Binary Constraint Trees and Structured Decomposability

Petr Kučera

Department of Theoretical Computer Science and Mathematical Logic, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Abstract

A binary constraint tree (BCT, Wang and Yap 2022) is a normalized binary CSP whose constraint graph is a tree. A BCT constraint is a constraint represented with a BCT where some of the variables may be hidden (i.e. existentially quantified and used only for internal representation). Structured decomposable negation normal forms (SDNNF) were introduced by Pipatsrisawat and Darwiche (2008) as a restriction of decomposable negation normal forms (DNNF). Both DNNFs and SDNNFs were studied in the area of knowledge compilation. In this paper we show that the BCT constraints are polynomially equivalent to SDNNFs. In particular, a BCT constraint can be represented with an SDNNF of polynomial size and, on the other hand, a constraint that can be represented with an SDNNF, can be represented as a BCT constraint of polynomial size. This generalizes the result of Wang and Yap (2022) that shows that a multivalued decision diagram (MDD) can be represented with a BCT. Moreover, our result provides a full characterization of binary constraint trees using a language that is well studied in the area of knowledge compilation. It was shown by Wang and Yap (2023) that a CSP on n variables of domain sizes bounded by d that has treewidth k can be encoded as a BCT on $O(n)$ variables with domain sizes $O(d^{k+1})$. We provide an alternative reduction for the case of binary CSPs. This allows us to compile any binary CSP to an SDNNF of size that is parameterized by d and k.

2012 ACM Subject Classification Theory of computation → Automated reasoning; Theory of computation → Constraint and logic programming

Keywords and phrases Binary CSP, Binary Constraint Tree, Structured Decomposability, Strucured DNNF, Polynomial Equivalence

Digital Object Identifier 10.4230/LIPIcs.CP.2023.22

1 Introduction

Constraint satisfaction problems (CSPs) offer an expressive and natural way of formulating problems. A CSP is a problem of checking satisfiablity of a conjunction of constraints on variables with finite domains. Constraints can be represented in various ways, which include tables (see e.g. [2, 17, 18]) or multivalued decision diagrams (MDD, see e.g. [1, 5, 6]).

The representation using binary constraint trees was introduced in [27]. A BCT constraint is a constraint c defined on a set of variables $\mathbf{x}$ that is represented with a normalized binary CSP P whose constraint graph is a tree. The CSP P itself is defined on a set of variables $\mathbf{z}$ which may include some *hidden* variables in addition to all the *original* variables from $\mathbf{x}$. BCTs have a nice property that an arc consistency propagator can be used to check their consistency [12]. Any CSP can be turned into a binary one with an encoding such as dual encoding [11], hidden variable encoding [22], double encoding [24], or bipartite encoding [25].

Decomposable negation normal forms (DNNFs) were introduced in [7] as a tractable language for knowledge representation. Structured DNNFs (SDNNF) were introduced in [20]. The definition of SDNNFs is based on the notion of a *v-tree* which is a rooted binary tree whose leaves are in one-to-one correspondence with the constraint variables (both original and hidden). The conjunction gates in an SDNNF are then required to respect a particular v-tree (see definitions 5 and 6 in Section 2.3 for more details). The structural requirements imposed

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).
Editor: Roland H. C. Yap; Article No. 22; pp. 22:1–22:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

on SDNNFs allow for instance a polynomial time construction of an SDNNF representing the conjunction of two SDNNFs that respect the same v-tree – something that is not possible for DNNFs without any structural requirements. Although DNNFs were introduced as a representation of functions on boolean variables, they were also considered as a representation of constraints on variables with finite domains [1, 14].

Both BCT constraints and SDNNFs have a structure based on a tree. We use this similarity to show the main result of this paper: BCT constraints and SDNNF constraints are polynomially equivalent. In particular, we show a polynomial time transformation of a BCT constraint into an SDNNF and also a polynomial time transformation of an SDNNF into a BCT constraint. The polynomial equivalence of BCTs with SDNNFs offers a characterization of BCTs by a language of SDNNFs which has been extensively studied in the area of knowledge compilation [3, 20, 21, 23]. Our result also generalizes the previous construction of a BCT constraint for an MDD described in [27]. It was shown in [26] that BCTs are strictly more succinct than MDDs. This also follows from a combination of our result with the fact that SDNNFs are strictly more succinct than MDDs by [20].

Recently, [28] studied BCTs from the perspective of knowledge compilation together with several other languages that are being used to represent ad-hoc constraints. The authors studied BCTs with respect to the queries and transformations considered in the knowledge compilation map [10] and showed that BCTs allow answering consistency, clausal entailment and model enumeration queries in polynomial time which is (unsurprisingly) the same as in the case of structured DNNFs [20]. The authors of [28] also studied BCTs with respect to transformations. If the input BCTs or SDNNFs are required to have the same tree structure, then they allow polynomial-time bounded conjunction, unbounded disjunction, forgetting any number of variables, and conditioning [20, 28]. Interestingly, [20] only considers the case of combining SDNNFs that respect the same v-tree while [28] also considers the case of combining BCTs that are not required to have the same tree structure. In this case BCTs do not allow polynomial time bounded conjunction, they do not allow an unbounded disjunction, and the case of bounded disjunction is unresolved in [28]. We believe that our result might help to resolve the case of bounded disjunction for BCTs, because it might be easier to reason about a disjunction of two SDNNFs than BCTs. It is also worth mentioning that according to [20], AOMDDs introduced in [19] are strictly less succinct than SDNNFs and thus also strictly less succinct than BCTs. This already answers one of the questions posed in [28].

Our transformation of a BCT constraint into an SDNNF leads to a smooth SDNNF. It is thus possible to use a domain consistency propagator for smooth DNNFs described in [14] as a domain consistency propagator for BCT constraints. An encoding of BCT constraints with propagation complete CNF formulas was described in [26]. Various CNF encodings of DNNF theories were considered in [1] and a propagation complete encoding of smooth DNNFs was introduced in [16]. Our result thus offers an alternative way of reducing BCT constraints to a CNF encoding.

If CNF φ has treewidth k, then it can be compiled to an SDNNF of size that is parameterized by k by the construction described in [21]. In particular, if φ has n variables and m clauses, then an equivalent SDNNF can be constructed in time $O(nm2^k)$. We can obtain a similar result also for binary CSPs, but we have to take into account also the domain sizes. It was shown in [28] that if P is a CSP on n variables of domain size d that has treewidth k, then it can be encoded as a BCT with $O(n)$ variables with domain size d^{k+1}. The construction in [28] uses the encoding described in [11]. In addition, if P is a binary CSP, its consistency can be checked in time $O(nd^{k+1})$ by [13]. To have a complete compilation procedure of a binary CSP into an SDNNF, we provide a direct reduction of a binary CSP to a BCT parameterized by the treewidth and the domain size. The bound we obtain is the same as in [28], so our result is not really new in this sense, but we obtain a slightly better bound on the size of an SDNNF constructed for the given binary CSP than if we would simply combine the bound of [28] with our construction of an SDNNF.

The paper is organized as follows. We introduce the necessary notation in Section 2, including the definitions of BCTs and structured DNNFs. In Section 3, we show that a BCT constraint can be represented with an SDNNF. The transformation of an SDNNF to a BCT is described in Section 4. A tranformation of a binary CSP with bounded treewidth into an SDNNF is described in Section 5. Section 6 closes the paper with a few concluding remarks.

2 Definitions

In this section, we shall recall the necessary notation and notions used in the paper.

2.1 BCT Constraint

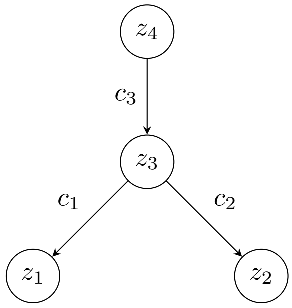
We use a notation adapted from [27] where binary constraint trees were introduced.

A CSP P is a pair $(\mathbf{x}, C)$ where $\mathbf{x}$ is a set of variables and C is a set of constraints. Each variable x has a finite domain denoted $\mathit{dom}(x)$. A *literal* on a variable x is a variable value assignment (x, a). A *tuple* over a set of variables $\{x_{i_1}, x_{i_2}, \ldots, x_{i_r}\}$ is a *set of literals* $\{(x_{i_1}, a_1), (x_{i_2}, a_2), \ldots, (x_{i_r}, a_r)\}$. Each constraint c_j has a constraint scope $\mathit{scp}(c_j) \subseteq \mathbf{x}$ and a relation $\mathit{rel}(c_j)$ defined by a set of tuples over $\mathit{scp}(c_j)$. A constraint c is a *binary constraint* if $|\mathit{scp}(c)| = 2$ and it is a *unary constraint* if $|\mathit{scp}(c)| = 1$. A CSP P is called a *binary CSP* if it consists of binary and unary constraints. A binary CSP is *normalized* if its constraints have pairwise different scopes. Given any set of variables $\mathbf{z}$ and literals τ, we use $\tau[\mathbf{z}] = \{(x, a) \in \tau \mid x \in \mathbf{z}\}$ to denote a subset of τ, while $T[\mathbf{z}] = \{\tau[\mathbf{z}] \mid \tau \in T\}$ is the *projection* of a set of tuples T on $\mathbf{z}$. A tuple τ over $\mathbf{x}$ is a solution of P if $\tau[\mathit{scp}(c)] \in \mathit{rel}(c)$ for all constraints $c \in C$ and $a \in \mathit{dom}(x)$ for all $(x, a) \in \tau$. We use $\mathit{sol}(\mathbf{x}, C)$ (or $\mathit{sol}(P)$) to denote the set of all solutions of P. We also say that P is *satisfied* by its solution and that a solution of P *satisfies* all constraints in C. A *support of a value* $a \in \mathit{dom}(x)$ in a constraint c is a tuple $\tau \in \mathit{rel}(c)$ such that $(x, a) \in \tau$ and $b \in \mathit{dom}(y)$ for all $(y, b) \in \tau$.

