Proceedings Paper:
Ganian, R., Ordyniak, S. orcid.org/0000-0003-1935-651X and Szeider, S. (2019) A join-based hybrid parameter for constraint satisfaction. In: Schiex, T. and de Givry, S., (eds.) Principles and Practice of Constraint Programming. 25th International Conference, CP 2019, 30 Sep - 04 Oct 2019, Stamford, CT, USA. Springer , pp. 195-212. ISBN 9783030300470

https://doi.org/10.1007/978-3-030-30048-7_12

A Join-Based Hybrid Parameter for Constraint Satisfaction*

Robert Ganian[1], Sebastian Ordyniak[2], and Stefan Szeider[1]

[1] Algorithms and Complexity group, TU Wien, Vienna, Austria
[2] Algorithms group, University of Sheffield, Sheffield, UK

Abstract. We propose *joinwidth*, a new complexity parameter for the Constraint Satisfaction Problem (CSP). The definition of joinwidth is based on the arrangement of basic operations on relations (joins, projections, and pruning), which inherently reflects the steps required to solve the instance. We use joinwidth to obtain polynomial-time algorithms (if a corresponding decomposition is provided in the input) as well as fixed-parameter algorithms (if no such decomposition is provided) for solving the CSP.
Joinwidth is a *hybrid* parameter, as it takes both the graphical structure as well as the constraint relations that appear in the instance into account. It has, therefore, the potential to capture larger classes of tractable instances than purely *structural* parameters like hypertree width and the more general fractional hypertree width (fhtw). Indeed, we show that any class of instances of bounded fhtw also has bounded joinwidth, and that there exist classes of instances of bounded joinwidth and unbounded fhtw, so bounded joinwidth properly generalizes bounded fhtw.
We further show that bounded joinwidth also properly generalizes several other known hybrid restrictions, such as fhtw with degree constraints and functional dependencies. In this sense, bounded joinwidth can be seen as a unifying principle that explains the tractability of several seemingly unrelated classes of CSP instances.

1 Introduction

The Constraint Satisfaction Problem (CSP) is a central and generic computational problem that provides a common framework for many theoretical and practical applications in AI and other areas of Computer Science [34]. An instance of the CSP consists of a collection of variables that must be assigned values subject to constraints, where each constraint is given in terms of a relation whose tuples specify the allowed combinations of values for specified variables.

CSP is NP-complete in general. A central line of research is concerned with the identification of classes of instances for which the CSP can be solved in polynomial time. The two main approaches are to define classes either in terms of the constraint relations that may occur in the instance (*syntactic restrictions*; see, e.g., [4]), or in terms of the constraint hypergraph associated with the instance (*structural restrictions*; see, e.g., [20]). There are also several prominent proposals for utilizing simultaneously syntactic and structural restrictions called *hybrid restrictions* (see, e.g., [30,7,8,6]).

Grohe and Marx [22] showed that CSP is polynomial-time tractable whenever the constraint hypergraph has bounded *fractional hypertree width*, which strictly generalizes previous tractability results based on hypertree width [17] and acyclic queries [36]. Bounded fractional hypertree width is the most general known structural restriction that gives rise to polynomial-time tractability of CSP.

As bounded fractional hypertree width is a structural restriction that is completely oblivious to the relations present in the instance, it is natural to expect that one can modify fractional hypertree width to take the shape of relations into account. This is indeed possible, but does not lead to a compact notion that is well-suited for further theoretical analysis.

Our contribution: Joinwidth. We propose a new hybrid restriction for the CSP, the width parameter *joinwidth*, which is based on the arrangement of basic relational operations along a tree, and not on hypertree decompositions. Interestingly, as we will show, our notion strictly generalizes (i) bounded fractional hypertree width, (ii) recently introduced extensions of fractional hypertree width with degree constraints and functional dependencies [26], (iii) various prominent hybrid restrictions [5], as well as (iv) tractable classes based on *functionality* and *root sets* [10,9,5]. Hence, joinwidth gives rise to a common framework that captures several different tractable classes considered in the past. Moreover, none of the other hybrid parameters that we are aware of [8], such as classes based on the Broken Triangle Property or topological

* Robert Ganian acknowledges support by the Austrian Science Fund (FWF, Project P31336) and is also affiliated with FI MUNI, Czech Republic.

minors [7,6] and directional rank [30], generalize fractional hypertree width and hence all of them are either less general or orthogonal to joinwidth.

Joinwidth is based on the arrangement of the constraints on the leaves of a rooted binary tree which we call a *join decomposition*. The join decomposition indicates the order in which relational joins are formed, where one proceeds in a bottom-up fashion from the leaves to the root, labeling a node by the join of the relations at its children, and projecting away variables that do not occur in relations to be processed later. Join decompositions are related to (structural) *branch decompositions* of hypergraphs, where the hyperedges are arranged on the leaves of the tree [2,21,32]. Related notions have been considered in the context of query optimization [1,24]. However, the basic form of join decompositions using only relational joins and projections is still a weak notion that cannot be used to tackle instances of bounded fractional hypertree width efficiently. We identify a further operation that—in conjunction with relational joins and projections—gives rise to the powerful new concept of joinwidth that captures and extends the various known tractable classes mentioned. This third operation *prunes* away all the tuples from an intermediate relation that are inconsistent with a relation to be processed later.

A join decomposition of a CSP instance specifies the order in which the above three operations are applied, and its *width* is the smallest real number w such that each relation appearing within the join decomposition has at most m^w many tuples (where m is the maximum number of tuples appearing in any constraint relation of the CSP instance under consideration). The joinwidth of a CSP instance is the smallest width over all its join decompositions. Observe that joinwidth is a hybrid parameter—it depends on both the graphical structure as well as the constraint relations appearing in the instance.

Exploiting Joinwidth. Similarly to other width parameters, also the property that a class of CSP instances has bounded joinwidth can only be exploited for CSP solving if a decomposition (in our case a join decomposition) witnessing the bounded width is provided as part of the input. While such a join decomposition can be computed efficiently from a fractional hypertree decomposition or when the CSP instance belongs to a tractable class based on functionality or root sets mentioned earlier, we show that computing an optimal join decomposition is NP-hard in general, mirroring the corresponding NP-hardness of computing optimal fractional hypertree decompositions [15].

However, this obstacle disappears if we move from the viewpoint of polynomial-time tractability to *fixed-parameter tractability* (FPT). Under the FPT viewpoint, one considers classes of instances $\mathbf{I}$ that can be solved by a fixed-parameter algorithm—an algorithm running in time $f(k)|\mathbf{I}|^{O(1)}$, where k is the parameter (typically the number of variables or constraints), $|\mathbf{I}|$ is the size of the instance, and f is a computable function [16,18,19]. We note that it is natural to assume that k is much smaller than $|\mathbf{I}|$ in typical cases. The use of fixed-parameter tractability is well motivated in the CSP setting; see, for instance, Marx's discussion on this topic [29].

Here, we obtain two single-exponential fixed-parameter algorithms for instances of bounded joinwidth (i.e., algorithms with a running time of $2^{\mathcal{O}(k)} \cdot |\mathbf{I}|^{\mathcal{O}(1)}$): one where k is the number of variables, and the other when k is the number of constraints. In this setting, we do not require an associated join decomposition to be provided with the input.

Under the FPT viewpoint, Marx [29] previously introduced the structural parameter *submodular width* (bounded submodular width is equivalent to bounded *adaptive width* [28]), which is strictly more general than fractional hypertree width, but when bounded only gives rise to fixed-parameter tractability and not polynomial-time tractability of CSP. In fact, Marx showed that assuming the Exponential Time Hypothesis [23], bounded submodular width is the most general purely structural restriction that yields fixed-parameter tractability for CSP. However, as joinwidth is a hybrid parameter, it can (and we show that it does) remain bounded even on instances of unbounded submodular width—and the same holds also for the recently introduced extensions of submodular width based on functional dependencies and degree bounds [26].

Roadmap. After presenting the required preliminaries on (hyper-)graphs, CSP, and fractional hypertree width in Section 2, we introduce and motivate join decompositions and joinwidth in Section 3. We establish some fundamental properties of join decompositions, provide our tractability result for CSP for the case when a join decomposition is given as part of the input, and then obtain our NP-hardness result for computing join decompositions of constant width. Section 4 provides an in-depth justification for the various design choices underlying join decompositions; among others, we show that the pruning step is required if the aim is to generalize fractional hypertree width. Our algorithmic applications for joinwidth are presented in Section 5: for instance, we show that joinwidth generalizes fractional hypertree width, but also other known (and hybrid) parameters such as functionality, root sets, and Turan sets. Section 6 contains our fixed-parameter tractability results for classes of CSP instances with bounded joinwidth. Finally, in Section 7, we compare the algorithmic power of joinwidth to the power of algorithms which rely on the unrestricted use of join and projection operations.

2 Preliminaries

We will use standard graph terminology [12]. An *undirected graph* G is a pair (V, E), where V or $V(G)$ is the vertex set and E or $E(G)$ is the edge set. All our graphs are simple and loopless. For a tree T we use $L(T)$ to denote the set of its leaves. For $i \in \mathbb{N}$, we let $[i] = \{1, \dots, i\}$.

2.1 Hypergraphs, Branchwidth and Treewidth

Similarly to graphs, a *hypergraph* H is a pair (V, E) where V or $V(H)$ is its vertex set and E or $E(H) \subseteq 2^V$ is its set of hyperedges. We denote by $H[V']$ the hypergraph *induced* on the vertices in $V' \subseteq V$, i.e., the hypergraph with vertex set V' and edge set $\{\, e \cap V' : e \in E \,\}$. Every subset F of $E(H)$ defines a *cut* of H, i.e., the pair $(F, E(H) \setminus F)$. We denote by $\delta_H(F)$ (or just $\delta(F)$ if H is clear from the context) the set of *cut vertices* of F in H, i.e., $\delta(F)$ contains all vertices incident to both an edge in F and an edge in $E(H) \setminus F$. Note that $\delta(F) = \delta(E(H) \setminus F)$.

Let H be a hypergraph. A *branch-decomposition* of H is a pair $\mathcal{B} = (B, \beta)$, where B is a rooted binary tree and $\beta : L(B) \to E(H)$ is a bijection between the leaves $L(B)$ of B and the edges of H. For simplicity, we write $\beta(B')$ to denote the set $\{\, \beta(l) : l \in L(B') \,\}$ of edges for a subtree B' of B. The *branchwidth* of an edge e of B, denoted by $\mathsf{bw}(e)$, is equal to $|\delta_H(\beta(B'))|$, where B' is any of the two components of $B - e$. The *branchwidth* of $\mathcal{B}$ is equal to $\max_{e \in E(B)} \mathsf{bw}(e)$ and the branchwidth of H, denoted by $\mathsf{bw}(H)$, is the minimum branchwidth of any of its branch-decompositions. We say that $\mathcal{B}$ is a *linear branch decomposition* if every inner node of B is adjacent to at least one leaf node and define the *linear branchwidth* of H, denoted by $\mathsf{lbw}(H)$, as the minimum branchwidth over all linear branch decompositions of H.

A *tree-decomposition* of H is a pair $\mathcal{T} = (T, (B_t)_{t \in V(T)})$, where T is a tree and $B_t \subseteq V(H)$ for every $t \in V(T)$ such that: (1) for every $e \in E(H)$ there is a $t \in V(T)$ such that $e \subseteq B_t$ and (2) for every $v \in V(H)$ the set $\{\, t \in V(T) : v \in B_t \,\}$ induces a non-empty subtree of T. The *treewidth* of $\mathcal{T}$ is equal to $\max_{t \in V(T)}(|B_t| - 1)$, and the *treewidth* of H is the minimum treewidth of any tree-decomposition of H. We say that $\mathcal{T}$ is a *path-decomposition* if T is a path and define the *pathwidth* of H as the minimum treewidth of any path-decomposition of H.

2.2 The Constraint Satisfaction Problem

Let D be a set and n and n' be natural numbers. An n-ary relation on D is a subset of D^n. For a tuple $t \in D^n$, we denote by $t[i]$, the i-th entry of t, where $1 \leq i \leq n$. For two tuples $t \in D^n$ and $t' \in D^{n'}$, we denote by $t \circ t'$, the concatenation of t and t'.

An instance of a *constraint satisfaction problem* (CSP) $\mathbf{I}$ is a triple $\langle V, D, C \rangle$, where V is a finite set of variables over a finite set (domain) D, and C is a set of constraints. A *constraint* $c \in C$ consists of a *scope*, denoted by $S(c)$, which is a completely ordered subset of V, and a relation, denoted by $R(c)$, which is a $|S(c)|$-ary relation on D. We let $|c|$ denote the number of tuples in $R(c)$ and $|\mathbf{I}| = |V| + |D| + \sum_{c \in C} |c|$. Without loss of generality, we assume that each variable occurs in the scope of at least one constraint.

