Journal of Artificial Intelligence Research 73 (2022) 535-552 Submitted 08/2020; published 02/2022

Sum-of-Products with Default Values: Algorithms and Complexity Results

Robert Ganian RGANIAN@AC.TUWIEN.AC.AT
Algorithms and Complexity Group, TU Wien, Vienna, Austria

Eun Jung Kim EUN-JUNG.KIM@LAMSADE.DAUPHINE.FR
LAMSADE/CNRS, Université Paris-Dauphin, Paris, France

Friedrich Slivovsky FS@AC.TUWIEN.AC.AT
Stefan Szeider SZ@AC.TUWIEN.AC.AT
Algorithms and Complexity Group, TU Wien, Vienna, Austria

Abstract

Weighted Counting for Constraint Satisfaction with Default Values (#CSPD) is a powerful special case of the sum-of-products problem that admits succinct encodings of #CSP, #SAT, and inference in probabilistic graphical models. We investigate #CSPD under the fundamental parameter of incidence treewidth (i.e., the treewidth of the incidence graph of the constraint hypergraph). We show that if the incidence treewidth is bounded, #CSPD can be solved in polynomial time. More specifically, we show that the problem is fixed-parameter tractable for the combined parameter incidence treewidth, domain size, and support size (the maximum number of non-default tuples in a constraint). This generalizes known results on the fixed-parameter tractability of #CSPD under the combined parameter primal treewidth and domain size. We further prove that the problem is not fixed-parameter tractable if any of the three components is dropped from the parameterization.

1. Introduction

Sum-of-products is a well-studied framework that captures many important tasks (Dechter, 1999; Bacchus, Dalmao, & Pitassi, 2009). Among others, it captures problems such as the counting constraint satisfaction problem (#CSP), the propositional model counting problem (#SAT), and inference problem in probabilistic graphical models (PGMs). Here, we consider a natural formalization of sum-of-products in the terminology of Constraints Satisfaction Problem (CSP): *Weighted Counting for Constraint Satisfaction with Default Values* (#CSPD). #CSPD extends the standard CSP formalism by adding

(i) a rational weight to each tuple in a constraint relation, as well as

(ii) a default weight for each constraint, indicating the weight of assignments not represented by a tuple in the relation.

The weight of an assignment is the product over the weights of all constraints under that assignment, and the value of a #CSPD instance is the sum of these weights taken over all total assignments. For example, an instance of #SAT can be represented by introducing, for each clause, a constraint with default weight 1 containing a single tuple with weight 0. Conditional probability tables of a Bayesian Network (Pearl, 1988) can be directly encoded

as constraints with tuple weights corresponding to conditional probabilities. Additionally, default values can be used to represent uniform probability distributions succinctly.

Canonical algorithms for the sum-of-products problem run in polynomial time for instances of bounded *primal treewidth*, which is the treewidth of the graph whose vertices are variables, and where two variables are adjacent if and only if they appear together in the scope of a constraint (Dechter, 1999; Bacchus et al., 2009; Kask, Dechter, Larrosa, & Dechter, 2005). A runtime bound of this kind also holds for a variable elimination procedure tailored to #CSPD (Capelli, 2016). However, an instance of primal treewidth k may only contain relations of arity up to $k+1$, so one can afford to expand any succinctly represented relation to a table of size $n^{O(k)}$. We, therefore, need a more fine-grained measure than primal treewidth to capture advantages afforded by the use of default values.

Our main contribution is an algorithm, laid out in detail in Section 3, that solves #CSPD in polynomial time for instances of bounded *incidence treewidth* (the treewidth of the bipartite graph on variables and constraints where a variable and a constraint are adjacent if and only if the variable appears in the scope of the constraint).[1] This result is significant since the incidence treewidth is more general than *primal treewidth*: an instance of primal treewidth k has incidence treewidth at most $k+1$, but there are instances of bounded incidence treewidth but arbitrarily large primal treewidth (see, e.g., Samer & Szeider, 2010b).

In the context of CSP and inference in PGMs, efforts toward obtaining even finer-grained measures have led to the development of *generalized hypertree decompositions* (GHDs) (Gottlob et al., 2005) and GHD-based inference algorithms (Kask et al., 2005). Recently, it was shown that the sum-of-products problem can be solved in polynomial time if a measure of GHDs known as the *fractional hypertree width* is bounded (Khamis, Ngo, & Rudra, 2016). This bound requires that factors/constraints are given in a format where each nonzero tuple is represented explicitly. It is unlikely that a similar bound can be obtained for #CSPD because #SAT (and thus #CSPD) is #P-hard already for instances with acyclic constraint hypergraphs (Samer & Szeider, 2010a).

Our algorithm is elementary and combinatorial. It is based on dynamic programming along a tree decomposition, with the key ingredient being a notion of *projection*, which allows us to store the effect of partial assignments locally in dynamic programming tables (Paulusma, Slivovsky, & Szeider, 2016; Slivovsky & Szeider, 2013; Sæther, Telle, & Vatshelle, 2015). The running time of our algorithm for #CSPD is polynomial, where the order of the polynomial depends on the incidence treewidth. In Section 4, we identify additional restrictions under which the algorithm runs in uniform polynomial time, i.e., where the degree of the polynomial does not depend on the incidence treewidth. Problems that such an algorithm can solve are called *fixed-parameter tractable* (Carbonnel & Cooper, 2016; Cygan, Fomin, Kowalik, Lokshtanov, Marx, Pilipczuk, Pilipczuk, & Saurabh, 2015; Downey & Fellows, 1999; Gottlob & Szeider, 2008). More specifically, we show that #CSPD is fixed-parameter tractable for the combined parameter consisting of the incidence treewidth, the domain size, and the

1. Inference in PGMs is known to be tractable for instances whose incidence graph is a tree (Barber, 2012, Ch.5). CSP without counting or weights, where constraints can either be represented by allowed or forbidden tuples, has also be addressed by Cohen, Green, and Houghton (2009) and by Chen and Grohe (2010); the latter work also obtains tractability results for such variants of CSP when the incidence treewidth is bounded.

maximum number of tuples present in a constraint. We also show that none of these three components of the parameter can be dropped without losing fixed-parameter tractability.

2. Preliminaries

In this section, we formalize the sum-of-products problem in terms of a weighted constraint satisfaction problem. Constraints are specified by weighted tuples; a default weight is provided for missing tuples. It is crucial to represent constraints of large arity succinctly. In such cases, we can utilize default values for a succinct representation. We also define the graph invariant treewidth and apply it to weighted constraint satisfaction instances via primal and incidence graphs. Since the treewidth of incidence graphs does not bound the arity of constraints, a succinct representation of constraints is critical in that setting.

2.1 Weighted Constraint Satisfaction with Default Values

Let $\mathcal{V}$ be a set of variables and $\mathcal{D}$ a finite set of values (the domain). A *weighted constraint* C *of arity* ρ *over* $\mathcal{D}$ *with default value* η is a tuple $C = (S, F, f, \eta)$ where

- the *scope* $S = (x_1, \ldots, x_\rho)$ is a sequence[2] of variables from $\mathcal{V}$,
- $\eta \in \mathbb{Q}$ is the *default value*,
- $F \subseteq D^\rho$ is called the *support* and
- $f : F \to \mathbb{Q}$ is a mapping that assigns rational weights to the support.

Here, $\mathbb{Q}$ denotes the set of rational numbers.[3] We define $|C| = |S| + |F| + 1$ and $\mathrm{var}(C) = S$. Since all the weighted constraints we consider will have a default value, we will use *weighted constraint* for brevity instead of *weighted constraint with default value*. On the other hand, a *constraint* is defined analogously as a weighted constraint, but without the components f and η.

An *assignment* $\alpha : X \to \mathcal{D}$ is a mapping defined on a set $X \subseteq \mathcal{V}$ of variables; if $X = \mathcal{V}$ then α is a *total* assignment. An assignment α' then *extends* α if $\forall x \in X : \alpha(x) = \alpha'(x)$. A weighted constraint $C = (S, F, f, \eta)$ naturally induces a total function on assignments of its scope $S = (x_1, \ldots, x_\rho)$: for each assignment $\alpha : X \to \mathcal{D}$ where $X \supseteq S$, we define the *value* $C(\alpha)$ of C under α as $C(\alpha) = f(\alpha(x_1), \ldots, \alpha(x_\rho))$ if $(\alpha(x_1), \ldots, \alpha(x_\rho)) \in F$ and $C(\alpha) = \eta$ otherwise.