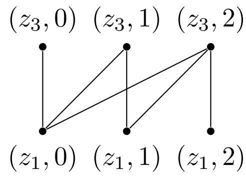
▶ **Definition 1** ([27]). *A* Binary Constraint Tree *(*BCT*) is a normalized binary CSP whose constraint graph is a tree. A* BCT constraint *c is a pair $(\mathbf{x}, P)$ such that $P = (\mathbf{z}, C)$ is a BCT, $\mathit{scp}(c) = \mathbf{x} \subseteq \mathbf{z}$, and $\mathit{rel}(c) = \mathit{sol}(\mathbf{z}, C)[\mathbf{x}]$. A* tree binary encoding *(*TBE*) of a constraint c^* is a BCT $P = (\mathbf{z}, C)$ such that the BCT constraint $(\mathit{scp}(c^*), P)$ has the same constraint relation as c^* where the variables in $\mathit{scp}(c^*)$ and $\mathbf{z} \setminus \mathit{scp}(c^*)$ are called the* original *and* hidden *variables, respectively.*

▶ **Example 2.** Let us consider a BCT constraint $c^* = (\mathbf{x}, P)$ on three variables $\mathbf{x} = \{x_1, x_2, x_3\}$ where $P = (\mathbf{z}, C)$ is a BCT described as follows. We have one hidden variable y in P, i.e. $\mathbf{z} = \{x_1, x_2, x_3, y\}$, and three constraints $C = \{c_1, c_2, c_3\}$ with $\mathit{scp}(c_i) = \{x_i, y\}$, $i = 1, 2, 3$. The domain of all variables (original and hidden) is $\{1, 2, 3\}$. For $i = 1, 2, 3$, we set $\mathit{rel}(c_i) = \{((x_i, a), (y, b)) \mid a \neq b\}$. That is, c_i enforces that y has a different value from x_i in any solution to c^*. Altogether, c^* is equal to the negation of the *alldifferent* constraint over the variables x_1, x_2, x_3.

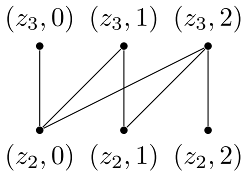
A general construction of a BCT representing the negation of the *alldifferent* constraint over variables $x_1, \ldots, x_r$ with domains $D = \{1, \ldots, r\}$ was described in [27] where the authors also noted that the size of the MDD representing the constraint is exponential in r.



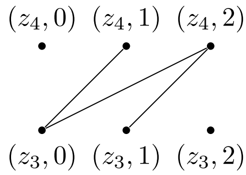
(a) The binary constraint tree of P.



(b) Relation of constraint c_1 ($z_1 \leq z_3$).



(c) Relation of constraint c_2 ($z_2 \leq z_3$).



(d) Relation of constraint c_3 ($z_3 < z_4$).

Figure 1 A binary constraint tree $P = (\mathbf{z}, C)$ from Example 3.

To demonstrate the techniques described in the paper, we shall consider a simple constraint that is a bit less symmetrical than the negation of the *alldifferent* constraint on three variables.

▶ **Example 3.** Figure 1 shows a BCT $P = (\mathbf{z}, C)$ that is defined on variables $\mathbf{z} = (z_1, z_2, z_3, z_4)$ with domains $dom(z_i) = \{0, 1, 2\}$ for all $i = 1, \ldots, 4$. C consists of three constraints. Constraints c_1 and c_2 represent inequalities $z_1 \leq z_3$ and $z_2 \leq z_3$ respectively and its relation is shown in figures 1b and 1c. Constraint c_3 represents inequality $z_3 < z_4$ and its relation is shown in Figure 1d. Note that literals $(z_4, 0)$ and $(z_3, 2)$ do not have support in c_3.

Let us now consider a constraint c^* with scope $scp(c^*) = \{x_1, x_2, x_3\}$ where $dom(x_i) = \{0, 1, 2\}$ for $i = 1, 2, 3$ and the set of tuples $rel(c^*)$ represents inequality $\max(x_1, x_2) < x_3$. If we identify variables z_1, z_2, and z_4 with x_1, x_2, and x_3 respectively, then P is a tree binary encoding of c^* in which z_1, z_2, and z_4 are original variables and z_3 is a hidden variable.

Note that the hidden variable z_3 is not actually needed for the constraint representation. We keep it to demonstrate how a hidden variable can be later forgotten in an SDNNF. We may also observe that literals $(z_4, 0)$ and $(z_3, 2)$ do not have a support in constraint c_3. We shall see later how this situation is dealt with during the construction of an SDNNF representing c^*.

2.2 DNNF

The notion of a DNNF was introduced in [7] as a restriction of NNF. We consider a multivalued variant that was used for instance in [14, 16]. This form is suitable for using DNNFs to represent constraints.

Consider a set of variables $\mathbf{x} = \{x_1, \ldots, x_n\}$ with a finite domain $dom(x_i)$ for each $x_i \in \mathbf{x}$. A sentence in *negation normal form* (NNF) D is a rooted DAG with vertices V, root $\rho \in V$, the set of edges E, and the set of leaves $L \subseteq V$. The inner vertices (also called *gates*) are labeled with logical connectives $\wedge$ or $\vee$. Each edge (v, u) in D connects an inner vertex v labeled $\wedge$ or $\vee$ with one of its inputs u. The edge is directed from v to u, so the inputs of a vertex are its successors (or child vertices). The leaves are labeled with literals of variables $\mathbf{x}$, i.e. each leaf is labeled with a literal (x_i, a) on a variable $x_i \in \mathbf{x}$ and a value $a \in dom(x_i)$. We assume that each literal (x_i, a) is used as a label of at most one leaf. Some of the literals may be missing in D, however, we assume that for each $i = 1, \dots, n$ at least one leaf of D is labeled with a literal on variable x_i. For technical reasons, we also allow to label leaves with constants 0 or 1. If a NNF is nonempty, constants can always be propagated and these constants are thus needed only on a NNF without any variables.

If all the constraint variables $x_i \in \mathbf{x}$ are boolean (i.e. $dom(x_i) = \{0, 1\}$) and we identify literal $(x_i, 1)$ with the propositional literal x_i and literal $(x_i, 0)$ with the propositional literal $\neg x_i$, then we obtain the usual definition of a NNF for representing a boolean function.

Assume that c is a constraint with the scope $scp(c) = \mathbf{x}$ and that D is a NNF defined on the variables $\mathbf{x}$. We say that D represents constraint c if for every tuple τ over variables $\mathbf{x}$ we have that $\tau \in rel(c)$ if and only if D evaluates to true on the tuple τ. Evaluating D on τ is done in a straightforward manner, we simply set the leaves $(x_i, a) \in \tau$ to true and the remaining leaves to false, then we use the usual semantic of the circuit D to get the value on this assignment.

Following [14], we define the decomposability and smoothness properties with respect to constraint variables $x_1, \dots, x_n$. For a vertex $v \in V$, let us denote $var(v) \subseteq \mathbf{x}$ the set of variables in the subcircuit of D rooted at v. More precisely, a variable $x_i \in \mathbf{x}$ belongs to $var(v)$ if and only if there is a directed path from v to a leaf labeled with a literal (x_i, a) for a value $a \in dom(x_i)$. We have by assumption that $var(\rho) = \mathbf{x}$.

▶ **Definition 4.** *We define the following structural restrictions of NNFs.*

- *We say that NNF D is* decomposable *(DNNF), if for every vertex $v = v_1 \wedge \cdots \wedge v_k$ the sets of variables $var(v_1), \dots, var(v_k)$ are pairwise disjoint.*
- *We say that DNNF D is* smooth *if for every vertex $v = v_1 \vee \cdots \vee v_k$ we have $var(v) = var(v_1) = \cdots = var(v_k)$.*

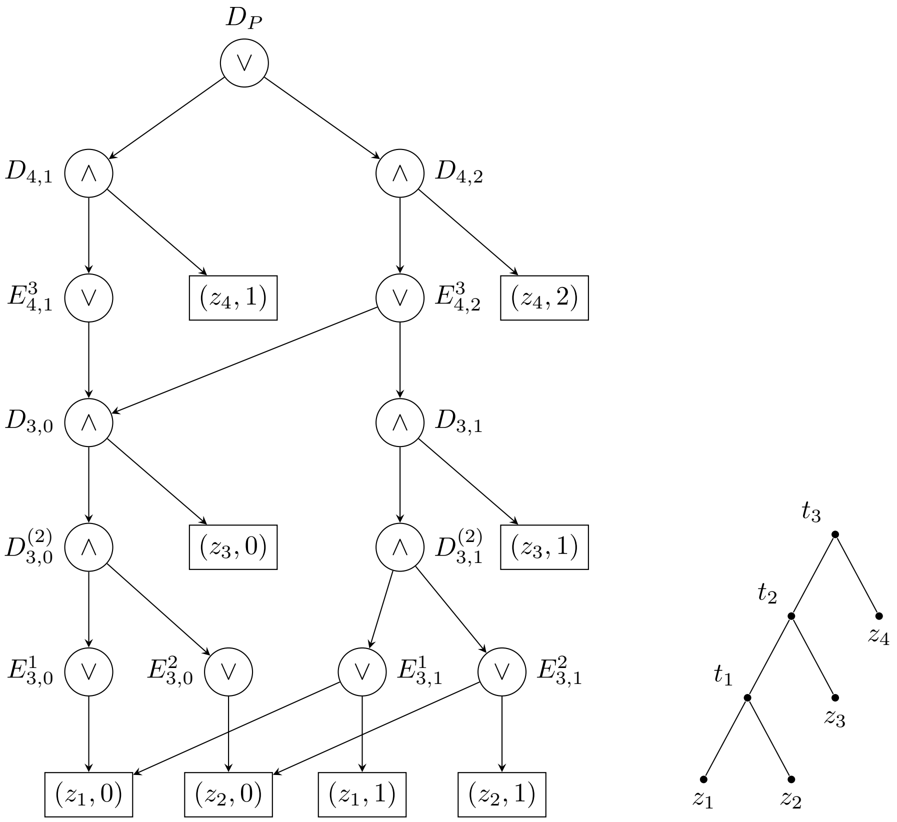
Assume that D is a DNNF representing constraint c with scope $scp(c) = \mathbf{z}$. Let $\mathbf{x} \subseteq \mathbf{z}$ and $\mathbf{y} = \mathbf{z} \setminus \mathbf{x}$. By *forgetting* variables $\mathbf{y}$ in D we mean the construction of a DNNF D' that represents the constraint c' which is a projection of c on variables $\mathbf{x}$. In particular, $scp(c') = \mathbf{x}$ and $rel(c') = rel(c)[\mathbf{x}]$. Forgetting can be done efficiently on a DNNF, we simply replace every literal (y, a) with constant 1 for all $y \in \mathbf{y}$ and $a \in dom(y)$ [8].