A *solution* for $\mathbf{I}$ is an assignment $\theta : V \to D$ of the variables in V to domain values (from D) such that for every constraint $c \in C$ with scope $S(c) = (v_1, \dots, v_{|S(c)|})$, the relation R contains the tuple $\theta(S(c)) = (\theta(v_1), \dots, \theta(v_{|S(c)|}))$. We denote by SOL($\mathbf{I}$) the constraint containing all solutions of $\mathbf{I}$, i.e., the constraint with scope $V = \{v_1, \dots, v_n\}$, whose relation contains one tuple $(\theta(v_1), \dots, \theta(v_n))$ for every solution θ of $\mathbf{I}$. The task in CSP is to decide whether the instance $\mathbf{I}$ has at least one solution or in other words whether SOL($\mathbf{I}$) $\neq \emptyset$. Here and in the following we will for convenience (and with a slight abuse of notation) sometimes treat constraints like sets of tuples.

For a variable $v \in S(c)$ and a tuple $t \in R(c)$, we denote by $t[v]$, the i-th entry of t, where i is the position of v in $S(c)$. Let V' be a subset of V and let V'' be all the variables that appear in V' and $S(c)$. With a slight abuse of notation, we denote by $S(c) \cap V'$, the sequence $S(c)$ restricted to the variables in V' and we denote by $t[V']$ the tuple $(t[v_1], \dots, t[v_{|V''|}])$, where $S(c) \cap V' = (v_1, \dots, v_{|V''|})$.

Let c and c' be two constraints of $\mathbf{I}$. We denote by $S(c) \cup S(c')$, the ordered set (i.e., tuple) $S(c) \circ (S(c') \setminus S(c))$. The *(natural) join* between c and c', denoted by $c \bowtie c'$, is the constraint with scope $S(c) \cup S(c')$ containing all tuples $t \circ t'[S(c') \setminus S(c)]$ such that $t \in R(c)$, $t' \in R(c')$, and $t[S(c) \cap S(c')] = t'[S(c) \cap S(c')]$. The *projection* of c to V', denoted by $\pi_{V'}(c)$, is the constraint with scope $S(c) \cap V'$, whose relation contains all tuples $t[V']$ with $t \in R(c)$. We note that if c contains at least one tuple, then projecting it onto a set V' with $V' \cap S(c) = \emptyset$ results in the constraint with an empty scope and a relation containing the empty tuple (i.e., a tautological constraint). On the other hand, if

$R(c)$ is the relation containing the empty tuple, then every projection of c will also result in a relation containing the empty tuple.

For a CSP instance $\mathbf{I} = \langle V, D, C\rangle$ we sometimes denote by $V(\mathbf{I})$, $D(\mathbf{I})$, $C(\mathbf{I})$, and $\sharp_{tup}(\mathbf{I})$ its set of variables V, its domain D, its set of constraints C, and the maximum number of tuples in any constraint relation of $\mathbf{I}$, respectively. For a subset $V' \subseteq V$, we will also use $\mathbf{I}[V']$ to denote the sub-instance of $\mathbf{I}$ induced by the variables in $V' \subseteq V$, i.e., $\mathbf{I}[V'] = \langle V', D, \{\, \pi_{V'}(c) : c \in C \,\}\rangle$. The *hypergraph* $H(\mathbf{I})$ of a CSP instance $\mathbf{I} = \langle V, D, C\rangle$ is the hypergraph with vertex set V and edge set $\{\, S(c) : c \in C \,\}$.

It is well known that for every instance $\mathbf{I}$ and every instance $\mathbf{I}'$ obtained by either (1) replacing two constraints in $C(\mathbf{I})$ by their natural join or (2) adding a projection of a constraint in $C(\mathbf{I})$, it holds that $\text{SOL}(\mathbf{I}) = \text{SOL}(\mathbf{I}')$. As a consequence, $\text{SOL}(\mathbf{I})$ can be computed by performing, e.g., a sequence of joins over all the constraints in C.

2.3 Fractional Hypertree Width

Let H be a hypergraph. A *fractional edge cover* for H is a mapping $\gamma : E(H) \to \mathbb{R}$ such that $\sum_{e \in E(H) \wedge v \in e} \gamma(e) \geq 1$ for every $v \in V(H)$. The *weight* of γ, denoted by $w(\gamma)$, is the number $\sum_{e \in E(H)} \gamma(e)$. The *fractional edge cover number* of H, denoted by $\mathsf{fec}(H)$, is the smallest weight of any fractional edge cover of H.

A *fractional hypertree decomposition* $\mathcal{T}$ of H is a triple $\mathcal{T} = (T, (B_t)_{t \in V(T)}, (\gamma_t)_{t \in V(T)})$, where $(T, (B_t)_{t \in V(T)})$ is a tree decomposition [33,13] of H and $(\gamma_t)_{t \in V(T)}$ is a family of mappings from $E(H)$ to $\mathbb{R}$ such that for every $t \in V(T)$, it holds that γ_t is a fractional edge cover for $H[B_t]$. We call the sets B_t the *bags* and the mappings γ_t the *fractional guards* of the decomposition. The *width* of $\mathcal{T}$ is the maximum $w(\gamma_t)$ over all $t \in V(T)$. The *fractional hypertree width* of H, denoted by $\mathsf{fhtw}(H)$, is the minimum width of any fractional hypertree decomposition of H. Finally, the *fractional hypertree width* of a CSP instance $\mathbf{I}$, denoted by $\mathsf{fhtw}(\mathbf{I})$, is equal to $\mathsf{fhtw}(H(\mathbf{I}))$.

Proposition 1 *Let $\mathbf{I}$ be a CSP instance with hypergraph H and let $\mathcal{T} = (T, (B_t)_{t \in V(T)}, (\gamma_t)_{t \in V(T)})$ be a fractional hypertree decomposition of H of width at most ω. For every node $t \in V(T)$ and every subset $B \subseteq B_t$, it holds that $|\text{SOL}(\mathbf{I}[B])| \leq (\sharp_{\text{tup}}(\mathbf{I}))^{\omega}$.*

Proof. It follows from the definition of fractional hypertree width that γ_t is a fractional edge cover for $H[B_t]$ of width at most ω for every $t \in V(T)$. Since fractional edge covers are heriditary, γ_t is also a fractional edge cover for $H[B]$ of width at most ω for every $B \subseteq B_t$. Moreover, it is shown in [22, Lemma 3], that any CSP instance $\mathbf{I}'$ has at most $(\sharp_{tup}(\mathbf{I}'))^{\mathsf{fec}(H(\mathbf{I}'))}$ solutions; note that [22, Lemma 3] actually only states that $\mathbf{I}'$ has at most $(|\mathbf{I}'|)^{\mathsf{fec}(H(\mathbf{I}'))}$ solutions, however, the slightly stronger bound of $(\sharp_{tup}(\mathbf{I}'))^{\mathsf{fec}(H(\mathbf{I}'))}$ follows immediately from the proof of [22, Lemma 3]. Hence, because $\mathsf{fec}(\mathbf{I}[B]) \leq \omega$, we obtain that $\mathbf{I}[B]$ has at most $(\sharp_{tup}(\mathbf{I}))^{\omega}$ solutions. □

3 Join Decompositions and Joinwidth

This section introduces two notions that are central to our contribution: *join decompositions* and *joinwidth*. In the following, let us consider an arbitrary CSP instance $\mathbf{I} = \langle V, D, C\rangle$.

Definition 2 *A* join decomposition *for $\mathbf{I}$ is a pair (J, ϱ), where J is a rooted binary tree and ϱ is a bijection between the leaves $L(J)$ of J and C.*

Let j be a node of J. We denote by J_j the subtree of J rooted at j and we denote by $X(j)$, $V(j)$, $\overline{V}(j)$, and $S(j)$ the (unordered) sets $\{\, \varrho(\ell) : \ell \in L(J_j) \,\}$, $\bigcup_{c \in X(j)} S(c)$, $\bigcup_{c \notin X(j)} S(c)$, and $V(j) \cap \overline{V}(j)$, respectively; infuitively, $X(j)$ is the set of constraints that occur in the subtree rooted at j, $V(j)$ is the set of variables that occur in the scope of constraints in $X(j)$, $\overline{V}(j)$ is the set of variables that occur in the scope of constraints not in $X(j)$, and $S(j)$ is the set of variables that occur in $V(j)$ and $\overline{V}(j)$. In some cases, we will also consider *linear join decompositions*, which are join decompositions where every inner node is adjacent to at least one leaf.

Semantics of Join Decompositions. Intuitively, every internal node of a join decomposition represents a join operation that is carried out over the constraints obtained for the two children; in this way, a join decomposition can be seen as a procedure for performing joins, with the aim of determining whether $\text{SOL}(\mathbf{I})$ is non-empty (i.e., solving the CSP instance $\mathbf{I}$). Crucially, the running time of such a procedure depends on the size of the constraints obtained and stored by the algorithm which performs such joins. The aim of this subsection is to formally define and substantiate an algorithmic procedure which uses join decompositions to solve CSP.

The Naive Approach. A naive way of implementing the above idea would be to simply compute and store the natural join at each node of the join decomposition and proceed up to the root; see for instance the work of [3]. Formally, we can recursively define a constraint $C_{naive}(j)$ for every node $j \in V(J)$ as follows. If j is a leaf, then $C_{naive}(j) = \varrho(j)$. Otherwise $C_{naive}(j)$ is equal to $C_{naive}(j_1) \bowtie C_{naive}(j_2)$, where j_1 and j_2 are the two children of j in J. It is easy to see that this approach can create large constraints even for very simple instances of CSP: for example, at the root r of T it holds that $\mathrm{SOL}(\mathbf{I}) = C_{naive}(r)$, and hence $C_{naive}(r)$ would have superpolynomial size for every instance of CSP with a superpolynomial number of solutions. In particular, an algorithm which computes and stores $C_{naive}(j)$ would never run in polynomial time for CSP instances with a superpolynomial number of solutions.

A Better Approach A more effective way of joining constraints along a join decomposition is to only store projections of constraints onto those variables that are still relevant for constraints which have yet to appear; this idea has been used, e.g., in algorithms which exploit hypertree width [17]. To formalize this, let $C_{proj}(j)$ be recursively defined for every node $j \in V(T)$ as follows. If j is a leaf, then $C_{proj}(j) = \pi_{\overline{V}(j)}(\varrho(j))$. Otherwise $C_{proj}(j)$ is equal to $\big(\pi_{\overline{V}(j)}(C_{proj}(j_1) \bowtie C_{proj}(j_2))\big)$, where j_1 and j_2 are the two children of j in J. In this case, $\mathbf{I}$ is a YES-instance if and only if $C_{proj}(r)$ does not contain the empty relation. Clearly, for every node j of J it holds that $C_{proj}(j)$ has at most as many tuples as $C_{naive}(j)$, but can have arbitrarily fewer tuples; in particular, an algorithm which uses join decompositions to compute C_{proj} in a bottom-up fashion can solve CSP instances in polynomial time even if they have a superpolynomial number of solutions (see also Observation 8).

However, the above approach still does not capture the algorithmic power offered by dynamically computing joins along a join decomposition. In particular, similarly as has been done in the evaluation algorithm for fractional edge cover [22, Theorem 3.5], we can further reduce the size of each constraint $C_{proj}(j)$ computed in the above procedure by *pruning* all tuples that would immediately violate a constraint c in $\mathbf{I}$ (and, in particular, in $C \setminus C(j)$). To formalize this operation, we let $\mathsf{prune}(c)$ denote the *pruned constraint* w.r.t. $\mathbf{I}$, i.e., $\mathsf{prune}(c)$ is obtained from c by removing all tuples $t \in R(c)$ such that there is a constraint $c' \in C$ with $t[S(c')] \notin \pi_{S(c)}(c')$. This leads us to our final notion of dynamically computed constraints: for a node j, we let $C(j) = \mathsf{prune}(C_{proj}(j))$. We note that this, perhaps inconspicuous, notion of pruning is in fact critical—without it, one cannot use join decompositions to efficiently solve instances of small fractional hypertree width or even small fractional edge cover. A more in-depth discussion on this topic is provided in Section 4.

We can now proceed to formally define the considered width measures.