An instance $\mathbf{I}$ of #CSPD is a tuple $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ where $\mathcal{V} = \mathrm{var}(\mathbf{I})$ is the set of variables of $\mathbf{I}$, $\mathcal{D}$ is its domain, and $\mathcal{C}$ is a set of weighted constraints over $\mathcal{D}$. We define $|\mathbf{I}|$ as the sum of $|\mathcal{V}|$, $|\mathcal{D}|$, and $|C|$ for each $C \in \mathcal{C}$. The task in #CSPD is to compute the total weight of all assignments of $\mathcal{V}$, i.e., to compute the value

$$\mathsf{sol}(\mathbf{I}) = \sum_{\alpha : \mathcal{V} \to \mathcal{D}} \prod_{C \in \mathcal{C}} C(\alpha).$$

2. We note that even though S is a sequence, we slightly abuse notation by sometimes treating it as a set; for example, we may write $X \subseteq S$.
3. The original definition of #CSPD only considers *nonnegative* rational weights and default values (Brault-Baron, Capelli, & Mengel, 2015). This restriction is not required for the present work.

We observe that every instance of the classical #CSP problem can be straightforwardly translated into an instance of #CSPD: for each constraint in the #CSP instance, we create a weighted constraint, add the tuples of the constraint into F, have f map these to the value 1, and set the default value to 0. Similarly, every instance of #SAT can also be represented as an instance of #CSPD: for each clause, we create a corresponding weighted constraint, set F to be the only tuple that does not satisfy that clause. Let f map this tuple to 0 and set $\eta = 1$. Naturally, #CSPD also generalizes the weighted counting variants for #CSP and #SAT, but is also significantly more powerful than each of these formalisms on their own; indeed, it, for instance, allows us to perform weighted counting for the Mixed CSP problem (Cohen et al., 2009), where each constraint can be represented explicitly or by a clause.

We use standard graph terminology, see Diestel's textbook (2012). The *primal graph* of a #CSPD instance $\mathbf{I}$ is the graph whose vertices correspond to the variables of $\mathbf{I}$ and where two variables a, b are adjacent if and only if there exists a weighted constraint in $\mathbf{I}$ whose scope contains both a and b. The *incidence graph* of $\mathbf{I}$ is the bipartite graph whose vertices correspond to the variables and weighted constraints of $\mathbf{I}$, and where vertices corresponding to a variable x and a weighted constraint C are adjacent if and only if $x \in \mathrm{var}(C)$.

2.2 Treewidth

Let G be a graph. A *tree decomposition* of G is a pair (T, χ) where T is a tree and $\chi : T \to 2^{V(G)}$ is a mapping from tree nodes to subsets of $V(G)$ such that:

- $\forall e = uv \in E(G), \exists t \in V(T) : \{u, v\} \subseteq \chi(t)$, and
- $\forall v \in V(G),\ T[\{t \mid v \in \chi(t)\}]$ is a non-empty connected subtree of T.

We call the vertices of T *nodes* and the sets in $\chi(t)$ *bags* of the tree decomposition (T, χ). The *width* of (T, χ) is equal to $\max\{|\chi(t)| - 1 \mid t \in V(T)\}$. The *treewidth* of G, denoted $\mathsf{tw}(G)$, is the minimum width over all tree decompositions of G. A tree decomposition (T, χ) is called *nice* if T is rooted and the following conditions hold:

- Every node of the tree T has at most two children;
- if a node t has two children t_1 and t_2, then t is called a *join node* and $\chi(t) = \chi(t_1) = \chi(t_2)$;
- if a node t has one child t_1, then either $|\chi(t)| = |\chi(t_1)| + 1$ and $\chi(t_1) \subset \chi(t)$ (in this case we call t an *introduce node*) or $|\chi(t)| = |\chi(t_1)| - 1$ and $\chi(t) \subset \chi(t_1)$ (in this case we call t a *forget node*);
- the root r of T satisfies $\chi(r) = \emptyset$.

It is possible to transform a tree decomposition (T, χ) into a nice tree decomposition (T', χ') of the same width in time $O(|V| + |E|)$ (Kloks, 1994). Furthermore, it is possible to construct near-optimal tree decompositions for graphs of low treewidth efficiently:

Fact 1 (Bodlaender et al., 2016)**.** *There exists an algorithm which, given an n-vertex graph G and an integer k, in time $2^{\mathcal{O}(k)} \cdot n$ either outputs a tree-decomposition of G of width at most $5k + 4$ and $\mathcal{O}(n)$ nodes, or determines that $\mathsf{tw}(G) > k$.*

The primal treewidth (tw) of a #CSPD instance $\mathbf{I}$ is the treewidth of its primal graph, and similarly, the incidence treewidth (tw^*) of $\mathbf{I}$ is the treewidth of its incidence graph.

3. Solving #CSPD Using Incidence Treewidth

Here we show that #CSPD can be solved in polynomial time when restricted to instances of bounded incidence treewidth. We remark that, in parameterized complexity terminology, the algorithm is an XP algorithm. However, before we proceed to the algorithm itself, we will need to introduce *projections*, which are instrumental in defining the records used by our dynamic programming algorithm.

3.1 Projections

Let $C = (S, F)$ be an unweighted constraint where $S = (x_1, \ldots, x_l)$ and let $\tau : X \to \mathcal{D}$ be an assignment. The *projection* of C with respect to assignment τ is the constraint $C|_\tau = (S, F')$, where F' is the set of tuples of F compatible with τ, formally

$$F' = \{\, (s_1, \ldots, s_l) \in F : \tau(x_i) = s_i \text{ for all } x_i \in X \cap S \,\}.$$

The algorithm presented in Section 3.2 groups assignments based on their projections. The key insight is that two assignments τ and σ are indistinguishable for a constraint C if the projections $C|_\tau$ and $C|_\sigma$ are identical. The projection $C|_\tau$ of a weighted constraint $C = (S, F, \eta, f)$ with respect to an assignment τ is simply the projection of its associated unweighted constraint (S, F) with respect to τ.

We write $C[X]$ to denote the set of projections of C with respect to assignments of X. The following observation notes that $C[X]$ is not too large; this contrasts with the fact that the number of assignments of X may be exponential in the size of X.

Observation 1. *Let $C = (S, F)$ be a constraint and let X be a set of variables. The following (in)equalities hold:*

1. $|C[X]| \leq |F| + 1$.
2. $\bigcup_{(S,F') \in C[X]} F' = F$.

Moreover, the union in the second bound is disjoint.

For example, consider a clause $(x \vee y \vee \bar{z})$ of a CNF formula. One possible way to represent this clause in the #CSPD format is $C = ((x, y, z), \{0, 1\}^3 \setminus \{(0, 0, 1)\}, f, 0)$, where $f : s \mapsto 1$ assigns value 1 to each tuple $s \in F$ in the support, and the default value 0 is assigned to the "missing" tuple. If $\alpha : \{x\} \to \{0, 1\}$ is an assignment with $\alpha(x) = 0$, then the projection of C with respect to α is $C|_\alpha = ((x, y, z), \{(0, 0, 0), (0, 1, 0), (0, 1, 1)\})$. In the case where $\alpha(x) = 1$ we get $C|_\alpha = ((x, y, z), \{(1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)\})$.

The projection of a constraint with respect to the union of two assignments can be computed from the projections of this constraint with respect to the individual assignments. We define the *intersection* of two unweighted constraints $C_1 = (S, F_1)$ and $C_2 = (S, F_2)$ with the same scope (which in the following will be projections of the same constraint) as $C_1 \cap C_2 = (S, F_1 \cap F_2)$.

Observation 2. *If C is a constraint and $\tau : X \to \mathcal{D}$, $\sigma : Y \to \mathcal{D}$ are assignments such that $\tau(x) = \sigma(x)$ for each variable $x \in X \cap Y$, then*

1. $(C|_\tau)|_\sigma = (C|_\sigma)|_\tau = C|_{\tau\cup\sigma}$, *and*
2. $C|_\tau \cap C|_\sigma = C|_{\tau\cup\sigma}$.

The value $C(\tau)$ of a constraint under a complete assignment τ can be obtained from the projection $C|_\tau$ in the following way. Let $C = (S, F, \eta, f)$ be a weighted constraint and $B = (S, F')$ a projection of C under an assignment of $X \supseteq S$; note that F' is either empty or contains a single tuple s. We define $val(C, B)$ as $val(C, B) = \eta$ in the former case and $val(C, B) = f(s)$ in the latter case.