2.3 Structured DNNF

Structured DNNFs were introduced in [20]. Structured decomposability is based on the notion of a v-tree defined as follows.

▶ **Definition 5** ([20])**.** *A* v-tree *for a set of variables $\mathbf{x}$ is a full, rooted binary tree whose leaves are in one-to-one correspondence with the variables in $\mathbf{x}$.*

Given a node t of a v-tree T, we denote $var(t)$ the set of variables associated with the leaves in the subtree of T rooted at t. We also denote $var(T) = var(\sigma)$ where σ is the root of T. For a non-leaf node t, we use t^l (t^r) to denote the left (right) child node of t. For the rest of this paper, we will assume that each conjunction in a DNNF has exactly two non-constant inputs, while a disjunction can have any number of inputs. This is a technical assumption used in [20] mainly to simplify the definition of a SDNNF, since with this assumption, it is enough to consider only binary v-trees. Note also that we can make this assumption without loss of generality due to the associativity and commutativity of conjunction.



(a) SDNNF D_P. **(b)** v-tree T.

Figure 2 An example of an SDNNF D_P and the corresponding v-tree T. This particular SDNNF is the result of our construction on the BCT P from Example 3. The labels of the nodes mark the steps of our construction.

▶ **Definition 6** ([20]). *A DNNF D respects a v-tree T if for every conjunction $v = v_1 \wedge v_2$ in D, there is a node t in T, such that $var(v_1) \subseteq var(t^l)$ and $var(v_2) \subseteq var(t^r)$.*

Let v be a node in a DNNF D that respects a v-tree T. The *decomposition node* (*d-node*) of v is defined as the deepest node d in T such that $var(v) \subseteq var(d)$.

▶ **Definition 7** ([20]). *A DNNF that respects a given v-tree T is denoted as DNNF$_T$. Moreover, the language of* structured DNNFs *(*SDNNF*) consists of all DNNF$_T$ for any v-tree T.*

Given a DNNF$_T$ D, we can construct an equivalent smooth DNNF$_T$ D' in quadratic time [23]. It means that we can always assume that the input DNNF$_T$ is smooth.

▶ **Example 8.** Figure 2 shows an example of a smooth DNNF$_T$. In particular, DNNF$_T$ D_P on Figure 2a respects v-tree T on Figure 2b. DNNF$_T$ D_P represents the BCT constraint $(\mathbf{z}, P)$ where P is the BCT from Example 3.

For the construction of an SDNNF representing a BCT constraint, we will need the following operation for composing two v-trees. Assume T_1 and T_2 are two v-trees on disjoint sets of variables, i.e. $var(T_1) \cap var(T_2) = \emptyset$. Then $T = T_1 \circ T_2$ denotes the v-tree with a newly added root σ whose left child node σ^l is set to the root of T_1 and the right child node σ^r is set to the root of T_2. It follows that $var(T) = var(T_1) \cup var(T_2)$.

3 Compiling a BCT Constraint into a Structured DNNF

We shall show in this section that we can construct an SDNNF representing a given BCT constraint in polynomial time.

▶ **Theorem 9.** *Let $c^* = (\mathbf{x}, P)$ be a BCT constraint where $P = (\mathbf{z}, C)$ is a BCT. Then there is a smooth SDNNF D representing c^* with $O(md)$ nodes and $O(md^2)$ edges where $m = |\mathbf{z}|$ and $d = \max_{z_i \in \mathbf{z}} |dom(z_i)|$.*

We will describe the construction of D in the rest of this section, thus proving Theorem 9. Assume that $n = |\mathbf{x}|$ and $\mathbf{z} = \{z_1, \dots, z_m\}$ where $\mathbf{x} \subseteq \mathbf{z}$ and $m \geq n$ is the number of all variables. We assume that $|dom(z_i)| \leq d$ for every $i = 1, \dots, m$.

The construction proceeds in two steps. First, we describe a construction of a SDNNF D_P representing P. A SDNNF D that represents c^* then originates from D_P by forgetting the hidden variables $\mathbf{y} = \mathbf{z} \setminus \mathbf{x}$. This step can be done efficiently by [20].

Let G be the constraint graph of P. G is a tree with the set of nodes $\mathbf{z}$, each edge corresponds to a single constraint from C. Let G^+ denote a directed tree that originates from G by picking an arbitrary node as a root and directing all edges from the root towards the leaves. Let us assume that the nodes $z_1, \dots, z_m$ are ordered in a reverse topological order with respect to G^+. It means that z_m is the root and if (z_i, z_j) is an edge in G^+, then $i > j$. See Figure 1a for an example.

For every $i = 1, \dots, m$, let us consider the subtree G_i of G^+ rooted at z_i. Let $C_i \subseteq C$ denote the set of constraints corresponding to the edges of G_i. Denote $\mathbf{z}_i = \bigcup_{c \in C_i} scp(c)$. In this way, we have defined BCT $P_i = (\mathbf{z}_i, C_i)$. For every value $a \in dom(z_i)$, we also define BCT $P_{i,a}$ as a restriction of P_i to the solutions that contain literal (z_i, a). This can be best understood as adding a unary constraint with scope z_i and a single relation (z_i, a) to C_i. Another way of looking at it is restricting the relation of every constraint $c \in C_i$ with $z_i \in scp(c)$ to the tuples containing (z_i, a) and setting $dom(z_i)$ to $\{a\}$.

The algorithm proceeds for every $i = 1, \dots m$ in order and constructs for every $a \in dom(z_i)$ a SDNNF $D_{i,a}$ representing BCT constraint $(\mathbf{z}_i, P_{i,a})$ and a v-tree T_i that is respected by $D_{i,a}$ using the following steps:

(A1) If z_i is a leaf (no edges leave z_i in G^+), then P_i has no constraints and $P_{i,a}$ has the domain of z_i restricted to the single value a. DNNF $D_{i,a}$ is a single node labeled with literal (z_i, a) and v-tree T_i is a single node labeled with variable z_i.

(A2) Assume that z_i is not a leaf and it has k outgoing edges $(z_i, z_{i_1}), \dots, (z_i, z_{i_k})$ associated with constraints $c_1, \dots, c_k \in C$. For every $p = 1, \dots, k$ we have that $i_p < i$, because the nodes are processed in a reverse topological order, and thus we have already constructed $D_{i_p,b}$ and T_{i_p} for each $b \in dom(z_{i_p})$. Let us now construct $D_{i,a}$ for $a \in dom(z_i)$ in the following two steps.

(A2a) For every $p = 1, \dots, k$, define $E^{i_p}_{i,a}$ as follows:

$$E^{i_p}_{i,a} = \bigvee_{\{(z_i,a),(z_{i_p},b)\} \in rel(c_p)} D_{i_p,b}.$$

(A2b) We construct $D_{i,j}$ as a conjunction of literal (z_i, a) with all DNNFs $E_{i,a}^{i_p}$ for $p = 1, \ldots, k$. However, since we only allow conjunctions with two inputs, we construct $D_{i,a}$ in $k+1$ steps as follows.

$$D_{i,a}^{(1)} = E_{i,a}^{i_1} \tag{1}$$
$$D_{i,a}^{(p)} = D_{i,a}^{(p-1)} \wedge E_{i,a}^{i_p} \qquad \text{for } p = 2, \ldots, k \tag{2}$$
$$D_{i,a} = (z_i, a) \wedge D_{i,a}^{(k)} \tag{3}$$

In addition, we define T_i as follows:

$$T_i^{(1)} = T_{i_1} \tag{4}$$
$$T_i^{(p)} = T_i^{(p-1)} \circ T_{i_p} \qquad \text{for } p = 2, \ldots, k \tag{5}$$
$$T_i = z_i \circ T_i^{(k)} \tag{6}$$

Variable z_i is identified with a tree consisting of a single leaf labeled with z_i in step (6).

Once we have constructed $D_{m,a}$ for every $a \in dom(z_m)$, we compose them to obtain D_P that respects v-tree $T = T_m$ as follows:

$$D_P = \bigvee_{a \in dom(z_m)} D_{m,a}. \tag{7}$$

Intuitively, $E_{i,a}^{i_p}$ represents the fact that (z_i, a) has a support $((z_i, a), (z_{i_p}, b))$ in c_p. Moreover, the constraints below z_{i_p} in G^+ can be satisfied with the value of z_{i_p} set to b. SDNNF $D_{i,a}$ represents the models of all constraints that correspond to the edges in the subtree of G^+ rooted at z_i and that contain literal (z_i, a). If (z_i, a) does not have a support in c_p for some $p = 1, \ldots, k$, then the empty disjunction $E_{i,a}^{i_p}$ is equal to constant 0 and so is $D_{i,a}$. However, it is possible that $E_{i,a}^{i_p}$ is inconsistent even if (z_i, a) has a support $((z_i, a), (z_{i_p}, b))$ in c_p, but $D_{i_p,b}$ is inconsistent for every such b.