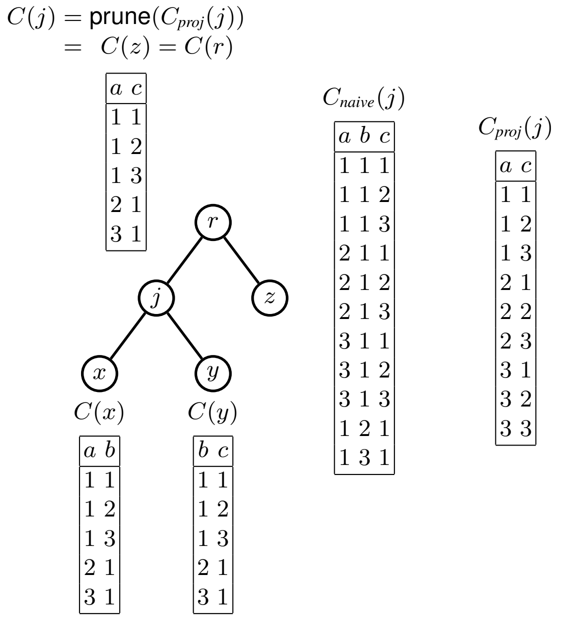
Definition 3 *Let $\mathcal{J} = (J, \varrho)$ be a join decomposition for $\mathbf{I}$ and let $j \in V(J)$. The* joinwidth *of j, denoted $\mathsf{jw}(j)$, is the smallest real number ω such that $|C(j)| \leq (\sharp_{\mathrm{tup}}(\mathbf{I}))^{\omega}$, i.e., $\omega = \log_{\sharp_{\mathrm{tup}}(\mathbf{I})} |C(j)|$. The joinwidth of $\mathcal{J}$ (denoted $\mathsf{jw}(\mathcal{J})$) is then the maximum $\mathsf{jw}(j)$ over all $j \in V(J)$. Finally, the joinwidth of $\mathbf{I}$ (denoted $\mathsf{jw}(\mathbf{I})$) is the minimum $\mathsf{jw}(\mathcal{J})$ over all join decompositions $\mathcal{J}$ for $\mathbf{I}$.*

In general terms, an instance $\mathbf{I}$ has joinwidth ω if it admits a join decomposition where the number of tuples of the produced constraints never increases beyond the ω-th power of the size of the largest relation in $\mathbf{I}$. Analogously as above, we denote by $\mathsf{ljw}(\mathbf{I})$ the minimum joinwidth of any linear join decomposition of a CSP instance $\mathbf{I}$.

Example 4 *Let $N \in \mathbb{N}$ and consider the CSP instance $\mathbf{I}$ having three variables a, b, and c and three constraints x, y, and z with scopes (a,b), (b,c), and (a,c), respectively. Assume furthermore that the relations of all three constraints are identical and contain all tuples $(1,i)$ and $(i,1)$ for every $i \in [N]$. Refer also to Figure 1 for an illustration of the example. Then $|x| = |y| = |z| = \sharp_{\mathrm{tup}}(\mathbf{I}) = 2N - 1$ and due to the symmetry of $\mathbf{I}$ any join-tree $\mathcal{J}$ of $\mathbf{I}$ has the same joinwidth, which (as we will show) is equal to 1. To see this consider for instance the join-tree $\mathcal{J}$ that has one inner node j joining x and y and a root node r joining $C(j)$ and z. Then $\mathsf{jw}(\ell) = 1$ for any leaf node ℓ of $\mathcal{J}$. Moreover $|C(j)| = |\mathsf{prune}(C_{\mathrm{proj}}(j))| = |z| = \sharp_{\mathrm{tup}}(\mathbf{I})$ since the pruning step removes all tuples from $C_{\mathrm{proj}}(j)$ that are not in z and consequently $C(r) = z$ and $\mathsf{jw}(\mathcal{J}) = 1$. Note that in this example $\mathsf{jw}(\mathbf{I}) = 1 < \mathsf{fhtw}(\mathbf{I}) = 3/2$.*

Finally, we remark that one could in principle also define joinwidth in terms of a (rather tedious and technically involved) variant of hypertree decompositions. However, the inherent algorithmic nature of join-trees makes them much better suited for the definition of joinwidth.

Properties of Join Decompositions. Our first task is to formalize the intuition behind the constraints $C(j)$ computed when proceeding through the join tree.



$C(j) = \mathsf{prune}(C_{proj}(j)) = C(z) = C(r)$

a	c
1	1
1	2
1	3
2	1
3	1

$C_{naive}(j)$

a	b	c
1	1	1
1	1	2
1	1	3
2	1	1
2	1	2
2	1	3
3	1	1
3	1	2
3	1	3
1	2	1
1	3	1

$C_{proj}(j)$

a	c
1	1
1	2
1	3
2	1
2	2
2	3
3	1
3	2
3	3

$C(x)$

a	b
1	1
1	2
1	3
2	1
3	1

$C(y)$

b	c
1	1
1	2
1	3
2	1
3	1

Fig. 1: The join decomposition given in Example 4 for $N = 3$ together with the intermediate constraints obtained for the node j.

Lemma 5 *Let (J, ϱ) be a join decomposition for $\mathbf{I} = \langle V, D, C\rangle$ and let $j \in V(J)$. Then $C(j) = \pi_{\overline{V}(j)}(\mathrm{SOL}(\mathbf{I}'))$, where $\mathbf{I}' = \mathbf{I}[V(j)]$.*

Proof. We prove the lemma by leaf-to-root induction along J. If j is a leaf such that $\varrho(j) = c$, then $C(j)$ is the constraint obtained from c by projecting onto $\overline{V}(j)$ and then applying pruning with respect to $\mathbf{I}$. Crucially, pruning c w.r.t. $\mathbf{I}'$ produces the same result as pruning c w.r.t. $\mathbf{I}$. Since pruning cannot remove tuples which occur in $\mathrm{SOL}(\mathbf{I}')$, each tuple in $\mathrm{SOL}(\mathbf{I}')$ must also occur in $C(j)$ (as a projection onto $\overline{V}(j)$). On the other hand, consider a tuple α in $C(j)$ and assume for a contradiction that α is not present in $\pi_{\overline{V}(j)}(\mathrm{SOL}(\mathbf{I}'))$. Since variables outside of $\overline{V}(j)$ do not occur in the scopes of constraints other than c, this means that there would exist a constraint c' in $\mathbf{I}'$ which is not satisfied by an assignment corresponding to α—but in that case α would be removed from $C(j)$ via pruning. Hence $C(j) = \pi_{\overline{V}(j)}(\mathrm{SOL}(\mathbf{I}'))$ holds for every leaf in T.

For the induction step, consider a node j with children j_1 and j_2 (with their corresponding instances being $\mathbf{I}'_1$ and $\mathbf{I}'_2$, respectively), and recall that $C(j)$ is obtained from $C(j_1) \bowtie C(j_2)$ by projecting onto $\overline{V}(j)$ and then pruning (w.r.t. $\mathbf{I}$ or, equivalently, w.r.t. $\mathbf{I}'$). We will also implicitly use the fact that $\overline{V}(j) \subseteq \overline{V}(j_1) \cup \overline{V}(j_2)$ and $V(j) = V(j_1) \cup V(j_2)$. First, consider for a contradiction that there exists a tuple β in $\mathrm{SOL}(\mathbf{I}'[\overline{V}(j)])$ which does not occur in $C(j)$. Clearly, β could not have been removed by pruning, and hence this would mean that there exists no tuple in $C(j_1) \bowtie C(j_2)$ which results in β after projection onto $\overline{V}(j)$; in particular, w.l.o.g. we may assume that every tuple in $C(j_1)$ differs from β in (the assignment of) at least one variable. However, since β occurs in $\mathrm{SOL}(\mathbf{I}'[\overline{V}(j)])$, there must exist at least one tuple, say β', which occurs in $\mathrm{SOL}(\mathbf{I}')$, and consequently there exists a tuple in $\mathrm{SOL}(\mathbf{I}'_1)$ which matches β in (the assignment of) all variables. At this point, we have reached a contradiction with the inductive assumption that $\mathrm{SOL}(\mathbf{I}'_1[\overline{V}(j_1)]) = C(j_1)$.

For the final case, consider a tuple γ in $C(j)$ and assume for a contradiction that γ is not present in $\pi_{\overline{V}(j)}(\mathrm{SOL}(\mathbf{I}'))$. This means that there exists at least one constraint, say c', in $\pi_{\overline{V}(j)}(\mathrm{SOL}(\mathbf{I}'))$ which would be invalidated by (an assignment corresponding to) γ. Let us assume that c' occurs in the subtree rooted in j_2, and let γ_1 be an arbitrary "projection" of γ onto $\overline{V}(j_1)$. Since $\overline{V}(j_1) \supseteq S(c')$, this means that γ_1 would have been removed from $C(j_1)$ by pruning; in particular, we see that there exists no tuple γ_1 in $C(j_1)$ which could produce γ in a join, contradicting our assumptions about γ. By putting everything together, we conclude that indeed $C(j) = \pi_{\overline{V}(j)}(\mathrm{SOL}(\mathbf{I}'))$. □

Next, we show how join decompositions can be used to solve CSP.

Theorem 6 CSP *can be solved in time $\mathcal{O}(|\mathbf{I}|^{2\omega+4})$ provided that a join decomposition of width at most ω is given in the input.*

Proof. Let $\mathcal{J} = (J, \varrho)$ be the provided join decomposition of width ω for $\mathbf{I}$. As noted before, the algorithm for solving $\mathbf{I}$ computes $C(j)$ for every $j \in V(J)$ in a bottom-up manner. Since J has exactly $2|C| - 1$ nodes, it remains to analyse the maximum time required to compute $C(j)$ for any node of J. If j is a leaf, then $C(j) = \mathsf{prune}(\pi_{S(j)}(\varrho(j)))$ and since the time required to compute the projection $P = \pi_{S(j)}(\varrho(j))$ from $\varrho(j)$ is at most $\mathcal{O}(\sharp_{tup}(\mathbf{I})|S(j)|)$ and the time required to compute the pruned constraint $\mathsf{prune}(P)$ from P is at most $\mathcal{O}(|C|(\sharp_{tup}(\mathbf{I}))^2|S(j)|)$, we obtain that $C(j)$ can be computed in time $\mathcal{O}(|C|(\sharp_{tup}(\mathbf{I}))^2|S(j)|) \in \mathcal{O}(|\mathbf{I}|^4)$. Moreover, if j is an inner node with children j_1 and j_2, then $C(j) = \mathsf{prune}(\pi_{S(j)}(C(j_1) \bowtie C(j_2)))$ and since we require at most $\mathcal{O}((\sharp_{tup}(\mathbf{I}))^{2\omega}|S(j_1) \cup S(j_2)|)$ time to compute the join $Q = C(j_1) \bowtie C(j_2)$ from $C(j_1)$ and $C(j_2)$, at most $\mathcal{O}((\sharp_{tup}(\mathbf{I}))^{2\omega}|S(j)|)$ time to compute the projection $P = \pi_{S(j)}(Q)$ from Q, and at most $\mathcal{O}(|C|(\sharp_{tup}(\mathbf{I}))^{2\omega+1} \cdot |S(j)|)$ time to compute the pruned constraint $\mathsf{prune}(P)$ from P, we obtain $\mathcal{O}(|C|(\sharp_{tup}(\mathbf{I}))^{2\omega+1} \cdot |S(j_1) \cup S(j_2)|) = \mathcal{O}(|\mathbf{I}|^{2\omega+3})$ as the total time required to compute $C(j)$. Multiplying the time required to compute $C(j)$ for an inner node $j \in V(J)$ with the number of nodes of T yields the running time stated in the lemma. □

Computing Join Decompositions. Next, let us address the problem of computing join decompositions of bounded joinwidth, formalized as follows.

ω-JOIN DECOMPOSITION
Input: A CSP instance $\mathbf{I}$.
Question: Compute a join decomposition for $\mathbf{I}$ of width at most ω, or correctly determine that $\mathsf{jw}(\mathbf{I}) > \omega$.

We show that ω-JOIN DECOMPOSITION is NP-hard even for width $\omega = 1$. This is similar to fractional hypertree width, where it was only very recently shown that deciding whether $\mathsf{fhtw}(\mathbf{I}) \leq 2$ is NP-hard [15], settling a question which had been open for about a decade. Our proof is, however, entirely different from the corresponding hardness proof for fractional hypertree width and uses a reduction from the NP-complete BRANCHWIDTH problem [35].

Theorem 7 1-JOIN DECOMPOSITION *is* NP-*hard, even on Boolean CSP instances.*

Proof. We will reduce from the well-established NP-complete BRANCHWIDTH problem, which given a graph G and an integer ω asks, whether G has branchwidth at most ω. Let (G, ω) be an instance of the BRANCHWIDTH problem. We will first show how to construct a ternary CSP instance $\mathbf{I} = (V, D, C)$ such that $\mathsf{bw}(G) \leq \omega$ if and only if $\mathsf{jw}(\mathbf{I}) \leq \log_{|V(G)|+2}|V(G)| + \omega \leq 2$; we will then later explain how to adapt the construction of $\mathbf{I}$ to make its domain boolean and to reduce the required joinwidth to 1. Let $V(G) = \{v_1, \dots, v_n\}$. We set:

- $V = V(G) \cup \{a\}$,
- $D = \{1, \cdots, n\}$,
- for every $e = \{v_i, v_j\} \in E(G)$, $\mathbf{I}$ has a constraint c_e with scope (a, v_i, v_j) whose relation $R(c_e)$ contains every tuple t such that:
  - $t[a] \in \{1, \cdots, n\}$,
  - for $l \in \{i, j\}$, it holds that $t[v_l] \in \{1, 2\}$ if $t[a] = l$ and $t[v_l] = 1$ otherwise.

Note that $\sharp_{tup}(\mathbf{I}) = n+2$. Since $\mathbf{I}$ can clearly be constructed in polynomial time it only remains to show that $\mathsf{bw}(G) \leq \omega$ if and only if $\mathsf{jw}(\mathbf{I}) \leq \log_{n+2}(n+\omega) \leq 2$.