Observation 3. *For every assignment $\tau : X \to \mathcal{D}$ and constraint C with scope $S \subseteq X$ we have $val(C, C|_\tau) = C(\tau)$.*

3.2 The Algorithm

For this section, let $\mathbf{I} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ be an arbitrary but fixed instance of #CSPD, and let (T, χ) be a nice tree decomposition of its incidence graph. Let $t \in V(T)$ be a node of this tree decomposition. We refer to a vertex v (variable or constraint) as *forgotten below* t if there is a forget node t' in the subtree rooted at t such that $\chi(t') = \chi(t'') \setminus \{v\}$, where t'' is the child node of t'. We write $X_t = \chi(t) \cap \mathcal{V}$ for the set of variables in the bag of t, Y_t for the set of variables forgotten below t, and $Z_t = X_t \cup Y_t$ for their union. Furthermore, we write $\mathcal{C}_t = \chi(t) \cap \mathcal{C}$ for the set of constraints in the bag of t and $\mathcal{F}_t$ for the set of constraints forgotten below t. Our goal is to compute the weight of assignments $\tau : Z_t \to \mathcal{D}$ restricted to $\mathcal{F}_t$, that is, we want to compute the value of the following expression:

$$\sum_{\tau : Z_t \to \mathcal{D}} \prod_{C \in \mathcal{F}_t} C(\tau). \tag{1}$$

Since every variable and constraint is eventually forgotten, expression (1) computes $\mathsf{sol}(\mathbf{I})$ at the root of T. To perform dynamic programming, we will split the set $\mathcal{D}^{Z_t}$ into equivalence classes that keep track of the influence of assignments on constraints in $\mathcal{C} \setminus \mathcal{F}_t$ (i.e., constraints that have not yet been forgotten). Let $\tau : Z_t \to \mathcal{D}$ be an assignment and let $C \in \mathcal{C} \setminus \mathcal{F}_t$. How can τ affect the constraint C? If $C \notin \mathcal{C}_t$ then $\mathrm{var}(C)$ cannot contain variables forgotten below t since $(T, \mathcal{X})$ is a tree decomposition, so the effect of τ on C can be determined purely in terms of the restricted assignment $\tau|_{X_t}$. On the other hand, if $C \in \mathcal{C}_t$ then the effect of τ on C can be characterized by a projection of C with respect to Z_t (recall that the projection of a weighted constraint is simply the projection of the underlying unweighted constraint). To simplify the presentation of the following arguments, we will assume an ordering on the set of constraints in each bag. Let $\mathcal{C}_t = (C_1, \ldots, C_p)$ be the constraints associated with node t. Let $\sigma \in \mathcal{D}^{X_t}$ be an assignment and let $\vec{B} = (B_1, \ldots, B_p)$ be a vector where each component B_i is a projection of C_i with respect to Z_t, formally $B_i \in C_i[Z_t]$ for each $i \in [p]$. We define $A_t(\sigma, \vec{B})$ as the set of assignments of Z_t compatible with the assignment σ and projections B_i, that is

$$A_t(\sigma, \vec{B}) = \{\, \tau : Z_t \to \mathcal{D} : \tau|_{X_t} = \sigma \text{ and } C_i|_\tau = B_i \text{ for } i \in [p] \,\}.$$

The sets $A_t(\sigma, \vec{B})$ yield a partition of the assignments in $\mathcal{D}^{Z_t}$, since σ varies over all assignments to X_t, the B_i vary over all projections of a constraint under an assignment of Z_t, and the projection of any constraint with respect to an assignment is unique. One can think of the pair $(\sigma, \vec{B})$ as the "state" of the bag $X_t \cup \mathcal{C}_t$ induced by an assignment of Z_t and of $A_t(\sigma, \vec{B})$ as the set of all assignments of Z_t which result in a particular state $(\sigma, \vec{B})$ at node t. For each node $t \in T$ and each pair $(\sigma, \vec{B})$, we will compute and store values $Q_t(\sigma, \vec{B})$ where

$$Q_t(\sigma, \vec{B}) := \sum_{\tau \in A_t(\sigma, \vec{B})} \prod_{C \in \mathcal{F}_t} C(\tau).$$

In the following, we will argue that the values $Q_t(\sigma, \vec{B})$ can be computed from values $Q_{t'}(\sigma', \vec{B}')$ associated with child nodes t' of t. To simplify notation, we may omit the names of nodes in the subscripts for nodes t with a single child node t'. For instance, we will write X instead of X_t, A instead of A_t, and so forth. Moreover, we will use primes when referring to objects associated with t' and write X' instead of $X_{t'}$, A' instead of $A_{t'}$, and so on. Further, for a variable x and an assignment σ whose domain includes x, we let σ_x denote the restriction of σ to x. For each domain value $d \in \mathcal{D}$, we let $\sigma_x^d : \{x\} \to \mathcal{D}$ denote the assignment such that $\sigma_x^d(x) = d$. For a vector $\vec{B} = (B_1, \ldots, B_l)$ of constraints and a single constraint B we will write $(\vec{B}, B) = (B_1, \ldots, B_l, B)$ for their concatenation.

We first consider *variable introduce* nodes, that is, nodes t with a unique child node t' such that $X = X' \cup \{x\}$ for some variable x. Variable x is also included in the set Z, and each assignment $\tau : Z \to \mathcal{D}$ is obtained from an assignment $\tau' : Z' \to \mathcal{D}$ by extending with the singleton assignment $\sigma_x : \{x\} \to \mathcal{D}$, where $\sigma_x(x) = \tau(x)$. If $\tau' \in A'(\sigma', \vec{B}')$, we can simply extend by σ_x^d and take the projection of each $B_i \in \vec{B}'$ with respect to σ_x to obtain the assignment σ and tuple $\vec{B}$ such that $\tau \in A(\sigma, \vec{B})$. Since the new variable x cannot occur in forgotten constraints, the values $Q(\sigma, \vec{B})$ and $Q'(\sigma', \vec{B}')$ coincide.

Lemma 1. *Let t be a variable introduce node with child t', and let x be the variable introduced by t. Further, let $\mathcal{C} = (C_1, \ldots, C_p)$, let $\sigma : X \to \mathcal{D}$ be an assignment, and let $\vec{B} = (B_1, \ldots, B_p)$ be a vector such that $B_i \in C_i[Z]$ for each $i \in [p]$. If $A(\sigma, \vec{B})$ is nonempty, then there is a unique vector $\vec{B}' = (B'_1, \ldots, B'_p)$ satisfying $B_i = B'_i|_{\sigma_x}$ for each $i \in [p]$ such that the mapping $f : \tau \mapsto \tau|_{Z'}$ is a bijection between $A(\sigma, \vec{B})$ and $A'(\sigma', \vec{B}')$, where $\sigma' = \sigma|_{X'}$. Moreover, $Q(\sigma, \vec{B}) = Q'(\sigma', \vec{B}')$ in this case.*

Proof. Suppose $A(\sigma, \vec{B})$ is nonempty and let $\xi \in A(\sigma, \vec{B})$. We let $\xi' = \xi|_{Z'}$ and define $B'_i = C_i|_{\xi'}$ for each i. Since $C_i|_\xi = B_i$ for each i this definition clearly satisfies $B_i = B'_i|_{\sigma_x}$. Let $\tau \in A(\sigma, \vec{B})$ be an assignment and let $\tau' = \tau|_{Z'}$ denote its image under f. It is trivially the case that $\tau' \in A'(\sigma', \vec{C})$, where $\vec{C} = (C_1|_{\tau'}, \ldots, C_p|_{\tau'})$. We argue that $B'_i = C_i|_{\tau'}$ for each $i \in [p]$. Observe that

$$(C_i|_{\tau'})|_{\sigma_x} = C_i|_\tau = B_i = C_i|_\xi = (C_i|_{\xi'})|_{\sigma_x} = B'_i|_{\sigma_x}$$

for $\tau, \xi \in A(\sigma, \vec{B})$. If the projections $B'_i = C_i|_{\xi'}$ and $C_i|_{\tau'}$ are distinct then by Observation 1 they must be disjoint. But since $(C_i|_{\tau'})|_{\sigma_x} = C_i|_{\tau'} \cap C_i|_{\sigma_x}$ and $(C_i|_{\xi'})|_{\sigma_x} = C_i|_{\xi'} \cap C_i|_{\sigma_x}$ by Observation 2, in that case the projections $C_i|_\tau$ and $C_i|_\xi$ would have to be disjoint

as well, a contradiction. We conclude that $C_i|_{\tau'} = B'_i$ and thus $\tau' \in A'(\sigma', B'_i)$. This proves that f is into. Since f is clearly injective, it remains to show that the mapping is surjective as well. Let $\tau' \in A'(\sigma', \vec{B}')$ and let $\tau = \tau' \cup \sigma_x$ so that $f(\tau) = \tau'$. Then $\tau|_X = \sigma$ and $C_i|_\tau = (C_i|_{\tau'})|_{\sigma_x} = B'_i|_{\sigma_x} = B_i$, so $\tau \in A(\sigma, \vec{B})$. We conclude that f is a bijection as claimed. Since $(T, \mathcal{X})$ is a tree decomposition, the newly introduced variable x does not occur in any constraint forgotten below t, so the assignments τ and $f(\tau)$ always have the same weight. It follows that $Q(\sigma, \vec{B}) = Q'(\sigma', \vec{B}')$. □

Next, we consider *constraint introduce* nodes t such that $\mathcal{C} = \mathcal{C}' \cup \{C\}$ for a constraint C. A newly introduced constraint C cannot contain forgotten variables, so its projection with respect to an assignment τ of Z is just the projection with respect to the restriction of τ to X.