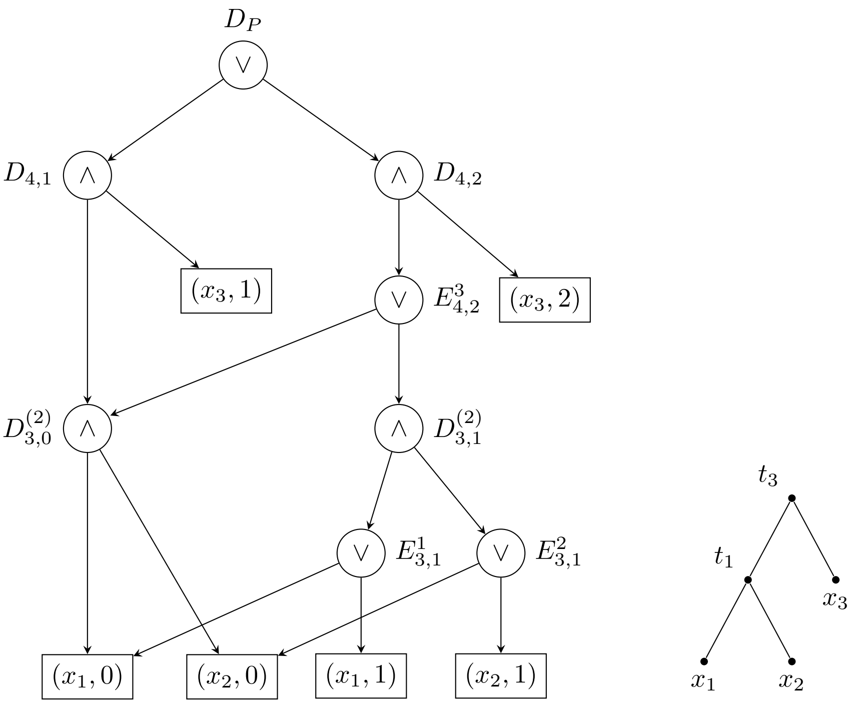
▶ **Example 10.** Figure 2 shows the result of the construction when applied to the BCT P from Example 3. Note that there is no leaf labeled with literal $(z_4, 0)$, because $(z_4, 0)$ has no support in constraint c_3. For the same reason, there is no leaf labeled with literal $(z_3, 2)$. Consequently, there are no leaves labeled with literals $(z_1, 2)$ and $(z_2, 2)$. It is worth noting that value 2 would be removed from the domains of variables z_1, z_2, and z_3 and value 0 would be removed from the domain of z_4 when enforcing arc consistency. In this way, enforcing arc consistency is part of the construction.

If variable z_3 would be forgotten from D_P, we would obtain a SDNNF representing the constraint c^* from Example 3. This would amount to replacing leaves labeled with literals $(z_3, 0)$ and $(z_3, 1)$ with constant 1. In this case, it just means removing these leaves altogether. Afterwards, we could simplify the SDNNF by removing the trivial gates with a single input, the result of this simplification can be seen in Figure 3.

We will now show that D_P is a smooth SDNNF of polynomial size that represents P. We will start by showing that D_P is a smooth SDNNF.

▶ **Lemma 11.** *D_P is a smooth SDNNF that respects v-tree T_m.*

Proof. Let us first show that D_P is an SDNNF that respects v-tree T_m. We will proceed by induction on $i = 1, \ldots, m$. If z_i is a leaf of G^+, then by step (A1), $D_{i,a}$ consists of a single leaf node for every $a \in dom(z_i)$. It follows that $D_{i,a}$ respects T_i which also consists of a single leaf node.



(a) Simplified SDNNF. **(b)** Simplified v-tree T.

Figure 3 Simplified SDNNF for the BCT constraint c^* from Example 3 representing constraint $\max(x_1, x_2) < x_3$. The SDNNF originated from the SDNNF in Figure 2a by forgetting z_3, identifying $x_1 = z_1, x_2 = z_2, x_3 = z_4$, and removing gates with a single input.

Let us now assume that z_i is not a leaf, in particular $i > 1$. Let us assume that z_i has k outgoing edges to nodes z_{i_1} to z_{i_k}. Assume a value $a \in dom(z_i)$. By induction hypothesis, for every $p = 1, \dots, k$ and every $b \in dom(z_{i_p})$ we have that $D_{i_p,b}$ is a SDNNF respecting v-tree T_{i_p}. It follows that $E^{i_p}_{i,a}$ constructed in step (A2a) is a SDNNF respecting T_{i_p}. We have that $var(E^{i_p}_{i,a}) = \mathbf{z}_{i_p}$. Since the subtrees rooted at nodes $z_{i_1}, \dots, z_{i_k}$ are pairwise disjoint, the same is true for sets $\mathbf{z}_{i_1}, \dots, \mathbf{z}_{i_p}$. Moreover, variable z_i is not in any of these sets. Therefore, $D_{i,a}$ constructed in step (A2b) is a DNNF. The construction of the tree T_i proceeds in a way similar to the construction of $D_{i,a}$, and thus $D_{i,a}$ respects T_i. In particular, the node of T_i introduced in (5) is the d-node of the conjunction (2) for the same value of p, and the node introduced in (6) is the d-node of the conjunction (3).

D_P is constructed in step (7) as a disjunction of $D_{m,a}$, $a \in dom(z_m)$. As each of these SDNNFs respects T_m, the same is true for D_P.

Let us now show the smoothness. Assume that $D_{i,a}$ is nontrivial, i.e. it is not just a single leaf labeled with 0. We show by induction that then $D_{i,a}$ is a SDNNF that depends on all variables in $\mathbf{z}_i$. This is true for the leaves. If z_i is not a leaf, $D_{i,a}$ is constructed in step (A2b). By induction hypothesis used on each $D_{i_p,b}$ we get that $E^{i_p}_{i,a}$ is a smooth disjunction that depends on all variables in $\mathbf{z}_{i_p}$. Thus also $D_{i,a}$ depends on all variables in $\mathbf{z}_i = \{z_i\} \cup \bigcup_{p=1}^{k} \mathbf{z}_{i_p}$. It follows that also the disjunction introduced in the final step (7) is smooth. ◀

Now, let us estimate the size of D_P.

▶ **Lemma 12.** *SDNNF D_P has $O(md)$ nodes and $O(md + s)$ edges where $s = \sum_{c \in C} |rel(c)|$.*

Proof. For every $i = 1, \dots, m$, we add one disjunction for each value $a \in dom(z_i)$ in step (A2a) and k conjunctions in step (A2b) assuming z_i is not a leaf. One more disjunction gate is added in the final step (7). Altogether, we thus add $O(md)$ gates to D_P. Each conjunction gate has at most two inputs, thus $O(md)$ edges are leaving the conjunction gates. For every constraint c_p with scope $\{z_i, z_{i_p}\}$, every tuple $((z_i, a), (z_{i_p}, b))$ adds one edge leaving disjunction gate $E^{i_p}_{i,a}$. The total number of edges leaving the disjunction gates added in step (A2a) is thus at most s. The disjunction gate added in the final step has at most $|dom(z_i)| \leq d$ inputs. Altogether, we have $O(md + s)$ edges in D_P. ◀

It remains to show that D_P represents P.

▶ **Lemma 13.** *SDNNF D_P represents P.*

Proof. We shall first show by induction on i that $D_{i,a}$ represents $P_{i,a}$ for every $i = 1, \dots, m$ and $a \in dom(z_i)$. This is true for leaves (and thus also for $i = 1$), because $P_{i,a}$ does not have any constraints, it depends only on z_i, and the domain of z_i is restricted to the value a. A single node labeled with literal (z_i, a) added in step (A1) is thus a correct representation of $P_{i,a}$ in this case.

Let us now consider a variable z_i with outgoing edges $(z_i, z_{i_1}), \dots, (z_i, z_{i_k})$ associated with constraints $c_1, \dots, c_k \in C$ where $k \geq 1$. C_i is thus a disjoint union of C_{i_p}, $p = 1, \dots, k$ with $\{c_1, \dots, c_k\}$. Let us also consider a value $a \in dom(z_i)$. Let us assume by induction hypothesis that each $D_{i_p,b}$ represents $P_{i_p,b}$ for every $b \in dom(z_{i_p})$. Recall that the scope of $P_{i,a}$ is the set $\mathbf{z}_i$ of variables in the subtree of G^+ rooted at z_i. Let τ be a tuple of variables $\mathbf{z}_i$ and let us fix some $a \in dom(z_i)$.

Let us first assume that $\tau \in sol(P_{i,a})$. It follows that $(z_i, a) \in \tau$. We will show that τ satisfies $D_{i,a}$. Let us consider literals $(z_{i_1}, b_1), \dots, (z_{i_k}, b_k) \in \tau$. For every $p = 1, \dots, k$ we have that τ satisfies P_{i_p}. It satisfies P_{i_p,b_p} as well since $(z_{i_p}, b_p) \in \tau$. By induction hypothesis, circuit D_{i_p,b_p} represents P_{i_p,b_p} and thus it evaluates to true on τ. By definition of $P_{i,a}$, τ satisfies P_i and thus also constraint c_p. It follows that $\{(z_{i_p}, b_p), (z_i, a)\} \in rel(c_p)$ and thus, by step (A2a), also $E^{i_p}_{i,a}$ evaluates to true. Since this holds for every $p = 1, \dots, k$ and $(z_i, a) \in \tau$, we have by step (A2b) that $D_{i,a}$ evaluates to true on τ.