Towards showing the forward direction, let $\mathcal{B} = (B, \beta)$ be a branch decomposition for G of width at most ω. We claim that $\mathcal{J} = (B, \varrho)$, where $\varrho(l) = c_{\beta(l)}$ for every $l \in L(B)$, is a join decomposition for $\mathbf{I}$ of width at most $\log_{n+2}(n+\omega)$. Consider any node $j \in V(B)$, then $S(j) = \delta(B_j) \cup \{a\}$, moreover, the constraint $C(j)$ contains every tuple t such that:

- $t[a] \in \{1, \cdots, n\}$,
- for every $v_i \in S(j) \setminus \{a\}$, it holds that $t[v_i] \in \{1, 2\}$ if $t[a] = i$ and $t[v_i] = 1$ otherwise.

Hence $\mathsf{jw}(j) = \log_{\sharp_{tup}(\mathbf{I})}|C(j)| = \log_{n+2}(n + |\delta(j)|) \leq \log_{n+2}(n+\omega)$. Since this holds for every $j \in V(J)$, we obtain that $\mathsf{jw}(\mathcal{J}) = \log_{n+2}(n+\omega)$, as required.

Towards showing the reverse direction, let $\mathcal{J} = (J, \varrho)$ be a join decomposition for $\mathbf{I}$ of width at most $\log_{n+2}(n+\omega)$. We claim that $\mathcal{B} = (J, \beta)$, where $\beta(l) = e$ if $\varrho(l) = c_e$ for every $l \in L(B)$, is a branch decomposition for G of width at most ω. Consider any node $j \in V(J)$, then $\delta(B_j) = S(j) \setminus \{a\}$, moreover, the constraint $C(j)$ contains every tuple t such that:

- $t[a] \in \{1, \cdots, n\}$,
- for every $v_i \in S(j) \setminus \{a\}$, it holds that $t[v_i] \in \{1, 2\}$ if $t[a] = i$ and $t[v_i] = 1$ otherwise.

Hence $|C(j)| = n + |\delta(j)|$, which because $|C(j)| \leq (\sharp_{tup}(\mathbf{I}))^{\log_{\sharp_{tup}(\mathbf{I})}(n+\omega)} = n + \omega$ implies that $|\delta(j)| \leq \omega$. Since this holds for every node $j \in V(J)$, we obtain that $\mathsf{bw}(G) \leq \omega$, as required. The completes the first part of the proof. We will now show how to adapt the construction of $\mathbf{I}$ in such a way that $\mathsf{bw}(G) \leq \omega$ if and only $\mathsf{jw}(\mathbf{I}) \leq 1$. The idea is to artificially increase $\sharp_{tup}(\mathbf{I})$ from $n+2$ to $n+\omega$ without changing the properties of $\mathbf{I}$ too much. To achieve this we simply introduce a new variable b with domain $\{1, \dots, n+\omega\}$ and add the complete constraint c with scope (b). It is easy to see that the reduction still works for this slightly modified CSP instance and moreover we obtain that $\mathsf{bw}(G) \leq \omega$ if and only if $\mathsf{jw}(\mathcal{J}) = \log_{n+\omega} n + \omega = 1$.

Finally, it is easy to convert $\mathbf{I}$ into a boolean CSP instance by replacing the variables a and b with $\log n$ respectively $\log n + \omega$ boolean variables. □

Partial Join Decompositions. Our final task in this section is to introduce the concept of partial join decompositions, which will serve as a useful tool in later sections. Let $\mathbf{I} = (V, D, C)$ be a CSP instance and let $\mathcal{J} = (J, \varrho)$ be a join decomposition for $\mathbf{I}$. We say that $\mathcal{J}' = (J_j, \varrho_j)$ is a *partial join decomposition* of $\mathcal{J}$ if $j \in V(J)$ and ϱ_j is the restriction of ϱ to $L(J_j)$. Note that the semantics of $\mathcal{J}'$, i.e., the notions $C(j)$ as well as $\mathsf{jw}(j)$ (for every node $j \in V(J_j)$, are independent of the concrete join decomposition $\mathcal{J}$, but only depend on $\mathcal{J}'$ and $\mathbf{I}$. We therefore say that $\mathcal{J}'$ is a *partial join decomposition* for $\mathbf{I}$ covering the constraints in $C' \subseteq C$ if $\varrho(L(J_j)) = C'$; assuming that the semantics for every node $j \in V(J_j)$ are defined as if $\mathcal{J}'$ would be part of an arbitrary join decomposition for $\mathbf{I}$.

4 Justifying Joinwidth

Below, we substantiate the use of both pruning and projections in our definition of join decomposition. In particular, we show that using pruning and projections allows the joinwidth to be significantly lower than if we were to consider joins carried out via C_{naive} or C_{proj}. More importantly, we show that join decompositions without pruning do not cover CSP instances with bounded fractional edge cover (and by extension bounded fractional hypertree width). To formalize this, let $\mathsf{jw}_{naive}(\mathbf{I})$ and $\mathsf{jw}_{proj}(\mathbf{I})$ be defined analogously as $\mathsf{jw}(\mathbf{I})$, with the distinction being that these measure the width in terms of C_{naive} and C_{proj} instead of C.

We also justify the use of trees for join decompositions by showing that there is an arbitrary difference between linear join decompositions (which precisely correspond to simple sequences of joins) and join decompositions.

Observation 8 *For every integer ω there exists a CSP instance $\mathbf{I}_\omega$ such that* $\mathsf{jw}_{\mathsf{proj}}(\mathbf{I}_\omega) \leq 1$, *but* $\mathsf{jw}_{\mathsf{naive}}(\mathbf{I}_\omega) \geq \omega$.

Proof. Consider the CSP instance $\mathbf{I}_\omega$ with variables $x, v_1, \dots, v_\omega$ and for each $i \in [\omega]$ a constraint c_i with scope $\{x, v_i\}$ containing the tuples $\langle 0, 1\rangle$ and $\langle 0, 0\rangle$. Since $\mathrm{SOL}(\mathbf{I}_\omega)$ contains 2^ω tuples, it follows that every join decomposition with root r must have $|C(r)_{naive}| = (\sharp_{tup}(\mathbf{I}))^\omega = 2^\omega$ tuples, hence $\mathsf{jw}_{naive}(\mathbf{I}_\omega) \geq \omega$.

On the other hand, consider a linear join decomposition which introduces the constraints in an arbitrary order. Then for each inner node j, it holds that $S(j) = \{x\}$ and in particular $C_{proj}(j)$ contains a single tuple (0) over scope $\{x\}$. We conclude that $\mathsf{jw}_{proj}(\mathbf{I}_\omega) \leq 1$. □

Towards showing that linear joinwidth differs from joinwidth in an arbitrary manner, we will use the following lemma, which establishes a tight relationship between the (linear) joinwidth of a CSP instance and the branchwidth of its hypergraph when all constraints of the CSP instance are *complete*, i.e., their relations contain all possible tuples.

Lemma 9 *Let $\mathbf{I} = \langle V, D, C\rangle$ be CSP instance that has only complete constraints. Then* $\mathsf{jw}(\mathbf{I}) = \mathsf{bw}(H(\mathbf{I}))/2$ *and* $\mathsf{ljw}(\mathbf{I}) = \mathsf{lbw}(H(\mathbf{I}))$.

Proof. Let $\mathbf{I} = \langle V, D, C\rangle$ be the given CSP instance with hypergraph $H = H(\mathbf{I})$. First note that any join decomposition $\mathcal{J} = (J, \varrho)$ for $\mathbf{I}$ is also a branch decomposition for H, whose width is equal to $\max_{j \in V(J)} |S(j)|$. Conversely, any branch decomposition of H is also a join decomposition for $\mathbf{I}$. Moreover, because all constraints of $\mathbf{I}$ are complete, we have that $C(j)$ is equal to the complete constraint on $|S(j)|$ variables for every node j in a join decomposition $\mathcal{J} = (J, \varrho)$ for $\mathbf{I}$. Hence,

$$\begin{aligned}
\mathsf{jw}(\mathbf{I}) &= \max_{j \in V(J)} \log_{\sharp_{tup}(\mathbf{I}}|C(j)| \\
&= \max_{j \in V(J)} \log_{d^2}(d^{|S(j)|}) \\
&= \max_{j \in V(J)} |S(j)|/2 = \mathsf{bw}(H)/2
\end{aligned}$$

The proof for the linear versions of join decompositions and branch decompositions is analogous. □

Using the above Lemma we are now ready to establish an arbitrary difference between join decompositions and linear join decompositions.

Proposition 10 *For every integer ω there exists a CSP instance $\mathbf{I}_\omega$ such that* $\mathsf{jw}(\mathbf{I}_\omega) \leq 1$ *but* $\mathsf{ljw}(\mathbf{I}_\omega) \geq \omega$.

Proof. It is well known that trees have branchwidth one but can have arbitrarily high pathwidth [11]. Since pathwidth is known to be within a constant factor of linear branchwidth [31], we obtain that for every ω, there is a tree T_ω with $\mathsf{bw}(T_\omega) \leq 1$ but $\mathsf{lbw}(T_\omega) \geq \omega$. Now let $\mathbf{I}_\omega$ be the (boolean) CSP instance containing only full binary constraints such that $H(\mathbf{I}_\omega) = T_{2\omega}$. Then because of Lemma 9 we have that $\mathsf{jw}(\mathbf{I}_\omega) = \mathsf{bw}(T_{2\omega})/2 = 0.5 \leq 1$ and $\mathsf{ljw}(\mathbf{I}_\omega) = \mathsf{lbw}(T_{2\omega})/2 = \omega$, as required. ☐

The next proposition is by far the most technically challenging of the three, and requires notions which will only be introduced later on. For this reason, we postpone its proof to the end of Subsection 5.1. The proposition shows not only that pruning can significantly reduce the size of stored constraints, but also that without pruning (i.e., with projections alone) one cannot hope to generalize structural parameters such as fractional hypertree width.

Proposition 11 *For every integer ω there exists a CSP instance $\mathbf{I}_\omega$ with hypergraph H^ω such that* $\mathsf{jw}(\mathbf{I}_\omega) \leq 2$ *and* $\mathsf{fec}(H^\omega) \leq 2$ *(and hence also* $\mathsf{fhtw}(H^\omega) \leq 2$*), but* $\mathsf{jw}_{\mathsf{proj}}(\mathbf{I}_\omega) \geq \omega$.

We believe that the above results are of general interest, as they provide useful insights into how to best utilize the joining of constraints.

5 Tractable Classes

Here, we show that join decompositions of small width not only allow us to solve a wide range of CSP instances, but also provide a unifying reason for the tractability of previously established structural parameters and tractable classes.

5.1 Fractional Hypertree Width

We begin by showing that joinwidth is a strictly more general parameter than fractional hypertree width. We start with a simple example showing that the joinwidth of a CSP instance can be arbitrarily smaller than its fractional hypertree width. Indeed, this holds for any structural parameter ψ measured purely on the hypergraph representation, i.e., we say that ψ is a *structural parameter* if $\psi(\mathbf{I}) = \psi(H(\mathbf{I}))$ for any CSP instance $\mathbf{I}$. Examples for structural parameters include fractional and generalized hypertree width, but also *submodular width* [29].

Observation 12 *Let ψ be any structural parameter such that for every ω there is a CSP instance with $\psi(\mathbf{I}) = \psi(H(\mathbf{I})) \geq \omega$. Then for every ω there is a CSP instance $\mathbf{I}_\omega$ with* $\mathsf{jw}(\mathbf{I}_\omega) \leq 1$ *but* $\psi(\mathbf{I}_\omega) \geq \omega$.

Proof. Let H_ω be any hypergraph with $\psi(H_\omega) \geq \omega$. Then the CSP instance $\mathbf{I}_\omega$ obtained from H_ω by replacing every hyperedge with an empty constraint satisfies $\psi(\mathbf{I}_\omega) \geq \omega$ and $\mathsf{jw}(\mathbf{I}_\omega) \leq 1$. ☐

Note that it is straightforward to construct more interesting examples that also show an arbitrary difference between joinwidth and the recently introduced extensions of fractional hypertree width (or even submodular width) with degree constraints and functional dependencies [26]. For instance, when comparing joinwidth with fractional hypertree width, there are many possibilities to fill the constrains in such a way that the join-width remains low and every such possibility leads to a new example showing the difference between these two width measures. To ensure this it would, e.g., be sufficient to ensure that every set of constrains that appear together in a bag of a fractional hypertree decomposition have small join-width; one such example is given in Proposition 11. The following theorem shows that, for the case of fractional hypertree width, the opposite of the above observation is not true.

Theorem 13 *For every CSP instance $\mathbf{I}$, it holds that* $\mathsf{jw}(\mathbf{I}) \leq \mathsf{fhtw}(\mathbf{I})$.