Lemma 2. *Let t be an introduce node with child node t' such that $\mathcal{C}' = (C_1, \ldots, C_{p-1})$ and $\mathcal{C} = (C_1, \ldots, C_{p-1}, C)$. Further, let $\sigma : X_t \to \mathcal{D}$ be an assignment, let $\vec{B} = (B_1, \ldots, B_p)$ be a vector of constraints, and let $\vec{B}' = (B_1, \ldots, B_{p-1})$ be the vector consisting of its first $p-1$ components. The following statements hold:*

1. *$Q(\sigma, \vec{B})$ is nonzero only if $B_p = C|_\sigma$.*
2. *If $B_p = C|_\sigma$ then $Q(\sigma, \vec{B}) = Q'(\sigma, \vec{B}')$.*

Proof. We must have $B_p = C|_\sigma$ in order for $Q(\sigma, \vec{B})$ to be nonzero since the newly introduced constraint C cannot contain variables forgotten below t. If $B_p = C|_\sigma$ then it is readily verified that $A(\sigma, \vec{B}) = A'(\sigma, \vec{B}')$. Since $\mathcal{F} = \mathcal{F}'$ the lemma follows. □

A *variable forget* node t satisfies $X = X' \setminus \{x\}$ for some variable x. Upon forgetting a variable x, we sum up the values $Q'(\sigma \cup \sigma^d_x, \vec{B})$ for all possible assignments σ^d_x.

Lemma 3. *Let t be a variable forget node with child t', and let x be the variable forgotten by t. Let $\sigma : X \to \mathcal{D}$ be an assignment and let $\vec{B} = (B_1, \ldots, B_p)$ be a vector of constraints. Then*

$$Q(\sigma, \vec{B}) = \sum_{d \in D} Q'(\sigma \cup \sigma^d_x, \vec{B}).$$

Proof. We show that $A(\sigma, \vec{B}) = \bigcup_{d \in \mathcal{D}} A'(\sigma \cup \sigma^d_x, \vec{B})$. If $\tau \in A(\sigma, \vec{B})$ then $\tau \in A'(\sigma \cup \sigma^{\tau(x)}_x, \vec{B})$. Conversely, if $\tau \in A'(\sigma', \vec{B})$ then $\tau \in A(\sigma, \vec{B})$, where $\sigma = \sigma'|_X$. The lemma now follows since $\mathcal{F} = \mathcal{F}'$ and the union is disjoint. □

For a *constraint forget* node t we have $\mathcal{C} = \mathcal{C}' \setminus \{C\}$ for some constraint C. As C is added to the set of forgotten constraints, we have to include it in our weight calculations for $Q(\sigma, \vec{B})$. Recall that $(\vec{B}, B)$ denotes the vector $(B_1, \ldots, B_k, B)$ that results from adding B as the last component to vector $\vec{B} = (B_1, \ldots, B_k)$.

Lemma 4. *Let t be a constraint forget node with child t' such that $\mathcal{C}' = (C_1, \ldots, C_{p-1}, C)$ and $\mathcal{C} = (C_1, \ldots, C_{p-1})$. Let $\sigma : X \to \mathcal{D}$ be an assignment and let $\vec{B} = (B_1, \ldots, B_{p-1})$ be a vector of constraints. Then*

$$Q(\sigma, \vec{B}) = \sum_{B \in C[Z]} val(C, B)\, Q'(\sigma, (\vec{B}, B)).$$

Proof. We first show that

$$A(\sigma, \vec{B}) = \dot{\bigcup_{B \in C[Z]}} A'(\sigma, (\vec{B}, B)). \tag{2}$$

The inclusion $A(\sigma, \vec{B}) \supseteq \bigcup_{B \in C[Z]} A'(\sigma, (\vec{B}, B))$ is trivial. For the other direction, let $\tau \in A(\sigma, \vec{B})$ and let $B = C|_\tau$. Clearly, $\tau \in A'(\sigma, (\vec{B}, B))$. Moreover, the union is disjoint since the sets $A'(\sigma, \vec{B}')$ are pairwise disjoint.
Let $B \in C[Z]$ be a projection and let $\tau \in A'(\sigma, (\vec{B}, B))$. Since C is forgotten at node t we have

$$\prod_{C' \in \mathcal{F}} C'(\tau) = C(\tau) \prod_{C' \in \mathcal{F}'} C'(\tau). \tag{3}$$

Moreover, $\mathrm{var}(C) \subseteq Z$ since we are forgetting C, so $val(C, B)$ is defined and $val(C, B) = C(\tau)$ by Observation 3. Putting everything together, we get

$$\begin{aligned}
Q(\sigma, \vec{B}) &= \sum_{\tau \in A(\sigma, \vec{B})} \prod_{C' \in \mathcal{F}} C'(\tau) \\
&= \sum_{\tau \in A(\sigma, \vec{B})} C(\tau) \prod_{C' \in \mathcal{F}'} C'(\tau) && \text{by (3)} \\
&= \sum_{B \in C[Z]} \sum_{\tau \in A'(\sigma, (\vec{B}, B))} C(\tau) \prod_{C' \in \mathcal{F}'} C'(\tau) && \text{by (2)} \\
&= \sum_{B \in C[Z]} \sum_{\tau \in A'(\sigma, (\vec{B}, B))} val(C, B) \prod_{C' \in \mathcal{F}'} C'(\tau) && \text{by Observation 3} \\
&= \sum_{B \in C[Z]} val(C, B) \sum_{\tau \in A'(\sigma, (\vec{B}, B))} \prod_{C' \in \mathcal{F}'} C'(\tau) \\
&= \sum_{B \in C[Z]} val(C, B)\, Q'(\sigma, (\vec{B}, B)).
\end{aligned}$$

□

Finally, we deal with *join nodes* t that have child nodes t_1, t_2 such that $\chi(t) = \chi(t_1) = \chi(t_2)$. In keeping with the presentation of the previous lemmas, we will simplify subscripts by writing, for instance, Z_i instead of Z_{t_i}, and A_i instead of A_{t_i}, for $i \in \{1, 2\}$. Further, we use the following notation: given two vectors $\vec{B}_1 = (B_1, \ldots, B_p)$ and $\vec{B}_2 = (B'_1, \ldots, B'_p)$ of constraints such that B_i and B'_i have the same scope for each $i \in [p]$, we write $\vec{B}_1 \cap \vec{B}_2 = (B_1 \cap B'_1, \ldots, B_p \cap B'_p)$ for the vector obtained by taking the componentwise intersections.

Lemma 5. *Let t be a join node with children t_1 and t_2, let $\sigma : X \to \mathcal{D}$ be an assignment, and let $\vec{B} = (B_1, \ldots, B_p)$ be a vector of constraints. We have*

$$Q(\sigma, \vec{B}) = \sum_{\vec{B}_1 \cap \vec{B}_2 = \vec{B}} Q_1(\sigma, \vec{B}_1)\, Q_2(\sigma, \vec{B}_2).$$

Proof. We first show that

$$A(\sigma, \vec{B}) = \{\, \tau \in \mathcal{D}^Z : \tau|_{Z_1} \in A_1(\sigma, \vec{B}_1), \tau|_{Z_2} \in A_2(\sigma, \vec{B}_2), \vec{B} = \vec{B}_1 \cap \vec{B}_2 \,\}. \tag{4}$$

Let $\tau_1 \in A_1(\sigma, \vec{B}_1)$ and $\tau_2 \in A_2(\sigma, \vec{B}_2)$ such that $\vec{B}_1 \cap \vec{B}_2 = \vec{B}$. Since $Y_1 \cap Y_2 = \emptyset$, the combined assignment $\tau = \tau_1 \cup \tau_2$ is well defined. We have $C_i|_\tau = C_i|_{\tau_1 \cup \tau_2} = C_i|_{\tau_1} \cap C_i|_{\tau_2}$ by Observation 2 and thus $C_i|_\tau = B_i$ for each $i \in [p]$, so $\tau \in A(\sigma, \vec{B})$. Conversely, let $\tau \in A(\sigma, \vec{B})$ and let $\tau_1 = \tau|_{Z_1}$ and let $\tau_2 = \tau|_{Z_2}$. Let $\vec{B}_1 = (C_1|_{\tau_1}, \ldots, C_p|_{\tau_1})$ and $\vec{B}_2 = (C_1|_{\tau_2}, \ldots, C_p|_{\tau_2})$. We have $\tau_1 \in A_1(\sigma, \vec{B}_1)$ and $\tau_2 \in A_2(\sigma, \vec{B}_2)$ by construction and $\vec{B} = \vec{B}_1 \cap \vec{B}_2$ by Observation 2.