Let us now assume that $D_{i,a}$ evaluates to true on τ and let us show that $\tau \in sol(P_{i,a})$. We have $(z_i, a) \in \tau$ by (3), it remains to show that $\tau \in sol(P_i)$. Let $p \in \{1, \dots, k\}$ be arbitrary. We have by (1) to (3) that $E^{i_p}_{i,a}$ evaluates to true. By (A2a), we have that D_{i_p,b_p} evaluates to true on τ for some $\{(z_i, a), (z_{i_p}, b_p)\} \in rel(c_p)$. Induction hypothesis implies $\tau[\mathbf{z}_{i_p}] \in sol(P_{i_p,b_p})$ and thus also $(z_{i_p}, b_p) \in \tau$. Together with $(z_i, a) \in \tau$ we obtain that c_p is satisfied by τ. In addition, all constraints in P_{i_p} are satisfied by τ. Since this holds for every $p = 1, \dots, k$, we get that all constraints of P_i are satisfied and thus $\tau \in sol(P_i)$.

Let us now show that D_P represents P. If τ is a tuple satisfying P and $(z_m, a) \in \tau$, then τ satisfies $P_{m,a}$. It follows that $D_{m,a}$ evaluates to true and that D_P evaluates to true as well. If, on the other hand, D_P evaluates to true on τ, then $D_{m,a}$ evaluates to true for some $a \in dom(z_m)$. Thus τ is a solution of both $P_{m,a}$ and $P = P_m$. ◀

Theorem 9 now follows from the above propositions.

Proof of Theorem 9. The SDNNF D representing c^* originates from D_P by forgetting variables $\mathbf{y} = \mathbf{z} \setminus \mathbf{x}$. This step can be performed in polynomial time by [20] by replacing the literals on variables from $\mathbf{y}$ with constants 1 and then propagating these constants. Note that in D_P, this is equivalent to removing the corresponding leaves which were added in (3). In particular, this step preserves smoothness which is ensured for D_P by Lemma 11. The size bound on D follows from Lemma 12 using the fact that $|C| \leq m$ and $|rel(c)| \leq d^2$ for every $c \in C$. ◀

4 Compiling an SDNNF to a BCT Constraint

In this section, we shall show the following theorem.

▶ **Theorem 14.** *Let c^* be a constraint represented by a smooth SDNNF. Then there is a tree binary encoding of c^* of polynomial size.*

It is useful to look at a DNNF in terms of *certificates* [4], also called *minimal satisfying subtrees* in [16]. A certificate for a satisfying assignment is simply a minimal satisfied sub-DNNF that contains the output gate. Due to decomposability, the certificates of a DNNF are trees whose leaves are some of the leaves of the DNNF. In addition, no two leaves of a certificate are labeled with a literal of the same variable. Assume D is a smooth DNNF on variables $\mathbf{x} = (x_1, \ldots, x_n)$ and S its certificate. We shall also assume that D has no leaves labeled with constants 0 or 1 (as these can always be propagated). Then for each $i = 1, \ldots, n$, we have that S contains exactly one leaf associated with a literal of variable x_i (see also [16] for more details). The leaves of S thus determine a tuple τ on which D evaluates to true. We say in this case that the leaves of S are *associated with the literals in tuple* τ. The certificates are thus in one-to-one correspondence with the satisfying assignments of D.

Let us now fix a constraint c^* with $scp(c^*) = \mathbf{x}$. Let us assume that c^* is represented by a smooth SDNNF D that respects a v-tree T and let ρ denote the root of D. In particular, $var(T) = var(\rho) = \mathbf{x}$. Then the certificates of D also respect T. In fact, we will show that if S is a certificate of D, then the conjunction gates of S are in one-to-one correspondence with the inner nodes of T. This property lies at the basis of our construction of a tree binary encoding $P = (\mathbf{z}, C)$ of c^*. The idea is to introduce a hidden variable for each inner node t of T with the domain being the $\wedge$-gates whose d-node is t. The constraints make sure that the models of P are in one-to-one correspondence with the certificates of D.

We will construct a BCT $P = (\mathbf{z}, C)$ satisfying $\mathbf{x} \subseteq \mathbf{z}$ and $rel(c^*) = sol(P)[\mathbf{x}]$. For each inner node t of T, we introduce a hidden variable y_t. The set of all these hidden variables will be denoted as $\mathbf{y}$. We then define $\mathbf{z} = \mathbf{x} \cup \mathbf{y}$. The domain of an original variable $x_i \in \mathbf{x}$ is $dom(x_i)$ as given by the constraint c^*. For a hidden variable $y_t \in \mathbf{y}$, we set $dom(y_t) = \Lambda(t)$ where $\Lambda(t)$ denotes the set of conjunction gates of D that have d-node t.

The constraints of C correspond to the edges of T. In particular, for each edge (t, t') of T where t' is a child node of t, we add a constraint $c_{t,t'}$ to C whose definition differs slightly depending on whether t' is a leaf or an inner node of T. A sequence of vertices $v_0, \ldots, v_k$ of D is called an $\vee$*-path* if it is a path, nodes $v_1, \ldots, v_{k-1}$ are $\vee$-gates, v_0 is a conjunction gate and v_k is either a conjunction gate, or a leaf node.

(C1) Assume t' is a leaf labeled with variable x_i. Then $scp(c_{t,t'}) = \{y_t, x_i\}$ and $rel(c_{t,t'})$ is defined as a set of tuples $\{(y_t, v), (x_i, a)\}$ such that D contains a $\vee$-path from v to the leaf labeled with literal (x_i, a).

(C2) Assume t' is an inner node of T. Then $scp(c_{t,t'}) = \{y_t, y_{t'}\}$ and $rel(c_{t,t'})$ is defined as a set of tuples $\{(y_t, v), (y_{t'}, v')\}$ such that D contains a $\vee$-path from v to v'.

▶ **Example 15.** Figure 4 shows the result of the application of the construction to the SDNNF from Figure 2. Recall that the SDNNF itself was constructed as a representation of the BCT P from Example 3. BCT in Figure 4 differs from P in that it has three hidden variables y_1, y_2, and y_3. Note that y_1 and y_2 are basically equivalent to z_3 and y_3 is equivalent to z_4. The auxiliary variables can thus be easily eliminated by which we obtain the constraints of P.

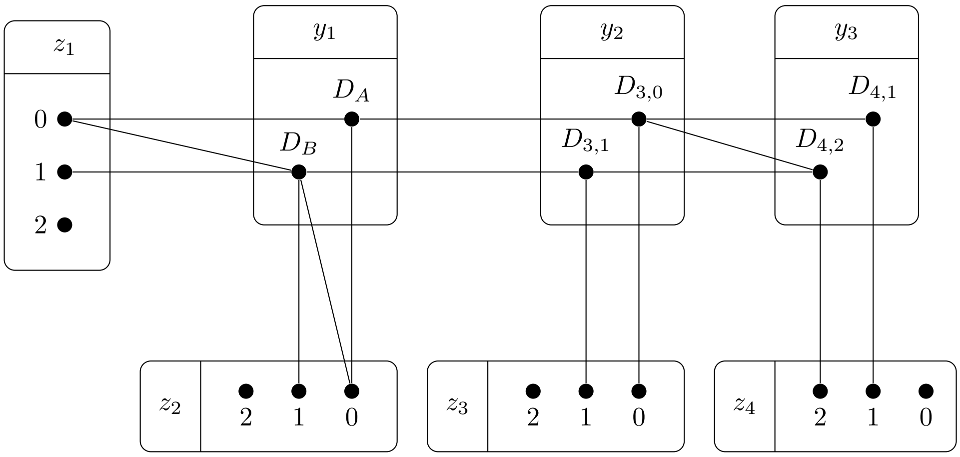


Figure 4 Example of the construction of the tree binary encoding of a constraint represented by the SDNNF D in Figure 2.

The size of P defined in this way is clearly polynomial in the size of D. The rest of this section is devoted to showing the correctness of the construction. We will start with a few technical propositions on the structure of the certificates of D.

▶ **Lemma 16.** *Assume v is a node of D with d-node t in T. Then $var(v) = var(t)$. Moreover,*
1. *if $v = v_1 \wedge v_2$, then t^l is the d-node of v_1 and t^r is the d-node of v_2, and*
2. *if $v = v_1 \vee \cdots \vee v_k$, then t is the d-node of all input nodes $v_1, \ldots, v_k$.*

Proof. We will proceed by the induction on the structure of D. If $v = \rho$ is the root of D, then $var(v) = \mathbf{x}$. It follows that its d-node t is the root of T and thus $var(t) = \mathbf{x} = var(v)$.

Let us assume that $v = v_1 \wedge v_2$ and that $var(v) = var(t)$. Since D does not contain leaves labeled with constants, we have that both $var(v_1)$ and $var(v_2)$ are nonempty and thus $var(v_i) \subsetneq var(v)$ for $i = 1, 2$ and $var(v) = var(v_1) \cup var(v_2)$. Let t_i be the d-node of v_i, $i = 1, 2$. By the definition of structured DNNFs, both t_1 and t_2 are descendants of t in T and thus $var(t_i) \subseteq var(t)$, $i = 1, 2$. By the definition of d-nodes, we also have that $var(v_i) \subseteq var(t_i)$, $i = 1, 2$. It follows that $var(t) = var(v) = var(v_1) \cup var(v_2) \subseteq var(t_1) \cup var(t_2) \subseteq var(t)$ and thus $var(t) = var(t_1) \cup var(t_2)$. The only possibility is that both t_1 and t_2 are the child nodes of t and thus $t_1 = t^l$, $t_2 = t^r$, and $var(v_i) = var(t_i)$, $i = 1, 2$.