Proof. Let H be the hypergraph of the given CSP instance $\mathbf{I} = (V, D, C)$ and let $\mathcal{T} = (T, (B_t)_{t \in V(T)}, (\gamma_t)_{t \in V(T)})$ be an optimal fractional hypertree decomposition of H. We prove the theorem by constructing a join decomposition $\mathcal{J} = (J, \varrho)$ for $\mathbf{I}$, whose width is at most $\mathsf{fhtw}(H)$. Let $\alpha : E(H) \rightarrow V(T)$ be some function from the edges of H to the nodes of T such that $e \subseteq B_{\alpha(e)}$ for every $e \in E(H)$. Note that such a function always exists, because $(T, (B_t)_{t \in V(T)})$ is a tree decomposition of H. We denote by $\alpha^{-1}(t)$ the set $\{ e \in E(H) : \alpha(e) = t \}$.

The construction of $\mathcal{J}$ now proceeds in two steps. First we construct a partial join decomposition $\mathcal{J}^t = (J^t, \varrho^t)$ for $\mathbf{I}$ that covers only the constraints in $\alpha^{-1}(t)$, for every $t \in V(T)$. Second, we show how to combine all the partial join decompositions into the join decomposition $\mathcal{J}$ for $\mathbf{I}$ of width at most $\mathsf{fhtw}(H)$.

Let $t \in V(T)$ and let $\mathcal{J}^t = (J^t, \varrho^t)$ be an arbitrary partial join decomposition for $\mathbf{I}$ that covers the constraints in $\alpha^{-1}(t)$. Let us consider an arbitrary node $j \in V(J^t)$. By Lemma 5, we know that $C(j) = \pi_{S(j)}(\mathsf{SOL}(\mathbf{I}[V(j)]))$. Moreover, the fact that $\bigcup_{e \in \alpha^{-1}(t)} e \subseteq B_t$ implies $V(j) \subseteq B_t$. Since $|\pi_{S(j)}(\mathsf{SOL}(\mathbf{I}[V(j)]))| \leq |\mathsf{SOL}(\mathbf{I}[V(j)])|$, by invoking Proposition 1 we obtain that $|C(j)| \leq |\mathsf{SOL}(\mathbf{I}[V(j)])| \leq \sharp_{tup}(\mathbf{I})^{\mathsf{fhtw}(\mathbf{I})}$. Hence we conclude that $\mathsf{jw}(j) \leq \mathsf{fhtw}(H)$.

Next, we show how to combine the partial join decompositions $\mathcal{J}^t$ into the join decomposition $\mathcal{J}$ for $\mathbf{I}$. We will do this via a bottom-up algorithm that computes a (combined) partial join decomposition $\mathcal{F}^t = (F^t, \rho^t)$ (for every node $t \in V(T)$) that covers all constraints in $\alpha^{-1}(T_t) = \bigcup_{t \in V(T)} \alpha^{-1}(t)$. Initially, we set $\mathcal{F}^l = \mathcal{J}^l$ for every leaf $l \in L(T)$. For a non-leaf $t \in V(T)$ with children $t_1, \dots, t_\ell$ in T, we obtain $\mathcal{F}^t$ from the already computed partial join decompositions $\mathcal{F}^{t_1}, \dots, \mathcal{F}^{t_\ell}$ as follows. Let P be a path on the new vertices $p_1, \dots, p_\ell$ and let r_t and $r_{t_1}, \dots, r_{t_\ell}$ be the root nodes of J^t and $F^{t_1}, \dots, F^{t_\ell}$, respectively. Then we obtain F^t from the disjoint union of P, J^t, $F^{t_1}, \dots, F^{t_\ell}$ after adding an edge between r_t and p_1 and an edge between r_{t_i} and p_i for every i with $1 \leq i \leq \ell$ and setting p_ℓ to be the root of F^t. Moreover, ρ^t is obtained as the combination (i.e., union) of the functions $\varrho^t, \rho^{t_1}, \dots, \rho^{t_\ell}$. Observe that because α assigns every hyperedge to precisely one bag of T, it holds that every constraint assigned to T_t is mapped to precisely one leaf of $\mathcal{F}^t$. At this point, all that remains is to show that $\mathcal{F}^t$ has joinwidth at most $\mathsf{fhtw}(H)$.

Since we have already argued that $|\mathsf{SOL}(\mathbf{I}[V(j)])| \leq \sharp_{tup}(\mathbf{I})^{\mathsf{fhtw}(H)}$ for every node j of $\mathcal{J}^t$ and moreover we can assume that the same holds for every node j of $\mathcal{F}^{t_1}, \dots, \mathcal{F}^{t_\ell}$ by the induction hypothesis, it only remains to show that the same holds for the nodes $p_1, \dots, p_\ell$. First, observe that since $(T, (B_t)_{t \in V(T)})$ is a tree decomposition of H, it holds that $S(r_t), S(r_{t_1}), \dots, S(r_{t_\ell}) \subseteq B_t$. Indeed, consider for a contradiction that, w.l.o.g., there exists a variable $x \in S(r_{t_1}) \setminus B_t$. Then there must exist a hyperedge $e_1 \ni x$ mapped to t_1 or one of its descendants, and another hyperedge $e_i \ni x$ mapped to some node t' that is neither t_1 nor one of its descendants. But then both $B_{t'}$ and B_{t_1} must contain x, and so B_t must contain x as well. Moreover, since $S(p_i) \subseteq (S(r_t) \cup S(r_{t_1}) \cup \dots \cup S(r_{t_\ell}))$ for every $i \in [\ell]$, it follows that $S(p_i) \subseteq B_t$ as well.

Finally, recall that $C(p_i) = \pi_{S(p_i)}(\mathsf{SOL}(\mathbf{I}[V(p_i)]))$ by Lemma 5, and observe that $|\pi_{S(p_i)}(\mathsf{SOL}(\mathbf{I}[V(p_i)]))| \leq |\mathsf{SOL}(\mathbf{I}[S(p_i)])|$. Then by Proposition 1 combined with the fact that $S(p_i) \subseteq B_t$, we obtain $|C(p_i)| \leq |\mathsf{SOL}(\mathbf{I}[S(p_i)])| \leq \sharp_{tup}(\mathbf{I})^{\mathsf{fhtw}(H)}$, which implies that the width of p_i is indeed at most $\mathsf{fhtw}(H)$. □

Since we have now established the relationship between joinwidth and fractional hypertree width, we can conclude this subsection with a proof of Proposition 11 (cf. Section 4).

Proof (Proof of Proposition 11). The proof uses an adaptation of a construction given by Atserias, Grohe and Marx [3, Theorem 7]. Let $m = 4\omega + 1$ and $n = \binom{2m}{m}$. The CSP instance $\mathbf{I}_\omega$ has:

- one variable v_S for every $S \subseteq [2m]$ with $|S| = m$,
- one constraint c_i for every i with $1 \leq i \leq 2m$ with scope $\{ v_S : i \in S \}$. The relation of c_i contains the following tuples t for every $v \in S(c_i)$: every tuple t such that $t[v] \in [n]$ and $t[u] = 1$ for every $u \neq v$,
- domain $[n]$.

Let H be the hypergraph of $\mathbf{I}_\omega$. Note first that H has n vertices, $2m$ edges, every edge of H has size $\binom{2m-1}{m-1} = n/2$, every vertex of H occurs in exactly m edges, and every set of at most m edges of H misses at least one vertex, i.e., the union of at most m edges of H misses at least one vertex from $V(H)$. Note furthermore that $\sharp_{tup}(\mathbf{I}_\omega) = \frac{n^2}{2}$.

It is shown in [3, Theorem 4] that H has a fractional edge cover of size at most 2. This is witnessed by the mapping $\gamma : E(H) \to \mathbb{R}$ defined by setting $\gamma(e) = 1/m$ for every $e \in E(H)$. Because every variable v_S of $\mathbf{I}_\omega$ is in the scope of exactly m constraints (i.e. the constraints in $\{ c_i : i \in S \}$), γ is indeed a fractional edge cover and because $\sum_{e \in E(H)} \gamma(e) = 2m(1/m) = 2$ it has size exactly 2. Note that because $\mathsf{fec}(H) \geq \mathsf{fhtw}(H)$, we obtain from Theorem 13 that $jw(\mathbf{I}_\omega) \leq 2$.

It remains to show that $\mathsf{jw}_{proj}(\mathbf{I}_\omega) \geq \omega$. Consider an arbitrary join decomposition $\mathcal{J} = (J, \varrho)$ of $\mathbf{I}_\omega$. Let j be a node of J such that $\lceil m/2 \rceil \leq |L(J_j)| < m$. Note that such a node j always exists due to the following well-known combinatorial argument.

First we show that there is a node j' of J such that $|L(J_{j'})| \geq m$ but $|L(J_{j''})| < m$ for every child j'' of j' in J. Namely, j' can be found by going down the tree J starting from the root and choosing a child j'' of j' with $|L(J_{j''})| \geq m$, as long as such a child j'' exists. Then letting j be the child of j' maximizing $|L(J_j)|$ implies that $\lceil m/2 \rceil \leq |L(J_j)| < m$, as required.

We claim that $\mathsf{jw}_{\mathit{proj}}(j) \geq \omega$. First note that $S(j) = V(j)$. This is because $|C(\mathbf{I}_\omega) \setminus X(j)| \geq 2m - (m-1) = m+1$ and hence $\overline{V}(j) = \bigcup_{c \in (C(\mathbf{I}_\omega) \setminus X(j))} S(c) = V(\mathbf{I}_\omega)$; this is because $S \cap \{\, i : c_i \in C(\mathbf{I}_\omega) \setminus X(j) \,\} \neq \emptyset$ for every variable v_S of $\mathbf{I}_\omega$. Consequently, $C_{\mathit{proj}}(j)$ is equal to the join of all constraints in $X(j)$. We show next that there are at least $\lceil m/2 \rceil$ variables that appear in the scope of exactly one constraint in $X(j)$, which implies that $C_{\mathit{proj}}(j)$ contains every of the $n^{\lceil m/2 \rceil}$ tuples on these variables. To see this let O be the set of all elements $o \in [2m]$ for which $c_o \in X(j)$ and let $\overline{O}$ be the set of all the remaining elements in $[2m]$. Then $|O| \geq \lceil m/2 \rceil$ and $|\overline{O}| \geq 2m - (m-1) = m+1$. Let S' be any set of exactly $m-1$ elements from $\overline{O}$ and for every $o \in O$, let S_o be equal to $\{o\} \cup S'$. Then v_{S_o} is in the scope of exactly one constraint in $C(j)$ (i.e., the constraint c_o and hence there are at least $\lceil m/2 \rceil$ variables that occur in the scope of exactly one constraint in $X(j)$, i.e., the variables $\{\, S_o : o \in O \,\}$. Hence $C_{\mathit{proj}}(j)$ contains at least $n^{\lceil m/2 \rceil}$ tuples, which because $\sharp_{\mathit{tup}}(\mathbf{I}_\omega) = n^2/2$ implies that $\mathsf{jw}_{\mathit{proj}}(j)$ is at least $\lceil m/2 \rceil / 2 \geq m/4 \geq \omega$, as required. □

5.2 Functionality and Root Sets

Consider a CSP instance $\mathbf{I} = \langle V, D, C \rangle$ with $n = |V|$. We say that a constraint $c \in C$ is *functional* on variable $v \in V$ if c does not contain two tuples that differ *only* at variable v; more formally, for every t and $t' \in R(c)$ it holds that if $t[v] \neq t'[v]$, then there exists a variable $z \in S(c)$ distinct from v such that $t[z] \neq t'[z]$. The instance $\mathbf{I}$ is then called *functional* if there exists a variable ordering $v_1 < \cdots < v_n$ such that, for each $i \in [n]$, there exists a constraint $c \in C$ such that $\pi_{\{v_1,\ldots,v_i\}}(c)$ is functional on v_i. Observe that every CSP instance that is functional can admit at most 1 solution [5]; this restriction can be relaxed through the notion of *root sets*, which can be seen as variable sets that form "exceptions" to functionality. Formally, a variable set Q is a root set if there exists a variable ordering $v_1 < \cdots < v_n$ such that, for each $i \in [n]$ where $v_i \notin Q$, there exists a constraint $c \in C$ such that $\pi_{\{v_1,\ldots,v_i\}}(c)$ is functional on v_i; we say that Q is *witnessed* by the variable order $v_1 < \cdots < v_n$.

Functionality and root sets were studied for Boolean CSP [10,9]. Cohen et al. [5] later extended these notions to the CSP with larger domains. Our aim in this section is twofold:

- generalize root sets through the introduction of *constraint root sets*;
- show that bounded-size constraint root sets (and also root sets) form a special case of bounded joinwidth.

Before we proceed, it will be useful to show that one can always assume the root set to occur at the beginning of the variable ordering.