Each constraint $C \in \mathcal{F}$ forgotten below t is either forgotten below t_1 or below t_2. If $C \in \mathcal{F}_1$ then $\mathrm{var}(C) \cap Y_2 = \emptyset$ by the connectivity properties of a tree decomposition. Conversely, if $C \in \mathcal{F}_2$ then $\mathrm{var}(C) \cap Y_1 = \emptyset$. Therefore for each $\tau \in A(\sigma, \vec{B})$ we get

$$\prod_{C \in \mathcal{F}} C(\tau) = \prod_{C \in \mathcal{F}_1} C(\tau) \prod_{C \in \mathcal{F}_2} C(\tau) = \prod_{C \in \mathcal{F}_1} C(\tau|_{Z_1}) \prod_{C \in \mathcal{F}_2} C(\tau|_{Z_2}), \tag{5}$$

and thus

$$\begin{aligned}
&\sum_{\tau \in A(\sigma, \vec{B})} \prod_{C \in \mathcal{F}} C(\tau) \\
&= \sum_{\substack{\tau_1 \in A_1(\sigma, \vec{B}_1), \\ \tau_2 \in A_2(\sigma, \vec{B}_2), \\ \vec{B}_1 \cap \vec{B}_2 = \vec{B}}} \prod_{C \in \mathcal{F}} C(\tau_1 \cup \tau_2) && \text{by (4)} \\
&= \sum_{\substack{\tau_1 \in A_1(\sigma, \vec{B}_1), \\ \tau_2 \in A_2(\sigma, \vec{B}_2), \\ \vec{B}_1 \cap \vec{B}_2 = \vec{B}}} \prod_{C \in \mathcal{F}_1} C(\tau_1) \prod_{C \in \mathcal{F}_2} C(\tau_2) && \text{by (5)} \\
&= \sum_{\vec{B}_1 \cap \vec{B}_2 = \vec{B}} \sum_{\tau_1 \in A_1(\sigma, \vec{B}_1)} \sum_{\tau_2 \in A_2(\sigma, \vec{B}_2)} \prod_{C \in \mathcal{F}_1} C(\tau_1) \prod_{C \in \mathcal{F}_2} C(\tau_2) \\
&= \sum_{\vec{B}_1 \cap \vec{B}_2 = \vec{B}} \left(\sum_{\tau_1 \in A_1(\sigma, \vec{B}_1)} \prod_{C \in \mathcal{F}_1} C(\tau_1) \right) \left(\sum_{\tau_2 \in A_2(\sigma, \vec{B}_2)} \prod_{C \in \mathcal{F}_2} C(\tau_2) \right) \\
&= \sum_{\vec{B}_1 \cap \vec{B}_2 = \vec{B}} Q_1(\sigma, \vec{B}_1)\, Q_2(\sigma, \vec{B}_2).
\end{aligned}$$

□

Lemma 6. *Let t be a leaf node such that $\mathcal{C}_t = (C_1, \ldots, C_p)$. Let $\sigma : X_t \to \mathcal{D}$ be an assignment and let $\vec{B} = (B_1, \ldots, B_p)$ be a vector of constraints. Then $Q(\sigma, \vec{B}) = 1$ if $B_i = C_i|_\sigma$ for all $i \in [p]$ and $Q(\sigma, \vec{B}) = 0$ otherwise.*

Proof. No variable is forgotten below t, so each set $A(\sigma, \vec{B})$ is either the singleton $\{\sigma\}$ (if $\vec{B}$ is the vector of projections of $\mathcal{C}_t$ with respect to σ) or empty (if the projections do not match). The set $\mathcal{F}$ of forgotten constraints is empty, so $Q(\sigma, \vec{B}) = 1$ in the first case, and $Q(\sigma, \vec{B}) = 0$ in the second case. □

Let $\mathbf{I} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ be an instance of #CSPD and let (T, χ) be a nice tree decomposition of $\mathbf{I}$'s incidence graph. The following algorithm computes values $R_t(\sigma, \vec{B})$—which can be shown to be equivalent to the values $Q_t(\sigma, \vec{B})$—for each tree node t:

1. For each leaf node t with $\mathcal{C}_t = (C_1, \ldots, C_p)$, enumerate the assignments $\sigma \in \mathcal{D}^{X_t}$, compute projections $C_i|_\sigma$ for each $i \in [p]$, and initialize $R_t(\sigma, \vec{B}) = 1$, where $\vec{B} = (C_1|_\sigma, \ldots, C_p|_\sigma)$. Mark t DONE.

2. Do the following until the root $r \in T$ is marked DONE. If $t \in T$ is an unmarked node all of whose children t' are marked DONE, compute the records R_t based on the node type of t:

   (a) If t introduces a variable x, go through all nonzero records $R_{t'}(\sigma', \vec{B}')$. For each assignment $\sigma_x^d = \{x \mapsto d\}$, compute the assignment $\sigma = \sigma' \cup \sigma_x^d$, as well as the vector $\vec{B} = (B_1'|_{\sigma_x^d}, \ldots, B_p'|_{\sigma_x^d})$, and set $R_t(\sigma, \vec{B}) = R_{t'}(\sigma', \vec{B}')$. Mark t DONE.

   (b) If t introduces a constraint C such that $\mathcal{C}_{t'} = (C_1, \ldots, C_p)$ and $\mathcal{C}_t = (C_1, \ldots, C_p, C)$, enumerate the nonzero records $R_{t'}(\sigma', \vec{B}')$ and then set $R_t(\sigma', \vec{B}) = R_{t'}(\sigma', \vec{B}')$, where $\vec{B} = (B_1, \ldots, B_p, C|_{\sigma'})$. Mark t DONE.

   (c) If t is a variable forget node, go through all nonzero records $R_{t'}(\sigma', \vec{B}')$ and add $R_{t'}(\sigma', \vec{B}')$ to the entry $R_t(\sigma'|_{X_t}, \vec{B}')$. If the entry does not exist, create it and initialize with 0. Mark t DONE.

   (d) If t is a forget node such that $\mathcal{C}_{t'} = (C_1, \ldots, C_{p-1}, C)$ and $\mathcal{C}_t = (C_1, \ldots, C_{p-1})$, go through all nonzero records $R_{t'}(\sigma', \vec{B}')$ for $\vec{B}' = (B_1, \ldots, B_{p-1}, B)$ and for each one add the product $val(C, B)R_{t'}(\sigma', \vec{B}')$ to the entry $R_t(\sigma', \vec{B})$, where $\vec{B} = (B_1, \ldots, B_{p-1})$. Again, create and initialize records with 0 whenever necessary. Mark t DONE.

   (e) For a join node t, go through all pairs of nonzero records $R_{t_1}(\sigma, \vec{B}_1)$ and $R_{t_2}(\sigma, \vec{B}_2)$ of its children t_1 and t_2, and add the product $R_{t_1}(\sigma, \vec{B}_1)R_{t_2}(\sigma, \vec{B}_2)$ to the record $R_t(\sigma, \vec{B}_1 \cap \vec{B}_2)$. Create and initialize records with 0 if necessary. Mark t DONE.

3. Once the root is marked DONE, there are two possibilities. If the record $R_r(\varepsilon, ())$ exists, output its value; otherwise, output 0. Here, $\varepsilon : \emptyset \to \mathcal{D}$ denotes the empty assignment and $()$ the empty tuple;

Lemma 7. *The above algorithm outputs* $\mathsf{sol}(\mathbf{I})$.

Proof. We prove that $R_t(\sigma, \vec{B}) = Q_t(\sigma, \vec{B})$ whenever the entry $R_t(\sigma, \vec{B})$ exists, and $Q_t(\sigma, \vec{B}) = 0$ otherwise. For leaf nodes t this is immediate from Lemma 6. Inductively assume the statement holds for the children of a node t.

(a) Let t be a node that introduces variable x. The entry $R_t(\sigma, \vec{B})$ exists if, and only if, there is a record $R_{t'}(\sigma', \vec{B}')$ with $\sigma = \sigma' \cup \sigma_x^d$ and $B_i = B_i'|_{\sigma_x^d}$ for each i. If the entry $R_t(\sigma, \vec{B})$ exists then $R_t(\sigma, \vec{B}) = R_{t'}(\sigma', \vec{B}')$ and by assumption, $R_{t'}(\sigma', \vec{B}') = Q_{t'}(\sigma', \vec{B}')$, so we have $R_t(\sigma, \vec{B}) = Q_t(\sigma, \vec{B})$ by Lemma 1. If the entry does not exist then there is no entry $R_{t'}(\sigma', \vec{B}')$, so $Q_{t'}(\sigma, \vec{B}') = 0$ by assumption and $Q_t(\sigma, \vec{B}) = 0$ by Lemma 1.

