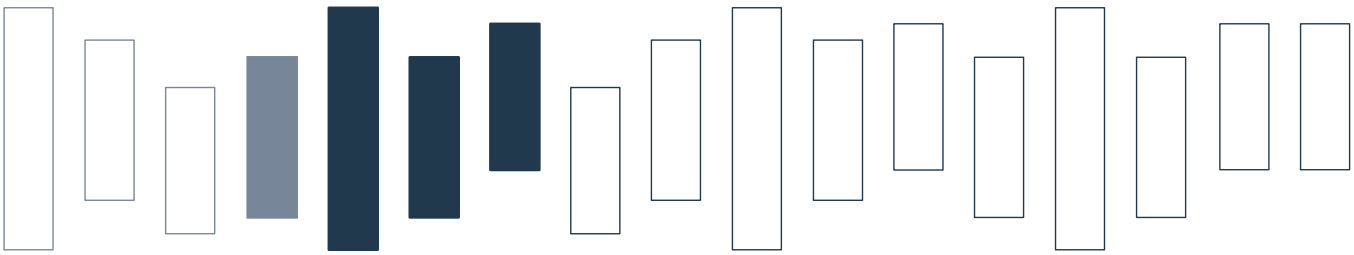
Technical Report AC-TR-18-006

July 2018

An SMT Approach to Fractional Hypertree Width

Johannes K. Fichte, Markus Hecher,
Neha Lodha, and Stefan Szeider



This is the authors' copy of a paper that is to appear the proceedings of CP 2018, the 24th International Conference on Principles and Practice of Constraint Programming, Lille, France, August 27–31, 2018. LNCS 11008, Springer Verlag, 2018.

www.ac.tuwien.ac.at/tr

An SMT Approach to Fractional Hypertree Width*

Johannes K. Fichte[1], Markus Hecher[2], Neha Lodha[3], and Stefan Szeider[3]

[1] International Center of Computational Logic, TU Dresden, Dresden, Germany
fichte@tu-dresden.de
[2] Database and Artificial Intelligence Group, TU Wien, Vienna, Austria
hecher@dbai.tuwien.ac.at
[3] Algorithms and Complexity Group, TU Wien, Vienna, Austria
{neha,sz}@ac.tuwien.ac.at

Abstract. Bounded fractional hypertree width (*fhtw*) is the most general known structural property that guarantees polynomial-time solvability of the constraint satisfaction problem. Bounded *fhtw* generalizes other structural properties like bounded induced width and bounded hypertree width.
We propose, implement and test the first practical algorithm for computing the *fhtw* and its associated structural decomposition. We provide an extensive empirical evaluation of our method on a large class of benchmark instances which also provides a comparison with known exact decomposition methods for hypertree width. Our approach is based on an efficient encoding of the decomposition problem to SMT (SAT modulo Theory) with Linear Arithmetic as implemented in the SMT solver *Z3*. The encoding is further strengthened by preprocessing and symmetry breaking methods. Our experiments show (i) that *fhtw* can indeed be computed exactly for a wide range of benchmark instances, and (ii) that state-of-the art SMT techniques can be successfully applied for structural decomposition.

1 Introduction

A prominent research question is the identification of structural restrictions that make the constraint satisfaction problem (CSP) tractable [10]. Structural restrictions are concerned only in the way how constraints and variables interact, in contrast to language restrictions that are only concerned with the relations that appear in the constraints. Hybrid restrictions are concerned with both aspects.

In his seminal work, Freuder [21] showed that the CSP is tractable under structural restrictions imposed in terms of bounded treewidth of the constraint graph. The following decades brought a phalanx of results that identified more and more general structural restrictions that still guarantee polynomial-time tractability of the CSP, some prominent notions are spread-cut width [11] and hypertree width [24]. This line of research found its culmination point in the work of Grohe and Marx [27,28], who introduced the notion of *fractional hypertree width*, which generalizes all known structural restrictions that guarantee polynomial-time tractability of the CSP.

* The work has been supported by the Austrian Science Fund (FWF), Grants Y698 and P26696, and the German Science Fund (DFG), Grant HO 1294/11-1. Fichte and Hecher are also affiliated with the University of Potsdam, Germany.