Observation 14 *Let Q be a root set in $\mathbf{I}$ witnessed by a variable order α, assume a fixed arbitrary ordering on Q, and let the set $V' = V(\mathbf{I}) \setminus Q$ be ordered based on the placement of its variables in α. Then Q is also witnessed by the variable order $\alpha' = Q \circ V'$.*

Proof. We need to show that Q is a root set witnessed by $\alpha' = v_1 < \cdots < v_{|V|}$, i.e., that for each i in $k < i \leq |V|$ there exists a constraint $c \in C$ such that $\pi_{\{v_1,\ldots,v_i\}}(c)$ is functional on v_i. Consider an arbitrary such variable v_i, and recall that since Q is witnessed by α, there must exist a constraint c such that $c' = \pi_{Q_i \cup V_i}(c)$ is functional on v_i, where V_i' and Q_i are, respectively, the elements of Q and V' which occur before v_i in α. Now, observe that the constraint $c^* = \pi_{Q \cup V_i}(c) = \pi_{\{v_1,\ldots,v_i\}}(c)$ must also be functional on v_i—indeed, for each pair of tuples t, t' in $R(c^*)$, either $t[v_i] = t'[v_i]$, or $t[v_i] \neq t'[v_i]$ and there exists a variable $z \in Q_i \cup V_i$ (and hence also in the scope of c^*) such that $t[z] \neq t'[z]$. □

For ease of presentation, we will say that $\mathbf{I}$ is *k-rooted* if k is the minimum integer such that $\mathbf{I}$ has a root set of size k. It is easy to see, and also follows from the work of David [9] and Cohen et al. [5], that for every fixed k the class of k-rooted CSP instances is polynomial-time solvable: generally speaking, one can first loop through and test all variable-subsets of size at most k to find a root Q, and then loop through all assignments $Q \to D$ to get a set of functional CSP instances, each of which can be solved separately in linear time.

While even 1-rooted CSP instance can have unbounded fractional hypertree width (see also the discussion of Cohen et al. [5]), the class of k-rooted CSP instances for a fixed value k is, in some sense, not very robust. Indeed, consider the CSP instance $\mathcal{W} = \langle \{v_1, \ldots, v_n\}, \{0,1\}, \{c\} \rangle$ where c ensures that precisely a single variable is set to 1 (i.e., its relation can be seen as an $n \times n$ identity matrix). In spite of its triviality, it is easy to verify that $\mathcal{W}$ is not k-rooted for any $k < n-2$.

Let us now consider the following alternative to measuring the size of root sets in a CSP instance $\mathbf{I}$. A constraint set P is a *constraint-root set* if $\bigcup_{c \in P} S(c)$ is a root set, and $\mathbf{I}$ is then called k-constraint-rooted if k is the minimum integer such that $\mathbf{I}$ has a constraint-root set of size k. Since we can assume that each variable occurs in at least one constraint, every k-rooted CSP also has a constraint-root set of size at most k; on the other hand, the aforementioned example of $\mathcal{W}$ shows that an instance can be 1-constraint-rooted while not being k-rooted for any small k. The following result, which we prove by using join decompositions and joinwidth, thus gives rise to strictly larger tractable classes than those obtained via root sets:

Proposition 15 *For every fixed $k \in \mathbb{N}$, every k-constraint-rooted CSP instance has joinwidth at most k and can be solved in time $|\mathbf{I}|^{\mathcal{O}(k)}$.*

Proof. Consider a CSP instance $\mathbf{I}$ with a constraint-root set P of size k. We argue that $\mathbf{I}$ has a linear join decomposition of width at most k where the elements of P occur as the leaves farthest from the root. Indeed, consider the linear join decomposition (J, ϱ) constructed in a bottom-up manner, as follows. First, we start by gradually adding the constraints in P as the initial leaves. At each step after that, consider a node j which is the top-most constructed node in the join decomposition. By definition, there must exist a variable v and a constraint c such that $\pi_{\bigcup_{c \in P} S(c)}(c)$ is functional on v. Moreover, this implies that $|\pi_{\bigcup_{c \in P} S(c)}(c) \bowtie C(j)| \leq |C(j)|$, and thus $|c \bowtie C(j)| \leq |C(j)|$. Hence this procedure does not increase the size of constraints at nodes after the initial k constraints, immediately resulting in the desired bound of k on the width of (J, ϱ).

To complete the proof, observe that a join decomposition with the properties outlined above can be found in time at most $|\mathbf{I}|^{\mathcal{O}(k)}$: indeed, it suffices to branch over all k-element subsets of $C(\mathbf{I})$ and test whether the union of their scopes is functional using, e.g., the result of Cohen et al. [5, Corollary 1]. Once we have such a join decomposition, we can solve the instance by invoking Theorem 6. □

As a final remark, we note that the class of k-constraint rooted CSP instances naturally includes all instances which contain k constraints that are in conflict (i.e., which cannot all be satisfied at the same time).

5.3 Other Tractable Classes

Here, we identify some other classes of tractable CSP instances with bounded joinwidth. First of all, we consider CSP instances such that introducing their variables in an arbitrary order always results in a subinstance with polynomially many solutions. In particular, we call a CSP instance $\mathbf{I}$ *hereditarily k-bounded* if for every subset V' of its variables it holds that $|\text{SOL}(\mathbf{I}[V'])| \leq \sharp_{tup}(\mathbf{I})^k$. Examples of hereditarily k-bounded CSP instances include k-Turan CSPs [5, page 12] and CSP instances with fractional edge covers of weight k [22].

Proposition 16 *The class of hereditarily k-bounded CSP instances has joinwidth at most k and can be solved in time at most $\mathcal{O}(|\mathbf{I}|^k)$.*

Proof. Consider an arbitrary linear join decomposition (J, ϱ). By definition, for each $j \in V(J)$ it holds that $|\text{SOL}(\mathbf{I}[V(j)])| \leq \sharp_{tup}(\mathbf{I})^k$. Then $|\pi_{\overline{V}(j)}(\text{SOL}(\mathbf{I}[V(j)]))| \leq \sharp_{tup}(\mathbf{I})^k$, and by Lemma 5 we obtain $|C(j)| \leq \sharp_{tup}(\mathbf{I})^k$, as required. □

Another example of a tractable class of CSP instances that we can solve using joinwidth are instances where all constraints interact in a way which forces a unique assignment of the variables. In particular, we say that a CSP $\mathbf{I} = \langle V, D, C \rangle$ is *unique at depth k* if for each constraint $c \in C$ there exists a *fixing set $C' \subseteq C$* such that $c \in C'$, $|C'| \leq k$, and $|(\bowtie_{c' \in C'} c')| \leq 1$.

Proposition 17 *The class of CSP instances which are unique at depth k has joinwidth at most k and can be solved in time at most $|\mathbf{I}|^{\mathcal{O}(k)}$.*

Proof. First of all, we observe that it is possible to determine whether a CSP instance $\mathbf{I}$ is unique at depth k in time at most $|\mathbf{I}|^{\mathcal{O}(k)}$. Indeed, it suffices to check whether each constraint c is contained in at least one fixing set—and to do that, we can simply loop over all constraint sets of arity at most k and join them together (in an arbitrary order). If $\mathbf{I}$ contains a constraint that does not appear in any fixing set of size at most k, then it is not unique at depth k; otherwise, for each constraint c we choose an arbitrary fixing set (containing c) and denote it as $\mathsf{Fix}(c)$.

Consider a linear join decomposition $\mathcal{J}$ constructed as follows. First, we choose an arbitrary ordering of the constraints and denote these $c_1 < \cdots < c_m$. We begin our construction by having $\mathcal{J}$ introduce the constraints which occur in $\mathsf{Fix}(c_1)$ (in an arbitrary order). Then, for each $2 \leq i \leq m$, we introduce those constraints in $\mathsf{Fix}(c_i)$ which are not yet introduced in $\mathcal{J}$ (also in an arbitrary order).

It remains to argue that $\mathcal{J}$ has joinwidth at most k. Consider the node j_1 whose child is the last constraint occurring in $\mathsf{Fix}(c_1)$ (i.e., all constraints in $\mathsf{Fix}(c_1)$ occur as leaves which are descendants of j_1, and all constraints not in $\mathsf{Fix}(c_1)$ occur as leaves which are not descendants of j_1). By definition, $| \bowtie_{c' \in \mathsf{Fix}(c_1)} c'| \leq 1$ and hence by Lemma 5 we obtain $|C(j_1)| \leq 1$. Moreover, since there are at most k leaves below j_1, for each descendant j_1' of j_1 we obtain that $|C(j_1')| < (\sharp_{tup}(\mathbf{I}))^k$, as required. Now, consider an arbitrary $2 \leq i \leq m$, and as before let j_i be the node whose child is the last constraint occurring in $\mathsf{Fix}(c_i)$. Once again, $|C(j_i)| \leq 1$ by definition. Moreover, since there are at most k leaves which occur "between" j_i and j_{i-1}, $\mathcal{J}$ contains at most $k-1$ non-leaf nodes between j_i and j_{i-1}. Since $|C(j_{i-1})| \leq 1$, it follows that every node j_i' between j_i and j_{i-1} also has $|C(j_1')| < (\sharp_{tup}(\mathbf{I}))^k$. Hence $\mathcal{J}$ indeed has joinwidth at most k, and the proposition follows by Theorem 6. □

6 Solving Bounded-Width Instances

This section investigates the tractability of CSP instances whose joinwidth is bounded by a fixed constant ω. In particular, one can investigate two notions of tractability. The first one is the classical notion of *polynomial-time tractability*, which asks for an algorithm of the form $|\mathbf{I}|^{\mathcal{O}(1)}$. In this setting, the complexity of CSP instances of bounded joinwidth remains an important open problem. Note that the NP-hardness of the ω-JOIN DECOMPOSITION problem established in Theorem 7 does not exclude polynomial-time tractability for CSP instances of bounded joinwidth. For instance, tractability could still be obtained with a suitable approximation algorithm for computing join decompositions (as it is the case for fractional hypertreewidth [27]) or by using an algorithm that does not require a join decomposition of bounded width as input.

The second notion of tractability we consider is called *fixed-parameter tractability* and asks for an algorithm of the form $f(k) \cdot |\mathbf{I}|^{\mathcal{O}(1)}$, where k is a numerical parameter capturing a certain natural measure of $\mathbf{I}$. Prominently, Marx investigated the fixed-parameter tractability of CSP and showed that CSP instances whose hypergraphs have bounded *submodular width* [29] are fixed-parameter tractable when k is the number of variables. Moreover, Marx showed that submodular width is the most general structural property *among those measured purely on hypergraphs* with this property.

Here, we obtain two single-exponential fixed-parameter algorithms for CSP instances of bounded joinwidth (i.e., algorithms with a running time of $2^{\mathcal{O}(k)} \cdot |\mathbf{I}|^{\mathcal{O}(1)}$): one where k is the number of variables, and the other where k is the number of constraints. Since there exist classes of instances of bounded joinwidth and unbounded submodular width (see Observation 12), this expands the frontiers of (fixed-parameter) tractability for CSP.

Parameterization by Number of Constraints. To solve the case where k is the number of constraints, our primary aim is to obtain a join decomposition of width at most ω, i.e., solve the ω-JOIN DECOMPOSITION problem defined in Section 3. Indeed, once that is done we can solve the instance by Theorem 6.

Theorem 18 *ω-JOIN DECOMPOSITION can be solved in time $\mathcal{O}(4^{|C|} + 2^{|C|}|\mathbf{I}|^{2\omega+1})$ and is hence fixed-parameter tractable parameterized by $|C|$, for a CSP instance $\mathbf{I} = \langle V, D, C\rangle$.*

Proof. Let $\mathbf{I} = \langle V, D, C\rangle$ be the given CSP instance. For a subset $C' \subseteq C$, we denote by $S(C')$ the set of all variables that are both in the scope of a constraint in C' and in the scope of a constraint in $C \setminus C'$. The algorithm uses dynamic programming to compute the mapping α that for every non-empty subset C' of C either:

- is equal to $\pi_{S(C')}(\mathrm{SOL}(\mathbf{I}[V(C')]))$, if there is a partial join decomposition for $\mathbf{I}$ that covers the constraints in C' of width at most ω, and
- otherwise is equal to ∞.

Note that $\mathbf{I}$ has a join decomposition of width at most ω if and only if $\alpha(C) \neq \infty$.

To compute α we use the observation that a partial join decomposition covering a set C' of constraints is either a leaf (if $|C'| = 1$) or is obtained as the "join" of a partial join decomposition covering C_0 and a partial join decomposition covering $C' \setminus C_0$ for some non-empty $C_0 \subset C'$. This immediately implies that α satisfies the following recurrence relation for every non-empty subset C' of C:

(R1) if $C' = \{c\}$ for some $c \in C$, then $\alpha(C') = \mathsf{prune}(\pi_{S(C')}(c))$,
(R2) if $|C'| > 1$ and there is a non-empty subset $C_0 \subset C'$ such that:
  (C1) $\alpha(C_0) \neq \infty$ and $\alpha(C' \setminus C_0) \neq \infty$, and
  (C2) $|\mathsf{prune}((\pi_{S(C')}(\alpha(C_0) \bowtie \alpha(C' \setminus C_0))))| \leq \sharp_{tup}(\mathbf{I})^{\omega}$,
  then $\alpha(C') = \mathsf{prune}(\pi_{S(C')}((\alpha(C_0) \bowtie \alpha(C \setminus C_0))))$,
(R3) otherwise, i.e., if $|C'| > 1$ and (R2) does not apply, then $\alpha(C') = \infty$.