(b) Let t be a constraint introduce node with $\mathcal{C}_t = (C_1, \ldots, C_{p-1}, C)$ and $\mathcal{C}_{t'} = (C_1, \ldots, C_{p-1})$. An entry $R_t(\sigma, \vec{B})$ exists if, and only if, there is a record $R_{t'}(\sigma, \vec{B}')$ and $B_p = C|_\sigma$. If the entry exists then $R_t(\sigma, \vec{B}) = R_{t'}(\sigma, \vec{B}')$. By assumption, $R_{t'}(\sigma, \vec{B}') = Q_{t'}(\sigma, \vec{B}')$ and by Lemma 2 $Q_t(\sigma, \vec{B}) = Q_{t'}(\sigma, \vec{B}')$, so $R_t(\sigma, \vec{B}) = Q_t(\sigma, \vec{B})$ as required. If the record does not exist then there is no record $R_{t'}(\sigma, \vec{B}')$ or $B_p \neq C|_\sigma$. In the former case $Q_{t'}(\sigma, \vec{B}') = 0$ by assumption and thus $Q_t(\sigma, \vec{B}) = 0$ by Lemma 2. In the latter case $Q_t(\sigma, \vec{B}) = 0$ by Lemma 2.

(c) Let t be a variable forget node and let x be the variable that is forgotten. A record $R_t(\sigma, \vec{B})$ exists if, and only if, there is a nonzero record $R_{t'}(\sigma \cup \sigma_x^d, \vec{B})$ for some $d \in \mathcal{D}$, and $R_t(\sigma, \vec{B})$ corresponds to their sum in this case. By assumption, $R_{t'}(\sigma \cup \sigma_x^d, \vec{B}) = Q_{t'}(\sigma \cup \sigma_x^d, \vec{B})$ if the record $R_{t'}(\sigma \cup \sigma_x^d, \vec{B})$ exists, and $Q_{t'}(\sigma \cup \sigma_x^d, \vec{B}) = 0$ otherwise. Therefore we have $R_t(\sigma, \vec{B}) = Q_t(\sigma, \vec{B})$ by Lemma 3. If there is no record $R_t(\sigma, \vec{B})$ then there is no record $R_{t'}(\sigma \cup \sigma_x^d, \vec{B})$ and thus $Q_{t'}(\sigma \cup \sigma_x^d, \vec{B}) = 0$ for each $d \in \mathcal{D}$ by assumption. Thus again $Q_t(\sigma, \vec{B}) = 0$ by Lemma 3.

(d) Let t be a forget node such that $\mathcal{C}_t = (C_1, \ldots, C_{p-1})$ and $\mathcal{C}_{t'} = (C_1, \ldots, C_{p-1}, C)$. There is a record $R_t(\sigma, \vec{B})$ if, and only if, there is a nonzero record $R_{t'}(\sigma, (\vec{B}, B))$, and in that case

$$R_t(\sigma, \vec{B}) = \sum_{R_{t'}(\sigma, (\vec{B}, B)) \neq 0} val(C, B) R_{t'}(\sigma, \vec{B}, B)).$$

By assumption we have $R_{t'}(\sigma, (\vec{B}, B)) = Q_{t'}(\sigma, (\vec{B}, B))$ for each such record and otherwise $Q_{t'}(\sigma, (\vec{B}, B)) = 0$, so $R_t(\sigma, \vec{B}) = Q_t(\sigma, \vec{B})$ by Lemma 4. If there is no record $R_t(\sigma, \vec{B})$ then there is no nonzero record $R_{t'}(\sigma, (\vec{B}, B))$ and therefore $Q_{t'}(\sigma, (\vec{B}, B)) = 0$ for all B by assumption. Thus $Q_t(\sigma, \vec{B}) = 0$ by Lemma 4.

(e) Let t be a join node with children t_1 and t_2. The entry $R_t(\sigma, \vec{B})$ exists if, and only if, there is a pair of nonzero records $R_{t_1}(\sigma, \vec{B}_1)$ and $R_{t_2}(\sigma, \vec{B}_2)$ such that $\vec{B}_1 \cap \vec{B}_2 = \vec{B}$. If such a pair exists we have

$$R_t(\sigma, \vec{B}) = \sum_{\substack{R_{t_1}(\sigma, \vec{B}_1) \neq 0, \\ R_{t_2}(\sigma, \vec{B}_2) \neq 0, \\ \vec{B}_1 \cap \vec{B}_2 = \vec{B}}} R_{t_1}(\sigma, \vec{B}_1)\, R_{t_2}(\sigma, \vec{B}_2).$$

By assumption, each term satisfies $R_{t_i}(\sigma, \vec{B}_i) = Q_{t_i}(\sigma, \vec{B}_i)$ for $i \in \{1, 2\}$. Moreover, $R_{t_1}(\sigma, \vec{B}_1) R_{t_2}(\sigma, \vec{B}_2) = 0$ for every pair $R_{t_1}(\sigma, \vec{B}_1), R_{t_2}(\sigma, \vec{B}_2)$ with $\vec{B}_1 \cap \vec{B}_2 = \vec{B}$ that does not appear as a term in the above sum. The equivalence $R_t(\sigma, \vec{B}) = Q_t(\sigma, \vec{B})$ is immediate from Lemma 5. If there is no record $R_t(\sigma, \vec{B})$ there is no pair of nonzero records $R_{t_1}(\sigma, \vec{B}_1), R_{t_2}(\sigma, \vec{B}_2)$ with $\vec{B}_1 \cap \vec{B}_2 = \vec{B}$. Thus, by assumption, $Q_{t_1}(\sigma, \vec{B}_1) = 0$ or $Q_{t_2}(\sigma, \vec{B}_2) = 0$ for each pair $\vec{B}_1, \vec{B}_2$ such that $\vec{B}_1 \cap \vec{B}_2 = \vec{B}$. It again follows from Lemma 5 that $Q_t(\sigma, \vec{B}) = 0$.

We conclude that, once the root node $r \in T$ is marked DONE, we have $R_r(\varepsilon, ()) = Q_r(\varepsilon, ())$ if the record $R_r(\varepsilon, ())$ exists and $Q_r(\varepsilon, ()) = 0$ otherwise. Since $A_r(\varepsilon, ()) = \mathcal{D}^{\mathcal{V}}$ we get $Q_r(\varepsilon, ()) = \mathsf{sol}(\mathbf{I})$ and the output is correct. □

Let sup be the largest size of a support over all constraints in $\mathcal{C}$, let dom denote $|\mathcal{D}|$, and let k be the width of the tree decomposition (T, χ).

Lemma 8. *The runtime of the above algorithm is* $(\mathsf{dom} + \mathsf{sup} + 1)^{\mathcal{O}(k)}|\mathbf{I}|$.

Proof. For each node t we may have entries $R_t(\sigma, \vec{B})$ indexed by pairs $(\sigma, \vec{B})$, where $\sigma \in \mathcal{D}^{|X_t|}$ and $\vec{B} \in C_1[Z_t] \times \cdots \times C_p[Z_t]$. By Observation 1, $|C[Z_t]| \leq \mathsf{sup} + 1$ for any constraint $C \in \mathcal{C}_t$ and thus the number of entries at node t is bounded by $\mathsf{dom}^{|X_t|} \cdot (\mathsf{sup}+1)^{|\mathcal{C}_t|}$. It is not difficult to see that computing the entries for join nodes t is the computationally most demanding step. For each fixed assignment σ of X_t we compute the product of $Q_{t_1}(\sigma, \vec{B}_1)$ and $Q_{t_2}(\sigma, \vec{B}_2)$ and add it to $Q_t(\sigma, \vec{B}_1 \cap \vec{B}_2)$. Therefore, the update at t takes $\mathcal{O}^*(\mathsf{dom}^{|X_t|} \cdot (\mathsf{sup} + 1)^{2|\mathcal{C}_t|})$, where $\mathcal{O}^*()$ suppresses polynomial factors. As the number of tree nodes is $\mathcal{O}(|\mathbf{I}|)$ by Fact 1, the overall running time of the dynamic programming algorithm is $\mathcal{O}^*(\mathsf{dom}^k \cdot (\mathsf{sup} + 1)^{2k})|\mathbf{I}|$ and thus in $(\mathsf{dom} + \mathsf{sup} + 1)^{ck}|\mathbf{I}|$ for large enough c. □

One can compute a nice tree-decomposition of the incidence graph of width at most $5\mathsf{tw}^* + 4$ in time $O(\mathsf{tw}^* \cdot c^{\mathsf{tw}^*}|\mathbf{I}|)$ by running the algorithm of Fact 1 tw^* times. In combination with the preceding lemmas, this proves the main result of this section.