Assume that $v = v_1 \vee \cdots \vee v_k$ and $var(v) = var(t)$. By smoothness we get that $var(t) = var(v) = var(v_1) = \cdots = var(v_k)$. It follows that t is the d-node of all input nodes $v_1, \ldots, v_k$.

We have shown that if $var(v) = var(t)$ and v' is a child node of v with d-node t', then $var(v') = var(t')$ which also holds for the leaves. ◀

▶ **Lemma 17.** *Assume that $v_0, \ldots, v_k$ is a $\vee$-path with $k > 0$. Assume that t is the d-node of v_0 and t' is the d-node of v_k. Then $v_1, \ldots, v_{k-1}$ have d-node t' and t' is a child node of t.*

Proof. By smoothness, $var(v_1) = var(v_2) = \cdots = var(v_k)$ and thus all gates $v_1, \ldots, v_{k-1}$ have the same d-node as v_k. In particular, t' is the d-node of v_1 which is an input to the conjunction gate v_0. By Lemma 16 we have that t' is a child node of t. ◀

Based on Lemma 17, we can show the following proposition.

▶ **Lemma 18.** *Assume S is a certificate and t is an inner node of T. Then S contains exactly one conjunction gate v from $\Lambda(t)$.*

Proof. Consider a variable $x_i \in var(t)$ and the path $v_0, \dots, v_k$ in S that leads from the root $v_0 = \rho$ to the leaf v_k of S labeled with a literal on x_i. It follows that $var(v_j) \subseteq var(v_{j-1})$ for every $j = 1, \dots, k$. Moreover $var(v_0) = \mathbf{x}$ and $var(v_k) = \{x_i\}$. Let $v_{i_1}, \dots, v_{i_p}$ be the subsequence of $v_0, \dots, v_k$ formed only by conjunction nodes. Then by Lemma 17 we get that $t_{i_1}, \dots, t_{i_p}, t_k$ form a path in T from the root to the leaf labeled with x_i. For some index i_j we thus have that $t_{i_j} = t$ and it follows that v_{i_j} is a conjunction gate in S with d-node t.

Let us now assume that S contains two $\wedge$-gates v_1 and v_2 with the same d-node t, thus $var(v_1) = var(v_2) = var(t)$ by Lemma 16. However, Lemma 16 also implies that there is no path from v_1 to v_2 or from v_2 to v_1. If we take the paths from the root ρ to v_1 and v_2 in S, they have to split in a $\wedge$-gate v (by minimality of S), but then v is not decomposable.

It follows that v_{i_j} is the only conjunction gate in S that belongs to $\Lambda(t)$. ◀

Note that each literal on a variable from $\mathbf{z} = \mathbf{x} \cup \mathbf{y}$ is associated with a node of D. In particular, for $x_i \in \mathbf{x}$, literal (x_i, a) is associated with the leaf of D labeled with (x_i, a). To this end, we need to assume that every such literal has a leaf labeled with it. However, if D does not contain any leaf associated with literal (x_i, a), then this literal does not have a support in c^* and a can be removed from $dom(x_i)$. We may thus assume that no such value is in $dom(x_i)$. For an inner node t of T, a literal (y_t, v) is associated with the node $v \in \Lambda(t)$.

▶ **Lemma 19.** *Let $\tau \in sol(P)$ be a tuple that is a solution to P. Then $\tau[\mathbf{x}] \in rel(c^*)$.*

Proof. Since D represents c^*, it is enough to show that there is a certificate S of D whose leaves are associated with the literals in $\tau[\mathbf{x}]$.

Tuple τ associates a node v of D with every node t of T. We proceed by induction on the structure of T to describe a certificate S_t for the sub-DNNF of D rooted at v.

Assume first that t is a leaf of T labeled with variable x_i. Consider the literal $(x_i, a) \in \tau$ and set the certificate S_t to a single node labeled with this literal.

Assume now that t is an inner node of T. Since t is an inner node of T, we have that $(y_t, v) \in \tau$ for some $v \in \Lambda(t)$. Tuple τ also contains literals associated with t^l and t^r. These literals associate a nodes v_l and v_r of D with t^l and t^r respectively. By induction hypothesis, we have constructed certificate S_l and S_r for the sub-DNNFs rooted at v_l and v_r respectively. Since τ satisfies constraints c_{t,t^l} and c_{t,t^r}, D contains a $\vee$-paths from v to v_l and from v to v^r. Certificate S_t for the the sub-DNNF rooted at v is then constructed as a union of S_l, S_r, node v and the $\vee$-paths from v to v_l and v_r.

Let σ be the root of T and let us assume that v is the node of D associated with σ by τ. Let S_σ be the certificate of the sub-DNNF rooted at v. If $v = \rho$ is the root of D, then $S = S_\sigma$ is a certificate of D. Otherwise, D contains a path from ρ to v that consists only of $\vee$-gates and we construct S by combining this path with S_σ. ◀

▶ **Lemma 20.** *For every $\tau^* \in rel(c^*)$, there is $\tau \in sol(P)$ satisfying $\tau^* = \tau[\mathbf{x}]$.*

Proof. Since $\tau^* \in rel(c^*)$, there is a certificate S of D whose leaves are associated with the literals from τ^*. By Lemma 18, the certificate S contains exactly one conjunction gate $v_t \in \Lambda(t)$ for each inner node t. We form τ by adding literals (y_t, v_t) to τ^* for all internal nodes t of T. Let us check that τ satisfies all constraints of P. Let $c_{t,t'}$ be a constraint of P where t' is a child node of t in T. By Lemma 16, one of the child nodes of v_t in D has d-node t', let us denote it v_1. Since v_t is a conjunction gate, v_1 must belong to S. If v_1 is a disjunction, then by Lemma 16, its child nodes have d-node t', too. If we follow the path in S from v_1 to a leaf or to the next conjunction gate, we get a $\vee$-path that ends in the node v_k whose d-node is still t' and v_k is either a leaf or a conjunction gate.

If t' is a leaf of T labeled with variable x_i, we must have that v_k is a leaf of S labeled with a literal (x_i, a) for some $a \in dom(x_i)$, it follows that $(x_i, a) \in \tau^* \subseteq \tau$ and $\{(y_t, v_t), (x_i, a)\} \in rel(c_{t,t'})$, constraint $c_{t,t'}$ is thus satisfied by τ.

If t' is an inner node, then v_k is a conjunction gate $v_{t'}$ associated with t' in S. It follows that $(y_{t'}, v_{t'}) \in \tau$ and $\{(y_t, v_t), (y_{t'}, v_{t'})\} \in rel(c_{t,t'})$, constraint $c_{t,t'}$ is thus satisfied by τ. ◀

Theorem 14 now follows by the above construction from the following proposition.

▶ **Theorem 21.** *$P = (\mathbf{z}, C)$ is a tree binary encoding of c^* of polynomial size.*

Proof. C consists of $O(n)$ constraints and the total size of the domains of variables in $\mathbf{z}$ is bounded by the number of the nodes in D. Lemmas 19 and 20 imply that P is a TBE of c^*. ◀

5 Binary Constraint Graphs With Bounded Treewidth

In this section, we shall extend the construction from Section 3 to BCG constraints that naturally generalize BCT constraints.

▶ **Definition 22.** *A* BCG constraint *c is a pair $(\mathbf{x}, P)$ such that $P = (\mathbf{z}, C)$ is a normalized binary CSP, $scp(c) = \mathbf{x} \subseteq \mathbf{z}$ and $rel(c) = sol(\mathbf{z}, C)[\mathbf{x}]$.*

The construction we describe is parameterized by the treewidth of the underlying constraint graph and the domain size. The treewidth of a graph is defined using a tree decomposition.

Given an undirected graph $G = (V, E)$, its *tree decomposition* is defined as a pair (T, χ) where $T = (V_T, E_T)$ is a tree and $\chi : V_T \to \mathcal{P}(V)$ is a function that assigns each vertex $t \in V_T$ a subset of V called a *bag* that satisfies the following conditions:

(d1) $V = \bigcup_{t \in V_T} \chi(t)$.

(d2) For each edge $\{u, v\} \in E$ there is a node $t \in V_T$ such that $\{u, v\} \subseteq \chi(t)$.

(d3) If a node v is contained in two bags $\chi(t_1)$ and $\chi(t_2)$, then $v \in \chi(t)$ for every node t on the path connecting t_1 with t_2.

The *width* of the tree decomposition is defined as $\max_{t \in V_T} |\chi(t)| - 1$. The *treewidth* $tw(G)$ of G is the minimum width among all possible tree decompositions of G. It should be noted that any tree decomposition of a graph G on n vertices can be transformed into a tree decomposition with the same width and $O(n)$ nodes [15].

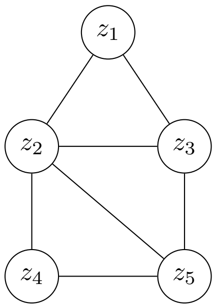
We are now ready to formulate the main result of this section.

▶ **Theorem 23.** *Assume that $c^* = (\mathbf{x}, P)$ is a BCG constraint defined by a normalized binary CSP $P = (\mathbf{z}, C)$. Denote G the constraint graph of P. Denote $m = |\mathbf{z}|$ and $d = \max_{z_i \in \mathbf{z}} |dom(z_i)|$. Then there is an SDNNF D representing c^* with $O(md^{tw(G)+1})$ nodes and $O(md^{2\,tw(G)+1})$ edges.*

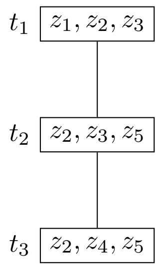
The proof of Theorem 23 is based on the following proposition.