So far, fractional hypertree width was mostly of theoretical interest, because of the lack of practical algorithms for actually computing the associated decompositions. One can approximate the fractional hypertree width in polynomial time with a cubic error factor [37]. This is prohibitive for practical applications, since CSP algorithms that exploit (fractional) hypertree decompositions are exponential in time and space in the width of the decomposition [11,24,27,28]. It is unlikely that one could compute the exact fractional hypertree width in polynomial time as checking whether a hypergraph has fractional hypertree width $\leq w$ is already NP-hard for $w = 2$ [20].

Contributions. In this paper we propose, implement and test the first practical approach to compute the fractional hypertree width. Our approach is based on an efficient SMT-encoding of the problem, and utilizes preprocessing and symmetry breaking methods. We establish an *ordering-based* characterization of fractional hypertree-width which is similar to the well-known elimination order characterization of treewidth (see, e.g., [8,13]), which traces back to the work of Rose [39]. Ordering-based characterizations of treewidth have been shown to be well-suited for SAT encodings of treewidth and related width measures [4,7,36,42], hence it was promising to establish such a characterization also for fractional hypertree width. This indeed turned out to be both feasible as well as effective. In fact, to encode the linear ordering as well as the hyperedges induced by the ordering, we could utilize the very same Boolean variables and constraints that have been used for treewidth encodings. However, for treewidth one needs to bound the cardinalities of certain sets of vertices, which in the existing encodings was accomplished by SAT-based cardinality constraints or Max-SAT formulations. For fractional hypertree width, however, we need to find certain *real-valued weights* of hyperedges and enforce lower and upper bounds on the sums of weights of certain sets of hyperedges. We found that these constraints can be handled well by the SAT modulo Theory (SMT) framework, in particularly by SMT with Linear Arithmetic as implemented in the state-of-the-art SMT solver *Z3* [38]. On top of the SMT encoding we also developed various *preprocessing* and *symmetry breaking* methods.

We would like to point out that for CSP instances of bounded fractional hypertree width, one can not only decide satisfiability, but also count the number of satisfying assignments in polynomial time, as observed by Duran and Mengel [15]. Hence also from a complexity theoretic point of view it is justified to use an SMT solver which operates in the class NP to facilitate the solution of a harder #P-complete counting problem.

We implemented our methods creating the prototype tool *FraSMT* and performed extensive experiments on benchmark instances which contain real-world instances from various application domains. To the best of our knowledge, there have not been any practical algorithms for fractional hypertree width reported in the literature. Thus we took as a reference point the algorithm *det-k-decomp* of Gottlob and Samer [26] for the related (but less general) parameter hypertree width, which in turn was shown to outperform the algorithm *opt-k-decomp* proposed earlier by Gottlob et al. [25].

Our results show that on an extensive collection of benchmark instances the new SMT approach clearly outperforms the known algorithm *det-k-decomp*, even without preprocessing or symmetry breaking. Adding these techniques gives again a significant performance boost.

In summary, our findings are significant as they show (i) that fractional hypertree width can indeed be computed for a wide range of benchmark instances, and (ii) that SMT techniques can be successfully applied for structural decomposition and outperform known methods.

2 Preliminaries

A *hypergraph* is a pair $H = (V(H), E(H))$, consisting of a set $V(H)$ of *vertices* and a set $E(H)$ of *hyperedges*, each hyperedge being a subset of $V(H)$.

For a hypergraph $H = (V, E)$ and a vertex $v \in V$, we write $E_H(v) = \{\, e \in E \mid v \in e \,\}$ and $N_H(v) = (\cup E_H(v)) \setminus \{v\}$; the latter set is the *neighborhood* of v. If $u \in N_H(v)$ we say that u and v are *adjacent*.

The *hypergraph* $H - v$ is defined by $H = (V \setminus \{v\}, \{e \setminus \{v\} \mid e \in E\})$.

The *primal graph* (or 2-*section*) of a hypergraph $H = (V, E)$ is the graph $P(H) = (V, E_{P(H)})$ with $E_{P(H)} = \{\, \{u, v\} \mid u \neq v$, there is some $e \in E$ such that $\{u, v\} \subseteq e \,\}$.