Theorem 1. #CSPD *can be solved in time*

$$(\mathsf{dom} + \mathsf{sup} + 1)^{\mathcal{O}(\mathsf{tw}^*)}|\mathbf{I}|.$$

4. Fixed-Parameter Tractability of #CSPD

We use the framework of Parameterized Complexity (Cygan et al., 2015; Downey & Fellows, 1999, 2013; Flum & Grohe, 2006; Gottlob & Szeider, 2008; Niedermeier, 2006) to provide a fine-grained complexity analysis of the algorithm presented in Subsection 3.2. A parameterized problem $\mathcal{P}$ takes a tuple $(\mathbf{I}, k)$ as an input instance, where $k \in \mathbb{N}$ is called the parameter. A parameterized problem is *fixed-parameter tractable* (FPT in short), *parameterized by* k, if it can be solved by an algorithm which runs in time $f(k) \cdot |\mathbf{I}|^{\mathcal{O}(1)}$ for some computable function f. Algorithms with a running time of this form are called *fixed-parameter algorithms*. On the other hand, an algorithm which solves $\mathcal{P}$ in time $|\mathbf{I}|^{f(k)}$ for some computable function f is called an *XP algorithm*, and parameterized problems which admit such an algorithm are said to belong to the class XP. The complexity class XP properly contains the class FPT. A parameterized problem belongs to the class para-NP if it admits a non-deterministic fixed-parameter algorithm.

In the parameterized complexity perspective, the algorithm of Subsection 3.2 is an XP algorithm for #CSPD parameterized by incidence treewidth. For a tuple σ of parameters, let us denote by #CSPD(σ) the problem #CSPD parameterized by the combined parameter σ. The following is immediate from Theorem 1, which states that #CSPD(tw^*) can be solved in time $|\mathbf{I}|^{O(\mathsf{tw}^*)}$.

Corollary 1. #CSPD(tw^*) *admits an* XP *algorithm.*

Consider the combined parameter $(\mathsf{tw}^*, \mathsf{dom}, \mathsf{sup})$, or simply take the sum of the three as the parameter. It is easy to see that the same analysis of Theorem 1 establishes that with respect to this combined parameter, #CSPD is fixed-parameter tractable.

Corollary 2. *#CSPD(σ) is FPT for the combined parameter $\sigma = (\mathsf{tw}^*, \mathsf{dom}, \mathsf{sup})$.*

Corollary 2 generalizes a result to the effect that #CSPD($\mathsf{tw}, \mathsf{dom}$) is FPT (Capelli, 2016). Before proceeding, we introduce the notion of *parameter domination* (Samer & Szeider, 2010b). Let $\sigma = (p_1, \ldots, p_r)$ and $\sigma' = (p'_1, \ldots, p'_s)$ be two combined parameters. We say that σ *dominates* σ', and write as $\sigma \preceq \sigma'$, if for each $1 \leq i \leq r$ there exists a computable function f that is monotonically increasing in each argument such that for each instance I we have $p_i(I) \leq f(p'_1(I), \ldots, p'_s(I))$. It is not difficult to see that the parameter domination propagates fixed-parameter tractability:

Lemma 9 (Samer & Szeider, 2010b)**.** *Let σ and σ' are two combined parameters such that $\sigma \preceq \sigma'$. If #CSPD(σ) is fixed-parameter tractable, then so is #CSPD(σ').*

Hence, to see that Corollary 2 implies fixed-parameter tractability of #CSPD($\mathsf{tw}, \mathsf{dom}$), we only need to settle the parameter dominance $(\mathsf{tw}^*, \mathsf{dom}, \mathsf{sup}) \preceq (\mathsf{tw}, \mathsf{dom})$. First, it is known that $\mathsf{tw}^* \leq \mathsf{tw} + 1$ (Kolaitis & Vardi, 2000). Second, the maximum arity d of a #CSPD instance provides a lower bound on the primal treewidth tw since any constraint of arity d yields a clique of size d in the primal graph. Therefore we have $d \leq \mathsf{tw} + 1$. Now, we have $\mathsf{sup} \leq \mathsf{dom}^d \leq \mathsf{dom}^{\mathsf{tw}+1}$. Therefore, the parameter domination holds as claimed.

A natural follow-up question to Corollaries 1 and 2 is whether #CSPD is fixed-parameter tractable when we drop some component(s) out of $(\mathsf{tw}^*, \mathsf{dom}, \mathsf{sup})$. To answer this question, we introduce some terminology of parameterized complexity.

An *fpt-reduction* from a parameterized problem $\mathcal{P}$ to a parameterized problem $\mathcal{Q}$ is a fixed-parameter algorithm that maps an instance $(\mathbf{I}, k)$ of $\mathcal{P}$ to an equivalent instance $(\mathbf{I}', k')$ of $\mathcal{Q}$ such that $k' \leq g(k)$ for some computable function g. The notion of fpt-reduction in parameterized complexity plays an analogous role of polynomial-time many-one reduction in classic complexity theory. Under fpt-reduction, a canonical hierarchy of complexity classes is well defined, which is called *W-hierarchy*. Namely, we have

$$\text{FPT} \subseteq \text{W[1]} \subseteq \text{W[2]} \subseteq \cdots \subseteq \text{W[P]} \subseteq \text{XP}.$$

The standard assumption is FPT $\neq$ W[1] and it is known that FPT = W[1] implies the failure of Exponential Time Hypothesis (Chen, Huang, Kanj, & Xia, 2006). Therefore, if a parameterized problem is W[i]-hard (under an fpt-reduction), it is unlikely that the said problem admits a fixed-parameter algorithm.

On the other hand, W[P] $\subseteq$ para-NP holds as well. A classic example of para-NP-complete problem is q-Coloring parameterized by q. One can verify whether a given q-coloring of a graph is proper in (uniform) polynomial time, and thus the problem is in para-NP. It is known that NP-completeness of q-Coloring implies para-NP-completeness. The class para-NP is not contained in XP unless P = NP. We refer the reader to other sources (Downey & Fellows, 1999, 2013; Flum & Grohe, 2006; Cygan et al., 2015) for in-depth treatment of parameterized complexity.

Now, we consider the problem CSPD, the decision version of #CSPD asking whether $\mathsf{sol}(\mathbf{I}) > L$ where L is a part of the input. Clearly, #CSPD is at least as hard as CSPD. The problem CSPD is NP-hard even when $(\mathsf{dom}, \mathsf{sup})$ are bounded by a constant (i.e., the problem is para-NP-hard). We can observe this by encoding 3CNF Satisfiability as CSPD with $\mathsf{dom} = 2$ and $\mathsf{sup} = 1$; a given 3-CNF formula is satisfiable if and only if $\mathsf{sol}(\mathbf{I}) > 0$ for

the corresponding instance $\mathbf{I}$ of CSPD. This implies that CSPD($\mathsf{dom}, \mathsf{sup}$) is para-NP-hard. On the other hand, CSPD($\mathsf{tw}^*, \mathsf{dom}$) generalizes CSP($\mathsf{tw}^*, \mathsf{dom}$) and hence is known to be W[1]-hard (Samer & Szeider, 2010b). This implies W[1]-hardness of CSPD(tw^*) by Lemma 9. The remaining case, the parameterization by ($\mathsf{tw}^*, \mathsf{sup}$), is settled by the next proposition.

Proposition 1. CSPD($\mathsf{tw}^*, \mathsf{sup}$) *is* W[1]*-hard even when all weighted constraints have arity at most* 2 *and* $\mathsf{sup} = 1$.

Proof. We give a reduction from Multicolored Clique, which is well known to be W[1]-hard (Pietrzak, 2003). An instance of Multicolored Clique consists of a graph G whose vertex set is partitioned into k independent sets $V_1, \ldots, V_k$ of the same cardinality. The aim is to decide whether there exists a clique in G of size k; note that such a clique must take a single vertex from each $V_1, \ldots, V_k$.

Given an instance G of Multicolored Clique where each V_i contains n vertices $v_i^1, \ldots, v_i^n$, we construct an instance of CSPD as follows. First, we set $\mathcal{D} = [n]$ and for each vertex subset V_i, $i \in [k]$, we create a variable z_i. Next, for each non-edge $\{v_i^q, v_j^p\}$ with $i < j$, we create the constraint $((z_i, z_j), \{(q, p)\}, \{(q, p) \mapsto 0\}, 1)$. This completes the construction of a CSPD instance $\mathbf{I} = (S, C)$. Since $\mathbf{I}$ has k variables, its incidence treewidth is at most k (Samer & Szeider, 2010b, Lemma 2).