▶ **Theorem 24.** *Assume that $c^* = (\mathbf{x}, P)$ is a BCG constraint defined by a normalized binary CSP $P = (\mathbf{z}, C)$. Denote G the constraint graph of P. Denote $m = |\mathbf{z}|$ and $d = \max_{z_i \in \mathbf{z}} |dom(z_i)|$. Then c^* has a tree binary encoding $P' = (\mathbf{z}', C')$ with $|\mathbf{z}'| = O(m)$ and $|dom(z'_i)| \le d^{tw(G)+1}$ for every $z'_i \in \mathbf{z}'$.*

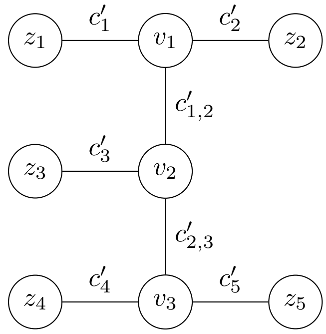
Note that Theorem 24 actually follows from Proposition 4 in [28] which is based on the encoding described in [11]. If we would simply combine the bound given by Theorem 24 with the bound given by Theorem 9, we would get an SDNNF D representing c^* with



(a) Binary constraint graph.



(b) Tree decomposition.



(c) Tree binary encoding.

Figure 5 Example of the construction, see the description in Example 25.

$O(md^{tw(G)+1})$ nodes and $O(md^{2(tw(G)+1)})$ edges. We provide a specific construction that proves Theorem 24 and that can be combined with Lemma 12 to prove a slightly better bound stated in Theorem 23. The construction we describe below is similar to the dual encoding described in [11].

Let us fix a BCG constraint $c^* = (\mathbf{x}, P)$ where $P = (\mathbf{z}, C)$ is a normalized binary CSP with constraint graph G. Let us assume that $\mathbf{z} = (z_1, \dots, z_m)$ and $d = \max_{i=1}^{m} |\mathit{dom}(z_i)|$. Let us also fix a tree decomposition (T, χ) of G. Let us assume that $V_T = (t_1, \dots, t_N)$ for some $N = O(m)$. We shall describe a BCT $P' = (\mathbf{z}', C')$ which is a TBE of c^*. First, let us define the variables in $\mathbf{z}'$. We associate a new variable v_i with every t_i, $i = 1, \dots, N$. Then we set $\mathbf{z}' = \mathbf{z} \cup \mathbf{v}$ where $\mathbf{v} = (v_1, \dots, v_N)$. The domains of variables in $\mathbf{z}$ are given by c^*. For every v_i, $i = 1, \dots, N$, we set the domain as follows. Let us consider the set of constraints defined on variables from $\chi(t_i)$ as $C_i = \{c \in C \mid \mathit{scp}(c) \subseteq \chi(t_i)\}$. Then the domain of v_i is defined as the set of solutions to CSP $(\chi(t_i), C_i)$, i.e. $\mathit{dom}(v_i) = \mathit{sol}(\chi(t_i), C_i)$.

Let us now define the constraints in C'.

(T1) For every edge $\{t_i, t_j\} \in E_T$ we add a constraint $c'_{i,j}$ into C' with $\mathit{scp}(c'_{i,j}) = \{v_i, v_j\}$. The constraint relation $\mathit{rel}(c'_{i,j})$ consists of pairs $\{(v_i, \tau_1), (v_j, \tau_2)\}$ where $\tau_1 \in \mathit{dom}(v_i)$, $\tau_2 \in \mathit{dom}(v_j)$, and $\tau_1[\chi(t_i) \cap \chi(t_j)] = \tau_2[\chi(t_i) \cap \chi(t_j)]$.

(T2) For every z_i, $i = 1, \dots, m$, we pick a representative node $t_{r_i} \in V_T$ satisfying $z_i \in \chi(t_{r_i})$. We then add a constraint c'_i into C' with $\mathit{scp}(c'_i) = \{z_i, t_{r_i}\}$. The set of tuples $\mathit{rel}(c'_i)$ consists of pairs $\{(z_i, a), (v_{r_i}, \tau)\}$ where $a \in \mathit{dom}(z_i)$, $\tau \in \mathit{dom}(v_{r_i})$, and $(z_i, a) \in \tau$.

▶ **Example 25.** Let us consider a binary CSP $P = (\mathbf{z}, C)$ with $\mathbf{z} = \{z_1, \dots, z_5\}$ whose constraint graph G is depicted in Figure 5a. We shall use $c_{i,j} \in C$ to denote the constraint with scope $\{z_i, z_j\}$. Figure 5b shows a tree decomposition T of the graph with the contents of the bags inside the rectangles. The structure of the tree binary encoding P' of P is then shown in Figure 5c. The domain of variable v_1 consists of tuples τ on variables z_1, z_2, and z_3 satisfying constraints $c_{1,2}$, $c_{2,3}$, and $c_{1,3}$. Assume a tuple $\sigma' \in \mathit{sol}(P')$. Constraint c'_2 makes sure that if $(z_2, a) \in \sigma'$, then σ' contains (v_2, τ) satisfying $(z_2, a) \in \tau$. Similarly for other variables. Constraints $c'_{1,2}$ and $c'_{2,3}$ extend this property also to nodes v_2 and v_3. The tuples assigned to variables v_1, v_2, and v_3 are thus consistent with each other and also with constraints C. We thus have that $\sigma'[\mathbf{z}] \in \mathit{sol}(P)$.

The proof of the correctness of the above construction and thus also the proof of Theorem 24 is moved to the appendix. Here, we will describe its application for proving the main result of this section.

Proof of Theorem 23. Using Theorem 24, we obtain a TBE encoding $P' = (\mathbf{z}', C')$ with $O(m)$ variables and the domain sizes bounded by $d^{tw(G)+1}$. We can then apply Theorem 9 to obtain an SDNNF D that represents c^*. D has $O(md^{tw(G)+1})$ nodes. By Lemma 12, the number of edges of D is bounded by $O(md^{tw(G)+1} + s)$ where $s = \sum_{c' \in C'} |rel(c')|$. Since $|C'| = O(m)$, it is enough to show that $|rel(c')| \leq d^{2\,tw(G)+1}$ for every $c' \in C'$.

Assume first a constraint $c'_{i,j}$ added in step (T1). We may assume that G is connected (otherwise we process each connected component of G separately) and therefore $\chi(t_i) \cap \chi(t_j) \neq \emptyset$. The number of pairs of tuples τ_1 and τ_2 that satisfy $\tau_1[\chi(t_i) \cap \chi(t_j)] = \tau_2[\chi(t_i) \cap \chi(t_j)]$ is thus at most $d^{tw(G)+1} \cdot d^{tw(G)} = d^{2\,tw(G)+1}$. Therefore $\left|rel(c'_{i,j})\right| \leq d^{2\,tw(G)+1}$.

Assume now a constraint c'_i added in step (T2). The number of tuples τ satisfying that $(z_i, a) \in \tau$ for one particular $a \in dom(z_i)$ is at most $d^{tw(G)}$ and thus $\left|rel(c'_{i,j})\right| \leq d^{tw(G)+1} \leq d^{2\,tw(G)+1}$. ◀

Note that the size estimate in Theorem 23 is only an upper bound and the real size of P' and the SDNNF D depends on the particular tree decomposition and, in particular, on how much the bags intersect. Therefore, there is a space for optimization in a practical setting.

6 Conclusion

As the main result of our paper, we have shown that binary constraint trees are polynomially equivalent to structured DNNF circuits. We would like to note that for a given BCT P the construction in Section 3 leads to a deterministic SDNNF D_P (thanks to rule 3 in step (A2b)). This means that for every pair of distinct children v_1 and v_2 of a disjunction node, the sub-NNFs rooted at v_1 and v_2 do not share any models (see [9, 20]). This property allows for instance model counting on D_P. However, forgetting the hidden variables from D_P does not preserve determinism in general [10] and thus the actual result of the construction is not a deterministic SDNNF. Introducing hidden variables is thus an important part of the construction described in Section 4 since SDNNFs are strictly more succinct than deterministic SDNNFs [20].

Several rules for reducing the number of hidden variables in a BCT constraint were described in [27], it would be interesting to investigate the effect of these rules on a SDNNF that is compiled into a BCT constraint, reduction rules are applied to it and then it is compiled back to a SDNNF. When compiling the BCT constraint back to a SDNNF, we can pick an arbitrary node of the constraint tree as a root which allows us to change the structure of the SDNNF to a different orientation of the v-tree. This, for instance, extends the applicability of the conjoin operation described in [20] to conjoining two SDNNFs whose v-trees differ, but their undirected versions are the same.

References

1 Ignasi Abío, Graeme Gange, Valentin Mayer-Eichberger, and Peter J. Stuckey. On CNF encodings of decision diagrams. In Claude-Guy Quimper, editor, *Integration of AI and OR Techniques in Constraint Programming*, pages 1–17, Cham, 2016. Springer International Publishing.

2 Christian Bessiere and Jean-Charles Régin. Arc consistency for general constraint networks: preliminary results. In *IJCAI (1)*, pages 398–404, 1997.

3 Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. On compiling CNFs into structured deterministic DNNFs. In Marijn Heule and Sean Weaver, editors, *Theory and Applications of Satisfiability Testing – SAT 2015*, pages 199–214, Cham, 2015. Springer International Publishing.

4 Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. Knowledge compilation meets communication complexity. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 1008–1014. AAAI Press, 2016. URL: `http://dl.acm.org/citation.cfm?id=3060621.3060761`.

5 Kenil C. Cheng and Roland H. Yap. Maintaining generalized arc consistency on ad hoc r-ary constraints. In *Proceedings of the 14th International Conference on Principles and Practice of Constraint Programming*, CP '08, pages 509–523, Berlin, Heidelberg, 2008. Springer-Verlag. `doi:10.1007/978-3-540-85958-1_34`.

