constraint relation has tn tuples. So the size of the DTBE is polynomial in that of the NFA constraint.

The size of the negation of the MDDs (Ordered Multi-valued Decision Diagrams) representing alldifferent constraints is exponential in r [4]. Therefore, the number of nodes in a MDD representing the family of NFA constraints given in Example 2 is exponential in r, where r is constraint arity and the NFA has $r+1$ states and $4r+r^2$ transitions. So BCT can be exponentially smaller than MDD on representing NFA constraints. ◀

▶ **Lemma 10.** *Given a BCT $P = (X, C)$ and a tree order O over X, if a literal $(v_a^{O_1}, true)$ is included in $UP(F)$ and all variable domains in $UP(F)$ are not empty, there is $\tau \in sol(P)$ such that $(O_1, a) \in \tau$ and $(v_b^x, true)$ is included in F for all $(x, b) \in \tau$, where $F = (A, C^A)|_{\tau'}$ and $B^x \subseteq A$ for all $x \in X$ and $T^O \subseteq C^A$ and τ' is a tuple over a subset of A.*

Proof. For any $c \in C$ where $scp(c) = \{O_i, O_j\}$ and $i < j$, if a literal $(v_a^{O_i}, true)$ is included in $UP(F)$, there must be a literal $(v_b^{O_j}, true)$ in $UP(F)$ such that $\{(O_i, a), (O_j, b)\} \in rel(c)$, otherwise unit propagation with the clause $cl(O_i, a, c)$ can remove $(v_a^{O_i}, true)$ from $UP(F)$. Note that the clause $cl(O_i, a, c)$ encodes the implication: if $v_b^{O_j} = false$ for all tuple $\{(O_i, a), (O_j, b)\} \in rel(c)$, then $v_a^{O_i} = false$.

So we can construct a series of tuples $\{\tau_1, ..., \tau_n\}$ such that $n = |X|$ and $\tau_1 = \{(O_1, b_1)\}$ and $b_1 = a$ and for $j > 1$, $\tau_j = \tau_{j-1} \cup \{(O_j, b_j)\}$ and $(v_{b_j}^{O_j}, True)$ is included in $UP(F)$ and $\{(b_i, O_i), (b_j, O_j)\} \in rel(c)$, where c is the only constraint in C such that $scp(c) = \{O_i, O_j\}$ and $i < j$. The tuple τ_n is a solution of P and $(O_1, a) \in \tau_n$. ◀

▶ **Proposition 11.** *The support encoding $F = (A \cup B, T \cup E)$ of BCT constraints (V, P) implements propagation completeness, where P is a BCT (X, C) and $A = \bigcup_{x \in X} A^x$ and $B = \bigcup_{x \in X} B^x$ and $T = \{cl(x, c) | c \in C, x \in scp(c)\}$ and $E = \bigcup_{x \in X} EO(x)$.*

Proof. Assume $F^\tau = UP(F|_\tau)$ and all variable domains in F^τ are not empty where τ is a tuple over a subset of $A \cup B$. The ladder encoding implements completeness propagation, therefore, for any $x \in X$, if F^τ includes a literal l of a variable in $A^x \cup B^x$, then there is a tuple $t(x, a)$ such that F^τ includes $t(x, a)$ and $t(x, a) \in sol(A^x \cup B^x, EO(x))$ and $l \in t(x, a)$ and $(v_a^x, true) \in t(x, a)$. We can set x as root and construct a tree order O over X such that $O_1 = x$, thus, there is a solution of P including (x, a) which corresponds to a solution of $F|_\tau$ including l (based on Lemma 10). So F implements propagation completeness. ◀

▶ **Proposition 13.** *The partial support encoding $F = (A^V \cup B^V \cup B^H, T \cup E^V)$ of a BCT constraint (V, P) implements unit refutation completeness where $P = (X, C)$.*

Proof. Let $x \in X$ and $F^\tau = UP(F|_\tau)$ where τ is a tuple over a subset of $A^V \cup B^V \cup B^H$. If all variable domains in F^τ are not empty, there is $a \in \mathcal{D}(x)$ such that $(v_a^x, true)$ is in F^τ, otherwise unit propagation with $EO(x)$ can remove all values of the variables in $A^x \cup B^x$, since the ladder encoding implements propagation completeness and every tuple in $sol(A^x \cup B^x, EO(x))$ includes at least a value *true* of a variable in B^x. Therefore, there is a solution of P including (x, a) which corresponds to a solution of $F|_\tau$ including $(v_a^x, true)$ by setting x as root (based on Lemma 10). So F implements unit refutation completeness. ◀

▶ **Proposition 14.** *The partial support encoding $F = (A^V \cup B^V \cup B^H, T \cup E^V)$ of a BCT constraint (V, P) implements domain consistency where $P = (X, C)$.*

Proof. Let $x \in V$ and $F^\tau = UP(F|_\tau)$ where τ is a tuple over a subset of $A^V \cup B^V$ and all variable domains in F^τ are not empty. If a literal l of a variable in $A^x \cup B^x$ is included in F^τ, there is a value $a \in \mathcal{D}(x)$ such that $l \in t(x, a)$ and $(v_a^x, true) \in t(x, a)$ and $(v_a^x, true)$ is included in F^τ (since ladder encoding implements propagation completeness). So there is a solution of P including (x, a) which corresponds to a solution of $F|_\tau$ including l (Lemma 10 by setting x as root). Hence, the partial support encoding implements domain consistency. ◀

▶ **Proposition 16.** *The minimal support encoding F of a BCT constraint (V, P) with respect to a tree order O implements unit refutation completeness where $x = O_1$ and $P = (X, C)$.*

Proof. Let $F^\tau = UP(F|_\tau)$ where τ is a tuple over a subset of $A^V \cup B^V \cup B^H$. If all variable domains in F^τ are not empty, there is $a \in \mathcal{D}(x)$ such that $(v_a^x, true)$ is in F^τ, otherwise unit propagation with $EO(x)$ can remove all values of the variables in $A^x \cup B^x$, since ladder encoding implements propagation completeness and every tuple in $sol(A^x \cup B^x, EO(x))$ includes at least a value *true*. Therefore, there is a solution of P including (x, a) which corresponds to a solution of $F|_\tau$ including $(v_a^x, true)$ based on Lemma 10 (where x is set as root). So F implements unit refutation completeness. ◀