Since $\mathbf{I}$ has a join decomposition of width at most ω if and only if $\alpha(C) \neq \infty$ and in this case a join decomposition for $\mathbf{I}$ can be easily constructed by following the recurrence relation starting from $\alpha(C)$, it only remains to show how to compute α in the required running time, which we do as follows.

Initially, we set $\alpha(\{c\}) = \mathsf{prune}(\pi_{S(C')}(c))$ for every $c \in C$ and $\alpha(C') = \infty$ for every C' with $\emptyset \neq C' \subseteq C$ and $|C'| > 1$. We then enumerate all subsets C' with $C' \subseteq C$ and $|C'| > 1$ in order of increasing size and check whether there is a subset C_0 of C' satisfying the conditions stated in (R2). If so we set $\alpha(C') = \mathsf{prune}((\pi_{S(C')}(\alpha(C_0) \bowtie \alpha(C \setminus C_0))))$ and otherwise we set $\alpha(C') = \infty$. Clearly, the time required to initialize α is at most $\mathcal{O}(2^{|C|} + |\mathbf{I}|)$. Furthermore, the time required to check, whether a subset C' satisfies the conditions stated in (R2) is at most $\mathcal{O}(2^{|C'|} + |\mathbf{I}|^{2\omega+1}) = \mathcal{O}(2^{|C|} + |\mathbf{I}|^{2\omega+1})$, which can be obtained as follows. First note that because of Lemma 5, it holds that $\mathsf{prune}(\pi_{S(C')}((\alpha(C_0) \bowtie \alpha(C' \setminus C_0))))$ is equal to $\mathsf{prune}((\pi_{S(C')}(\alpha(C_0') \bowtie \alpha(C' \setminus C_0'))))$ for any two subsets C_0 and C_0' satisfying (C1) and hence once we found a subset C_0 satisfying (C1) and have computed $\mathsf{prune}((\pi_{S(C')}(\alpha(C_0) \bowtie \alpha(C' \setminus C_0))))$, we can determine whether (R2) or (R3) applies for C'. This implies that we have to compute $\mathsf{prune}((\pi_{S(C')}(\alpha(C_0) \bowtie \alpha(C' \setminus C_0))))$ for at most one subset C_0, which explains why $|\mathbf{I}|^{2\omega+1}$ only appears additively in the running time. Moreover, the term $2^{|C|}$ is required for the enumeration of all subsets C_0 of C'. Since we have to enumerate all subsets C' of C, we hence obtain $\mathcal{O}(2^{|C|}(2^{|C|} + |\mathbf{I}|^{2\omega+1})) = \mathcal{O}(4^{|C|} + 2^{|C|}|\mathbf{I}|^{2\omega+1})$ as the total running time for computing α. □

From Theorem 18 and Theorem 6 we immediately obtain:

Corollary 19 *A CSP instance* $\mathbf{I}$ *with* k *constraints and joinwidth at most* ω *can be solved in time* $2^{\mathcal{O}(k)} \cdot |\mathbf{I}|^{\mathcal{O}(\omega)}$.

Parameterization by Number of Variables. Note that Corollary 19 immediately establishes fixed-parameter tractability for the problem when k is the number of variables (instead of the number of constraints), because one can assume that $|C| \leq 2^{|V|}$ for every CSP instance $\mathbf{I} = (V, D, C)$. However, the resulting algorithm would be double-exponential in $|V|$. The following theorem shows that this can be avoided by designing a dedicated algorithm for CSP parameterized by the number of variables. The main idea behind both algorithms is dynamic programming, however, in contrast to the algorithm for $|C|$, the table entries for the fpt-algorithm for $|V|$ correspond to subsets of V instead of subsets of C. Interestingly, the fpt-algorithm for $|V|$ does not explicitly construct a join decomposition, but only implicitly relies on the existence of one.

Theorem 20 *A CSP instance* $\mathbf{I}$ *with* k *variables and joinwidth at most* ω *can be solved in time* $2^{\mathcal{O}(k)} \cdot |\mathbf{I}|^{\mathcal{O}(\omega)}$.

Proof. Let $\mathbf{I} = \langle V, D, C \rangle$ be the given CSP instance with $\mathsf{jw}(\mathbf{I}) \leq \omega$. For a subset $V' \subseteq V$, we denote by $S(V')$ the set of variables in $V' \cap \{ v : c \in C \wedge S(c) \setminus V' \neq \emptyset \wedge v \in S(c) \}$ (i.e., $S(V')$ contains variables in V' which occur in the scope of a constraint that also has variables outside of V'). The algorithm uses dynamic programming to compute a mapping α which maps non-empty subsets V' of V to either the constraint $\pi_{S(V')}(\mathrm{SOL}(\mathbf{I}[V']))$, or to ∞. α is defined using the following recurrence:

(R1) if $V' = S(c)$ for some $c \in C$, then $\alpha(V') = \mathsf{prune}(\pi_{S(V')}(c))$,
(R2) if (R1) does not apply and there are subsets V_0 and V_1 with $\emptyset \neq V_0, V_1 \subset V'$ and $V_0 \cup V_1 = V'$ such that:
  (C1) $\alpha(V_0) \neq \infty$ and $\alpha(V_1) \neq \infty$, and
  (C2) $|\mathsf{prune}(\pi_{S(V')}(\alpha(V_0) \bowtie \alpha(V_1)))| \leq \sharp_{tup}(\mathbf{I})^{\omega}$,
  then $\alpha(V') = \mathsf{prune}(\pi_{S(V')}(\alpha(V_0) \bowtie \alpha(V_1)))$,
(R3) otherwise, i.e., if neither (R1) nor (R2) applies, then $\alpha(V') = \infty$.

Using arguments very similar to the ones used in the proof of Lemma 5 it is now easy to show that if $\alpha(V') \neq \infty$, then $\alpha(V') = \pi_{S(V')}(\mathsf{SOL}(\mathbf{I}[V']))$ for every subset $V' \subseteq V$.

It follows that if $\alpha(V) \neq \infty$, then $\mathbf{I}$ has a solution if and only if $\alpha(V)$ is not equal to the empty constraint. Hence an algorithm that computes α can be used to solve every CSP instance for which $\alpha(V) \neq \infty$. In order to show that we can solve a CSP instance $\mathbf{I}$ of joinwidth at most ω, it is hence sufficient to show that $\alpha(V) \neq \infty$.

To this end, consider a join decomposition $\mathcal{J} = (J, \varrho)$ for $\mathbf{I}$ of width at most ω. Using a bottom-up induction on J, we will show that $\alpha(V(j)) \neq \infty$ and moreover $C(j) = \alpha(V(j))$ for every $j \in V(j)$. Since $\alpha(V) = C(r) \neq \infty$ for the root r of J, this then implies the desired property. $C(j) = \alpha(V(j))$ clearly holds for every leaf l of J, because $V(l)$ satisfies (R1) and by Lemma 5 we have $C(l) = \pi_{S(l)}(\mathsf{SOL}(\mathbf{I}[V(l)])) = \pi_{S(V(l))}(\mathsf{SOL}(\mathbf{I}[V(l)]))$. Now, consider an inner node j of J with children j_1 and j_2. Then setting V' to $V(j)$, V_0 to $V(j_1)$, and V_1 to $V(j_2)$ satisfies (C1) (of (R2)) because of the induction hypothesis. Moreover, (C2) is also satisfied because $\mathsf{prune}(\pi_{S(V')}(\alpha(V_0) \bowtie \alpha(V_1))) = \pi_{S(V')}(\mathsf{SOL}(\mathbf{I}[V'])) = \pi_{S(j)}(\mathsf{SOL}(\mathbf{I}[V(j)])) = C(j)$ and $|C(j)| \leq \sharp_{tup}(\mathbf{I})^{\omega}$. Hence an algorithm that computes α can be used to solve every CSP instance of bounded joinwidth, and it only remains to show how to compute α.

Initially, we set $\alpha(V') = \mathsf{prune}(c)$ for every $V' \subseteq V$ such that there is constraint $c \in C$ with $V' = S(c)$ and $\alpha(V') = \infty$ for every $V' \subseteq V$ for which this is not the case. We then enumerate all subsets V' with $\emptyset \neq V' \subseteq V$ in order of increasing sizes and if $\alpha(V') = \infty$, we check whether there are subsets V_0 and V_1 of V' satisfying the conditions stated in (R2). If so we set $\alpha(V') = \mathsf{prune}(\pi_{S(V')}(\alpha(V_0) \bowtie \alpha(V_1)))$ and otherwise we set $\alpha(V') = \infty$. Clearly, the time required to initialize α is at most $\mathcal{O}(2^{|V|} + |C|)$. Furthermore, the time required to check whether a subset V' satisfies the conditions stated in (R2) is at most $\mathcal{O}(2^{|V|}2^{|V|} + |\mathbf{I}|^{2\omega+1}) = \mathcal{O}(4^{|V|} + |\mathbf{I}|^{2\omega+1})$—indeed, this can be carried out as follows. To begin, we observe that $\mathsf{prune}(\pi_{S(V')}(\alpha(V_0) \bowtie \alpha(V_1)))$ is equal to $\mathsf{prune}(\pi_{S(V')}(\alpha(V_0') \bowtie \alpha(V_1')))$ for any two pairs (V_0, V_1) and (V_0', V_1') of subsets satisfying (C1), and hence after we find a pair (V_0, V_1) of subsets satisfying (C1) and compute $\mathsf{prune}(\pi_{S(V')}(\alpha(V_0) \bowtie \alpha(V_1)))$, we can determine whether (R2) or (R3) applies for V'. This implies that we have to compute $\mathsf{prune}(\pi_{S(V')}(\alpha(V_0) \bowtie \alpha(V_1)))$ for at most one pair (V_0, V_1) of subsets, which explains why $|\mathbf{I}|^{2\omega+1}$ only appears additively in the running time. Moreover, the term $4^{|V|}$ is required for the enumeration of all pairs (V_0, V_1) for V'. Since we have to enumerate all subsets V' of V, we obtain $\mathcal{O}(2^{|V|}(4^{|V|} + |\mathbf{I}|^{2\omega+1})) = \mathcal{O}(6^{|V|} + 2^{|V|}|\mathbf{I}|^{2\omega+1})$, as the total running time for computing α. □

7 Beyond Join Decompositions

Due to their natural and "mathematically clean" definition, one might be tempted to think that join decompositions capture all the algorithmic power offered by join and projection operations. It turns out that this is not the case, i.e., we show that if one is allowed to use join and projections in an arbitrary manner (instead of the more natural but also more restrictive way in which they are used within join decompositions) one can solve CSP instances that are out-of-reach even for join decompositions. This is interesting as it points towards the possibility of potentially more powerful parameters based on join and projections than joinwidth.

Theorem 21 *For every ω, there exists a CSP instance $\mathbf{I}_\omega$ that can be solved in time $\mathcal{O}(|\mathbf{I}|^4)$ using only join and projection operations but* $\mathsf{jw}(\mathbf{I}_\omega) \geq \omega$.

Proof. Let $n = \omega * 16$ and let $\mathbf{I}_\omega = \langle V, D, C\rangle$ be the CSP instance with variables $x_1, \ldots, x_n$, domain $\{1, \ldots, n\}$, and the following constraints:

- A set $C_< = \{ c_1, \ldots, c_{n-1} \}$ of binary constraints c_i with scope (x_i, x_{i+1}) containing all tuples $t \in n \times n$ such that $t[x_i] < t[x_{i+1}]$,
- A set $C_C = \{ c_{l,m} : l < m - 1 \wedge 1 \leq l, m \leq n \}$ of binary and complete constraints $c_{l,m}$ with scope (x_l, x_m).

We start by showing that $\mathsf{jw}(\mathbf{I}_\omega) \geq \omega$. Let (J, ϱ) be any join decomposition for $\mathbf{I}_\omega$. Let j be a node of J such that $\lceil 1/4n \rceil \leq |V(j)| \leq \lfloor 1/2n \rfloor$, which exists due to the following well-known argument.

First we show that there is a node j' of J such that $|V(j')| \geq \lfloor 1/2n \rfloor$ but $|V(j'')| \leq \lfloor 1/2n \rfloor$ for every child j'' of j' in J. Namely, j' can be found by going down the tree J starting from the root and choosing a child j'' of j' with $|V(j'')| \geq \lfloor 1/2n \rfloor$; as long as such a child j'' exists. Then letting j be the child of j' maximizing $|V(j)|$ implies that $\lceil 1/4n \rceil \leq |V(j)| \leq \lfloor 1/2n \rfloor$, as required.

Note that because $\mathbf{I}_\omega$ has a constraint with scope (x_l, x_m) for every $l < m$, it holds that $\overline{V}(j) = V$ for the node $j \in V(J)$; this is because there is at least one variable in $V \setminus V(j)$ and this variable occurs in a binary constraint with every other variable. Hence by Lemma 5, it holds that $C(j) = \mathsf{SOL}(\mathbf{I}_\omega[V(j)])$.