6 Kenil C. Cheng and Roland H. Yap. An mdd-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. *Constraints*, 15(2):265–304, April 2010. `doi:10.1007/s10601-009-9087-y`.

7 Adnan Darwiche. Compiling knowledge into decomposable negation normal form. In *Proceedings of the 16th International Joint Conference on Artifical Intelligence - Volume 1*, IJCAI'99, pages 284–289, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. URL: `http://dl.acm.org/citation.cfm?id=1624218.1624260`.

8 Adnan Darwiche. Decomposable negation normal form. *J. ACM*, 48(4):608–647, July 2001. `doi:10.1145/502090.502091`.

9 Adnan Darwiche. A compiler for deterministic, decomposable negation normal form. In *Eighteenth National Conference on Artificial Intelligence*, pages 627–634, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence. URL: `http://dl.acm.org/citation.cfm?id=777092.777189`.

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

11 Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38(3):353–366, April 1989. `doi:10.1016/0004-3702(89)90037-4`.

12 Eugene C. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the ACM*, 32(4):755–761, October 1985. `doi:10.1145/4221.4225`.

13 Eugene C. Freuder. Complexity of K-tree structured constraint satisfaction problems. In *Proceedings of the eighth National conference on Artificial intelligence - Volume 1*, AAAI'90, pages 4–9, Boston, Massachusetts, 1990. AAAI Press.

14 Graeme Gange and Peter J. Stuckey. Explaining propagators for s-DNNF circuits. In Nicolas Beldiceanu, Narendra Jussien, and Éric Pinson, editors, *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimization Problems*, pages 195–210. Springer Berlin Heidelberg, 2012.

15 Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994. `doi:10.1007/BFb0045375`.

16 Petr Kučera and Petr Savický. Propagation complete encodings of smooth DNNF theories. *Constraints*, 27(3):327–359, July 2022. `doi:10.1007/s10601-022-09331-2`.

17 Christophe Lecoutre. STR2: optimized simple tabular reduction for table constraints. *Constraints*, 16(4):341–371, October 2011. `doi:10.1007/s10601-011-9107-6`.

18 Christophe Lecoutre, Chavalit Likitvivatanavong, and Roland H. C. Yap. STR3: A path-optimal filtering algorithm for table constraints. *Artificial Intelligence*, 220:1–27, March 2015. `doi:10.1016/j.artint.2014.12.002`.

19 Robert Mateescu and Rina Dechter. Compiling Constraint Networks into AND/OR Multi-valued Decision Diagrams (AOMDDs). In Frédéric Benhamou, editor, *Principles and Practice of Constraint Programming - CP 2006*, Lecture Notes in Computer Science, pages 329–343, Berlin, Heidelberg, 2006. Springer. `doi:10.1007/11889205_25`.

20 Knot Pipatsrisawat and Adnan Darwiche. New compilation languages based on structured decomposability. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 1*, AAAI'08, pages 517–522. AAAI Press, 2008. URL: `http://dl.acm.org/citation.cfm?id=1619995.1620079`.
21 Knot Pipatsrisawat and Adnan Darwiche. Top-down algorithms for constructing structured DNNF: Theoretical and practical implications. In *Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, pages 3–8, Amsterdam, The Netherlands, The Netherlands, 2010. IOS Press. URL: `http://dl.acm.org/citation.cfm?id=1860967.1860970`.
22 Francesca Rossi, Charles J. Petrie, and Vasant Dhar. On the equivalence of constraint satisfaction problems. In *Proceedings of the 9th European Conference on Artificial Intelligence*, ECAI'90, pages 550–556, USA, 1990. Pitman Publishing, Inc.
23 Andy Shih, Guy Van den Broeck, Paul Beame, and Antoine Amarilli. Smoothing structured decomposable circuits. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL: `https://proceedings.neurips.cc/paper/2019/file/940392f5f32a7ade1cc201767cf83e31-Paper.pdf`.
24 Kostas Stergiou and Toby Walsh. Encodings of non-binary constraint satisfaction problems. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, AAAI '99/IAAI '99, pages 163–168, USA, 1999. American Association for Artificial Intelligence.
25 Ruiwei Wang and Roland H. C. Yap. Bipartite encoding: A new binary encoding for solving non-binary CSPs. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, IJCAI'20, Yokohama, Yokohama, Japan, 2021.
26 Ruiwei Wang and Roland H. C. Yap. CNF Encodings of Binary Constraint Trees. In Christine Solnon, editor, *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*, volume 235 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.CP.2022.40`.
27 Ruiwei Wang and Roland H.C. Yap. Encoding multi-valued decision diagram constraints as binary constraint trees. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(4):3850–3858, June 2022. `doi:10.1609/aaai.v36i4.20300`.
28 Ruiwei Wang and Roland H.C. Yap. The expressive power of ad-hoc constraints for modelling CSPs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(4):4104–4114, June 2023. `doi:10.1609/aaai.v37i4.25526`.

A Proof of Theorem 24

In this section, we shall prove the correctness of the construction described in Section 5. We shall also prove Theorem 24 that states the properties of the construction. We use the same notation that was used in Section 5. In particular, we assume a fixed BCG constraint $c^* = (\mathbf{x}, P)$ where $P = (\mathbf{z}, C)$ is a normalized binary CSP with constraint graph G. We assume that that BCT $P' = (\mathbf{z}', C')$ is the result of the construction from Section 5. We shall show that P' is a TBE of c^*. The construction of C' implies the following property.

▶ **Lemma 26.** *Let $p \in \{1, \ldots, m\}$ be arbitrary and let $t_i \in V_T$ be such that $z_p \in \chi(t)$. Assume that $\sigma' \in sol(P')$ and assume that $(z_p, a), (v_i, \tau) \in \sigma'$. Then $(z_p, a) \in \tau$.*

Proof. Let t_{r_p} be the representative node picked for z_p in step (T2) and consider the path $t_{r_p} = t_{j_1}, t_{j_2}, \ldots, t_{j_k} = t_i$ in T connecting t_{r_p} with t_i. For every $q = 1, \ldots, k$ we have that $z_p \in \chi(t_{r_p}) \cap \chi(t_i)$ and thus $z_p \in \chi(t_{j_q})$ by condition (d3). Denote $\tau_q \in dom(v_{j_q})$ the tuple for which $(v_{j_q}, \tau_q) \in \sigma'$. We shall show by induction on q that $(z_p, a) \in \tau_q$ for every $q = 1, \ldots, k$. Since $\tau_k = \tau$, we then have that $(z_p, a) \in \tau$.

Since $t_1 = t_{r_p}$, we have that $(z_p, a) \in \tau_1$ by constraint c'_p added to C' in step (T2). Assume now that $q > 1$ and consider constraint c'_{j_{q-1},j_q} added in step (T1). The induction hypothesis implies that $(z_p, a) \in \tau_{q-1}$, and by definition of $rel(c'_{j_{q-1},j_q})$ we also have that $(z_p, a) \in \tau_q$. ◀

We are now ready to prove the correctness of the construction.

▶ **Lemma 27.** *$(\mathbf{x}, P')$ is a tree binary encoding of BCG constraint $c^* = (\mathbf{x}, P)$.*

Proof. The constraint graph of P' is a tree that originates from T by adding leaves corresponding to the constraints c'_i added in step (T2). We shall show that $sol(P')[\mathbf{z}] = sol(P)$. Then $rel(c^*) = sol(\mathbf{z}, C)[\mathbf{x}] = sol(\mathbf{z}', C')[\mathbf{x}]$ and the proposition follows.

Assume first that we have a solution $\sigma' \in sol(P')$. Denote $\sigma = \sigma'[\mathbf{z}]$ and let us show that σ satisfies all constraints of P. Let $c \in C$ be a constraint with $scp(c) = \{z_p, z_q\}$. We have $(z_p, a) \in \sigma$ and $(z_q, b) \in \sigma$ for some $a \in dom(z_p)$ and $b \in dom(z_q)$. By condition (d2) we have that $scp(c) \subseteq \chi(t_i)$ for some $t_i \in V_T$. Consider literal $(v_i, \tau) \in \sigma'$. By Lemma 26, we have that $(z_p, a) \in \tau$ and $(z_q, b) \in \tau$. Since $\tau \in dom(v_i)$, we have that $\{(z_p, a), (z_q, b)\} = \tau[scp(c)] \in rel(c)$. Since this holds for every constraint $c \in C$, we get that $\sigma \in sol(P)$.

Assume now that we have a solution $\sigma \in sol(P)$. Let us now define a tuple $\sigma' = \sigma \cup \{(v_i, \sigma[\chi(t_i)]) \mid i = 1, \ldots, N\}$. Since $\sigma \in sol(P)$, we have that $\sigma[\chi(t_i)] \in rel(C_i)$ and thus $\sigma[\chi(t_i)] \in dom(v_i)$. Tuple σ' is thus correctly defined. It also satisfies all constraints (T1) and (T2) and thus $\sigma' \in sol(P')$. ◀

Proof of Theorem 24. Assume that $P' = (\mathbf{z}', C')$ is constructed as above. Then $(\mathbf{x}, P')$ is a tree binary encoding of $c^* = (\mathbf{x}, P)$ by Lemma 27. We may assume by [15] that $|V_T| = O(m)$ and thus $|\mathbf{z}'| = m + |V_T| = O(m)$. For every variable $z_i \in \mathbf{z}$ we have $|dom(z_i)| \leq d$ by assumption. For every variable $v_i \in \mathbf{z}' \setminus \mathbf{z}$ we have that $|\chi(t_i)| \leq tw(G) + 1$ and thus $|dom(v_i)| \leq d^{tw(G)+1}$ by the definition of $dom(v_i)$. ◀