We claim that $\mathsf{sol}(\mathbf{I}) > 0$ if and only if G is a YES-instance. For the forward direction, consider an assignment α such that $\prod_{C \in \mathcal{C}} C(\alpha) \neq 0$. This means that for each $1 \leq i < j \leq n$, none of the constraints whose scope is (z_i, z_j) is evaluated to 0, and in particular $\{v_i^{\alpha(z_i)}, v_j^{\alpha(z_j)}\}$ is not a non-edge in G. Hence $\{v_1^{\alpha(z_1)}, \ldots, v_n^{\alpha(z_n)}\}$ forms a clique of size k in G. For the backward direction, it suffices to reverse the above argument: given a k-clique $\{v_1^{u_i}, \ldots, v_n^{u_n}\}$ in G, the assignment $\alpha(z_i) = u_i$ is easily verified to satisfy $\prod_{C \in \mathcal{C}} C(\alpha) = 1$. Hence the claim holds and the proof is complete. □

5. Concluding Remarks

We have (i) presented an algorithm for #CSPD that runs in polynomial time for instances of bounded incidence treewidth, and (ii) identified additional restrictions that make the problem fixed-parameter tractable, and (iii) shown that none of the restrictions can be dropped without losing fixed-parameter tractability. Our algorithmic result entails tractability for several special cases of #CSPD:

1. Fixed-parameter tractability of CSP parameterized by domain size and primal treewidth (Gottlob, Scarcello, & Sideri, 2002).

2. Fixed-parameter tractability of sum-of-products parameterized by domain size and primal treewidth (Dechter, 1999).

3. Fixed-parameter tractability of #CSPD parameterized by domain size and primal treewidth (Capelli, 2016).

4. Polynomial-time tractability of sum-of-products for instances whose incidence graph is a tree (Barber, 2012).

5. Fixed-parameter tractability of CSP parameterized by domain size, support size, and incidence treewidth (Samer & Szeider, 2010a).

6. Fixed-parameter tractability of #SAT parameterized by incidence treewidth (Fischer, Makowsky, & Ravve, 2008; Samer & Szeider, 2010a).

Moreover, our algorithm can be easily adapted to compute a maximum of sums (rather than a sum of products) and deal with valued constraint satisfaction problems (VCSP) (Cohen, Cooper, Jeavons, & Krokhin, 2006).

Tractability of #CSPD for instances with β-acyclic constraint hypergraphs was shown through an intricate variable elimination algorithm (Brault-Baron et al., 2015). This procedure naturally gives rise to a width parameter called the *cover-width* (Capelli, 2016). There are currently no efficient algorithms for computing this parameter. Whether bounds on the incidence treewidth can be translated into bounds on the cover-width (thus relating our dynamic programming algorithm to variable elimination) is an intriguing open question.

Acknowledgments

This research was supported by Austrian Science Fund (FWF), via grants P27721 and P31336.

References

Bacchus, F., Dalmao, S., & Pitassi, T. (2009). Solving #SAT and Bayesian inference with backtracking search. *J. Artif. Intell. Res.*, *34*, 391–442.

Barber, D. (2012). *Bayesian reasoning and machine learning*. Cambridge University Press.

Bodlaender, H. L., Drange, P. G., Dregi, M. S., Fomin, F. V., Lokshtanov, D., & Pilipczuk, M. (2016). A $c^k n$ 5-approximation algorithm for treewidth. *SIAM J. Comput.*, *45*(2), 317–378.

Brault-Baron, J., Capelli, F., & Mengel, S. (2015). Understanding model counting for beta-acyclic CNF-formulas. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, Vol. 30 of *LIPIcs*, pp. 143–156. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

Capelli, F. (2016). *Structural restrictions of CNF-formulas: applications to model counting and knowledge compilation*. Ph.D. thesis, Université Paris Diderot.

Carbonnel, C., & Cooper, M. C. (2016). Tractability in constraint satisfaction problems: a survey. *Constraints*, *21*(2), 115–144.

Chen, H., & Grohe, M. (2010). Constraint satisfaction with succinctly specified relations. *J. of Computer and System Sciences*, *76*(8), 847–860.

Chen, J., Huang, X., Kanj, I. A., & Xia, G. (2006). Strong computational lower bounds via parameterized complexity. *J. of Computer and System Sciences*, *72*(8), 1346–1367.

Cohen, D. A., Cooper, M. C., Jeavons, P., & Krokhin, A. A. (2006). The complexity of soft constraint satisfaction. *Artif. Intell.*, *170*(11), 983–1016.

Cohen, D. A., Green, M. J., & Houghton, C. (2009). Constraint representations and structural tractability. In Gent, I. P. (Ed.), *Principles and Practice of Constraint Programming - CP 2009*, Vol. 5732 of *Lecture Notes in Computer Science*, pp. 289–303. Springer Verlag.

Cygan, M., Fomin, F. V., Kowalik, L. u., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., & Saurabh, S. (2015). *Parameterized algorithms*. Springer, Cham.

Dechter, R. (1999). Bucket elimination: a unifying framework for reasoning. *Artificial Intelligence*, *113*(1-2), 41–85.

Diestel, R. (2012). *Graph Theory, 4th Edition*, Vol. 173 of *Graduate texts in mathematics*. Springer.

Downey, R. G., & Fellows, M. R. (1999). *Parameterized Complexity*. Monographs in Computer Science. Springer Verlag, New York.

Downey, R. G., & Fellows, M. R. (2013). *Fundamentals of parameterized complexity*. Texts in Computer Science. Springer Verlag.

Fischer, E., Makowsky, J. A., & Ravve, E. R. (2008). Counting truth assignments of formulas of bounded tree-width or clique-width.. *Discr. Appl. Math.*, *156*(4), 511–529.

Flum, J., & Grohe, M. (2006). *Parameterized Complexity Theory*, Vol. XIV of *Texts in Theoretical Computer Science. An EATCS Series*. Springer Verlag, Berlin.

Gottlob, G., Grohe, M., Musliu, N., Samer, M., & Scarcello, F. (2005). Hypertree decompositions: Structure, algorithms, and applications. In Kratsch, D. (Ed.), *Proceedings of the 31st International Workshop on Graph-Theoretic Concepts in Computer Science (WG'05)*, Vol. 3787 of *Lecture Notes in Computer Science*, pp. 1–15. Springer Verlag.

Gottlob, G., Scarcello, F., & Sideri, M. (2002). Fixed-parameter complexity in AI and nonmonotonic reasoning. *Artificial Intelligence*, *138*(1-2), 55–86.

Gottlob, G., & Szeider, S. (2008). Fixed-parameter algorithms for artificial intelligence, constraint satisfaction, and database problems. *The Computer Journal*, *51*(3), 303–325. Survey paper.

Kask, K., Dechter, R., Larrosa, J., & Dechter, A. (2005). Unifying tree decompositions for reasoning in graphical models. *Artificial Intelligence*, *166*(1-2), 165–193.

Khamis, M. A., Ngo, H. Q., & Rudra, A. (2016). FAQ: questions asked frequently. In Milo, T., & Tan, W. (Eds.), *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems - PODS 2016*, pp. 13–28. Assoc. Comput. Mach., New York.

Kloks, T. (1994). *Treewidth: Computations and Approximations*. Springer Verlag, Berlin.

Kolaitis, P. G., & Vardi, M. Y. (2000). Conjunctive-query containment and constraint satisfaction. *J. of Computer and System Sciences*, *61*(2), 302–332.

Niedermeier, R. (2006). *Invitation to fixed-parameter algorithms*. Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, Oxford.

Paulusma, D., Slivovsky, F., & Szeider, S. (2016). Model counting for CNF formulas of bounded modular treewidth. *Algorithmica*, *76*(1), 168–194.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. The Morgan Kaufmann Series in Representation and Reasoning. Morgan Kaufmann, San Mateo, CA.

Pietrzak, K. (2003). On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *J. of Computer and System Sciences*, *67*(4), 757–771.

Sæther, S. H., Telle, J. A., & Vatshelle, M. (2015). Solving #SAT and MAXSAT by dynamic programming. *J. Artif. Intell. Res.*, *54*, 59–82.

Samer, M., & Szeider, S. (2010a). Algorithms for propositional model counting. *J. Discrete Algorithms*, *8*(1), 50–64.

Samer, M., & Szeider, S. (2010b). Constraint satisfaction with bounded treewidth revisited. *J. of Computer and System Sciences*, *76*(2), 103–114.

Slivovsky, F., & Szeider, S. (2013). Model counting for formulas of bounded clique-width. In Cai, L., Cheng, S., & Lam, T. W. (Eds.), *Algorithms and Computation - 24th International Symposium, ISAAC 2013*, Vol. 8283 of *Lecture Notes in Computer Science*, pp. 677–687. Springer Verlag.