We claim that $C(j)$ contains at least $(n/2)^{n/4}$ tuples. Towards showing the claim, let $\mathbf{I}_\omega(i)$ be any sub-instance of $\mathbf{I}_\omega$ induced by any subset of V of size i. It is easy to see that the domain of any variable in $\mathbf{I}_\omega(i)$ has size at least $n-i$ and hence SOL($\mathbf{I}_\omega$) has at least $(n-i)^i$ tuples. Hence, because $\lceil 1/4n\rceil |V(j)| \leq \lfloor 1/2n \rfloor$, we obtain that $C(j)$ has at least $\min\{(3/4n)^{1/4n}, (1/2n)^{1/2n}\} \geq (n/2)^{n/4}$ tuples, as required.

Finally, since $\sharp_{tup}(\mathbf{I}_\omega) = n^2$, we obtain the following for the joinwidth of $\mathbf{I}_\omega$.

$$\begin{aligned}
\mathsf{jw}(\mathbf{I}_\omega) &= \log_{n^2}((n/2)^{n/4}) \\
&= n/2 \log_{n^2} n/4 \\
&= n/2(\log_{n^2} n - \log_{n^2} 4) \\
&= n/2(1/2 - \frac{\log_4 4}{\log_4 n^2}) \\
&= n/2(1/2 - \frac{1}{\log_4 n^2}) \\
&\geq n/2(1/2 - \frac{1}{4}) \\
&= n/8 \\
&= 2\omega
\end{aligned}$$

The inequality in the above sequence follows since $n^2 \geq 16^2 = 4^4$.

It remains to show that we can solve $\mathbf{I}_\omega$ in time $\mathcal{O}(|\mathbf{I}_\omega|^4)$ using only join and projection operations.

Towards showing this we now define various auxiliary constraints that can be obtained efficiently using only join and projection operations from the constraints in C.

Namely, for every i with $1 < i \leq n$ let $b_i^\uparrow$ be the constraint defined iteratively as follows. Set $b_2^\uparrow$ to be the constraint $\pi_{\{x_2\}}(c_1)$ and for every $i > 2$, set $b_i^\uparrow$ to be the constraint $\pi_{\{x_i\}}(b_{i-1}^\uparrow \bowtie c_{i-1})$. Note that $S(b_i^\uparrow) = \{x_i\}$ and $R(b_i^\uparrow)$ contains all tuples t with $t[x_i] \in \{i, \dots, n\}$. Moreover, it follows immediately from their defintion that the constraints $b_2^\uparrow, \dots, b_n^\uparrow$ can be computed in time at most $\mathcal{O}(n^4) = \mathcal{O}(|\mathbf{I}_\omega|^4)$ using only join and projection operations.

Similarily, let $b_i^\downarrow$ for every i with $1 \leq i < n$ be the constraint defined recursively as follows. Let $b_{n-1}^\downarrow$ be the constraint $\pi_{\{x_{n-1}\}}(c_{n-1})$ and for every $i < n-1$, we set $b_i^\downarrow$ to be the constraint $\pi_{\{x_i\}}(b_{i+1}^\downarrow \bowtie c_i)$. Note that $S(b_i^\downarrow) = \{x_i\}$ and $R(b_i^\downarrow)$ contains all tuples t with $t[x_i] \in \{1, \dots, i\}$. Moreover, it follows immediately from their definition that the constraints $b_1^\downarrow, \dots, b_{n-1}^\downarrow$ can be computed in time at most $\mathcal{O}(n^4) = \mathcal{O}(|\mathbf{I}_\omega|^4)$ using only join and projection operations.

Note that $b_i^\uparrow \bowtie b_i^\downarrow$ for every i with $1 < i < n$ has scope $\{x_i\}$ and contains only one tuple, i.e., the tuples t with $t[x_i] = i$. Moreover, $b_1^\downarrow$ and $b_n^\uparrow$ also contain only one tuple, i.e., the tuples t with $t[x_1] = 1$ and $t[x_n] = n$, respectively. To use this observation let b_1 be the constraint $b_1^\downarrow$, b_n be the constraint $b_n^\uparrow$ and for every i with $1 < i < n$ let b_i be the constraint $b_i^\uparrow \bowtie b_i^\downarrow$. Then b_i has scope $\{x_i\}$ and $R(b_i)$ contains only one tuple, i.e., the tuple t with $t[x_i] = i$. Moreover, it follows immediately from the definition of the constraints $b_1, \dots, b_n$ that they can be computed from the constraints $b_i^\uparrow, b_i^\downarrow$ in time at most $\mathcal{O}(n^2) = \mathcal{O}(|\mathbf{I}_\omega|^2)$ using only join operations.

Note that at this stage, we have already identified the unique solution of $\mathbf{I}_\omega$; the one that sets every variable x_i to the value i. However, since we have not yet even considered all constraints of $\mathbf{I}_\omega$, we cannot yet bet sure that what we computed is actually a solution of $\mathbf{I}$. To circumvent this caveat, we now compute the constraint $b = b_1 \bowtie \cdots \bowtie b_n$ in time $\mathcal{O}(n)$; note that b has scope V and $R(b)$ only contains the tuple t with $t[x_i] = i$ for every i with $1 \leq i \leq n$. Finally, we join every constraint $c \in C$ with b one-by-one as follows. Assume that $C = \{c^1, \dots, c^{|C|}\}$. Then for every i with $1 \leq i \leq |C|$, we now compute the constraint a_i iteratively by setting: a_1 to be the constraint $b \bowtie c^1$ and for every $i > 1$, a_i is the contraint $a_{i-1} \bowtie c^i$. Now we can be sure that the constraint $a_{|C|}$ contains all solutions (and only the solutions) of $\mathbf{I}$. Since a_i is equal to b for every i it also follows that we can compute the constraints $a_1, \dots, a_{|C|}$ in time $\mathcal{O}(|C|) = \mathcal{O}(|\mathbf{I}_\omega|)$ using only join operations. □

8 Conclusions and Outlook

The main contribution of our paper is the introduction of the notion of a join decomposition and the associated parameter joinwidth (Definitions 2 and 3). These notions are natural as they are entirely based on fundamental operations of relational algebra: joins, projections, and pruning (which can equivalently be stated in terms of semijoins). It is also worth noting that our algorithms seamlessly extend to settings where each variable has its own domain (this can be modeled, e.g., by unary constraints).

We establish several structural and complexity results that put our new notions into context. In particular, we show that:

1. bounded joinwidth captures and properly contains several known restrictions that render CSP tractable (Theorem 13, Propositions 15, 16 and 17).
2. CSP instances of bounded joinwidth can be solved in polynomial time assuming the corresponding join decomposition is provided with the input (Theorem 6);
3. finding a join decomposition of optimal width is NP-hard, already for a constant upper bound on the width (Theorem 7), mirroring the situation surrounding fractional hypertree width [15];
4. CSP instances of bounded joinwidth can be solved by a single-exponential fixed-parameter algorithm parameterized either by the number of constraints (Corollary 19) or the number of variables (Theorem 20);
5. there are instances of bounded joinwidth but unbounded submodular (or adaptive) width (Observation 12); bounded submodular width is the most general hypergraph restriction that allows for fixed-parameter tractability of CSP under the Exponential Time Hypothesis [29];
6. using joins and projections, one can even solve instances of unbounded joinwidth (Theorem 21).

Our results give rise to several interesting directions for future work. We believe that result (2) can be generalized to other problems, such as #CSP or the FAQ-Problem [25]. Result (3) gives rise to the question of whether there exists a polynomial-time approximation algorithm for computing join decompositions of suboptimal joinwidth, similar to Marx's algorithm for fractional hypertree-width [27]. One can also try to develop an efficient heuristic approach for computing the exact joinwidth, possibly similar to the SMT-encoding for fractional hypertree width as recently proposed by Fichte et al. [14].

Result (5) shows that submodular width is not more general than joinwidth. We conjecture that also the converse direction holds, i.e., that the two parameters are actually incomparable. Motivated by result (6), one could try to define a natural parameter that captures the full generality of join and projection operations, or to at least define a parameter that is more general than join decompositions without sacrificing the simplicity of the definition.

References

1. Rafi Ahmed, Rajkumar Sen, Meikel Poess, and Sunil Chakkappen. Of snowstorms and bushy trees. *Proceedings of the VLDB Endowment*, 7(13):1452–1461, 2014.
2. Michael Alekhnovich and Alexander A. Razborov. Satisfiability, branch-width and Tseitin tautologies. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS'02)*, pages 593–603, 2002.
3. Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM J. Comput.*, 42(4):1737–1767, 2013.
4. Andrei A. Bulatov, Peter Jeavons, and Andrei A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM J. Comput.*, 34(3):720–742, 2005.
5. David A. Cohen, Martin C. Cooper, Martin James Green, and Dániel Marx. On guaranteeing polynomially bounded search tree size. In *Principles and Practice of Constraint Programming - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, pages 160–171, 2011.
6. David A. Cohen, Martin C. Cooper, Peter G. Jeavons, and Stanislav Zivny. Binary constraint satisfaction problems defined by excluded topological minors. *Information and Computation*, 264:12–31, 2019.
7. Martin C. Cooper, Aymeric Duchein, Achref El Mouelhi, Guillaume Escamocher, Cyril Terrioux, and Bruno Zanuttini. Broken triangles: From value merging to a tractable class of general-arity constraint satisfaction problems. *Artif. Intell.*, 234:196–218, 2016.
8. Martin C. Cooper and Stanislav Zivny. Hybrid tractable classes of constraint problems. In *The Constraint Satisfaction Problem*, volume 7 of *Dagstuhl Follow-Ups*, pages 113–135. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.
9. Philippe David. Using pivot consistency to decompose and solve functional CSPs. *J. Artif. Intell. Res.*, 2:447–474, 1995.
10. Yves Deville and Pascal Van Hentenryck. An efficient arc consistency algorithm for a class of CSP problems. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence. Sydney, Australia, August 24-30, 1991*, pages 325–330, 1991.
11. Reinhard Diestel. Graph minors 1: A short proof of the path-width theorem. *Combinatorics, Probability & Computing*, 4:27–30, 1995.
12. Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

13. Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
14. Johannes Klaus Fichte, Markus Hecher, Neha Lodha, and Stefan Szeider. An SMT approach to fractional hypertree width. In *Proc. CP'18*, volume 11008 of *Lecture Notes in Computer Science*, pages 109–127. Springer, 2018.
15. Wolfgang Fischl, Georg Gottlob, and Reinhard Pichler. General and fractional hypertree decompositions: Hard and easy cases. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, pages 17–32. ACM, 2018.
16. Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science. An EATCS Series*. Springer Verlag, Berlin, 2006.
17. Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *J. of Computer and System Sciences*, 64(3):579–627, 2002.
18. Georg Gottlob and Stefan Szeider. Fixed-parameter algorithms for artificial intelligence, constraint satisfaction, and database problems. *The Computer Journal*, 51(3):303–325, 2006. Survey paper.
19. Martin Grohe. Parameterized complexity for the database theorist. *SIGMOD Rec.*, 31(4):86–96, 2002.
20. Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1), 2007.
21. Martin Grohe. Logic, graphs, and algorithms. In *Logic and Automata: History and Perspectives*, volume 2 of *Texts in Logic and Games*, pages 357–422. Amsterdam University Press, 2007.
22. Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Trans. Algorithms*, 11(1):4:1–4:20, 2014. URL: `http://doi.acm.org/10.1145/2636918`.
23. Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. of Computer and System Sciences*, 63(4):512–530, 2001.
24. Yannis E. Ioannidis and Younkyung Cha Kang. Left-deep vs. bushy trees: An analysis of strategy spaces and its implications for query optimization. In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data.*, pages 168–177. ACM Press, 1991.
25. Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. FAQ: questions asked frequently. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 13–28. ACM, 2016.
26. Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. What do Shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 429–444. ACM, 2017.
27. Dániel Marx. Approximating fractional hypertree width. *ACM Transactions on Algorithms*, 6(2):Art. 29, 17, 2010.
28. Dániel Marx. Tractable structures for constraint satisfaction with truth tables. *Theory Comput. Syst.*, 48(3):444–464, 2011.
29. Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. ACM*, 60(6):42:1–42:51, 2013.
30. Wady Naanaa. Unifying and extending hybrid tractable classes of csps. *J. Exp. Theor. Artif. Intell.*, 25(4):407–424, 2013.
31. Joakim Alme Nordstrand. Exploring graph parameters similar to tree-width and path-width. Master's thesis, University of Bergen, Department of Informatics, 2017.
32. Neil Robertson and P. D. Seymour. Graph minors. I. Excluding a forest. *J. Combin. Theory Ser. B*, 35(1):39–61, 1983.
33. Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.
34. F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*. Elsevier, 2006.
35. Paul D. Seymour and Robin Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.
36. Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Very Large Data Bases, 7th International Conference, Sep. 9–11, 1981, Cannes, France, Proceedings*, pages 81–94. IEEE Computer Society, 1981.