Consider a hypergraph $H = (V, E)$ and a set $S \subseteq V$. An *edge cover* of S is a set $F \subseteq E$ such that for every $v \in S$ there is some $e \in F$ with $v \in e$. A *fractional edge cover* of S (with respect to H) is a mapping $\gamma : E \to [0, 1]$ such that for every $v \in S$ we have $\sum_{e \in E,\, v \in e} \gamma(e) \geq 1$. The *weight* of γ is defined as $\sum_{e \in E} \gamma(e)$. The *fractional edge cover number* of S with respect to a hypergraph H, denoted $\mathit{fn}_H(S)$, is the minimum weight over all its fractional edge covers with respect to H.

A *tree decomposition* of a hypergraph $H = (V, E)$ is a pair $\mathcal{T} = (T, \chi)$ where $T = (V(T), E(T))$ is a tree and χ is a mapping that assigns each $t \in V(T)$ a set $\chi(t) \subseteq V$ (called the *bag* at t) such that the following properties hold:

- for each $v \in V$ there is some $t \in V(T)$ with $v \in \chi(t)$ ("v is covered by t"),
- for each $e \in E$ there is some $t \in V(T)$ with $e \subseteq \chi(t)$ ("e is covered by t"),
- for any three $t, t', t'' \in V(T)$ where t' lies on the path between t and t'', we have $\chi(t') \subseteq \chi(t) \cap \chi(t'')$ ("bags containing the same vertex are connected").

The width of a tree decomposition $\mathcal{T}$ of H is the size of a largest bag of $\mathcal{T}$ minus 1. The treewidth $\mathit{tw}(H)$ of H is the smallest width over all its tree decompositions.

We will frequently use the following well-known fact (see, e.g. [9]).

Fact 1 *Let (T, χ) be a tree decomposition of a graph G and K a clique in G, then there exists a node $t \in V(T)$ with $V(K) \subseteq \chi(t)$.*

Using this fact it is easy to see that $\mathit{tw}(H) = \mathit{tw}(P(H))$ holds for every hypergraph H.

A *generalized hypertree decomposition* of H is a triple $\mathcal{G} = (T, \chi, \lambda)$ where (T, χ) is a tree decomposition of H and λ is a mapping that assigns each $t \in V(T)$ an *edge cover* $\lambda(t)$ of $\chi(t)$. The *width* of $\mathcal{G}$ is the size of a largest edge cover $\lambda(t)$ over all $t \in V(T)$. A *hypertree decomposition* is a generalized hypertree decomposition that satisfies a certain additional property which was added in order to make the computation of the decomposition tractable [24]. The *generalized hypertree width* $\mathit{ghtw}(H)$ of H is the smallest width over all generalized hypertree decompositions of H. The *hypertree width* $\mathit{htw}(H)$ is the smallest width over all hypertree decompositions of H.

A *fractional hypertree decomposition* of H is a triple $\mathcal{F} = (T, \chi, \gamma)$ where (T, χ) is a tree decomposition of H and γ is a mapping that assigns each $t \in V(T)$ a fractional edge cover $\lambda(t)$ of $\chi(t)$ with respect to H. The *width* of $\mathcal{F}$ is the largest weight of the fractional edge covers $\lambda(t)$ over all $t \in V(T)$. The fractional hypertree width $\mathit{fhtw}(H)$ of H is the smallest width over all fractional hypertree decompositions of H.

To avoid trivial cases, we consider only hypergraphs $H = (V, E)$ where $E_H(v) \neq \emptyset$ for all $v \in V$. Consequently, every considered hypergraph H has a (fractional) edge cover and $\mathit{fhtw}(H)$ is always defined. If $|V| = 1$ then $\mathit{fhtw}(H) = 1$.

Since an edge cover can be seen as the special case of a fractional edge cover, with weights restricted to $\{0, 1\}$, it follows that for every hypergraph H we have $\mathit{fhtw}(H) \leq \mathit{ghtw}(H) \leq \mathit{htw}(H) \leq \mathit{tw}(P(H))$.

3 Ordering-Based Characterization of Fractional Hypertree Width

The first SAT encoding of treewidth was suggested by Samer and Veith [42], it uses an ordering-based characterization of treewidth. Also more recent SAT encodings of treewidth are ordering-based [4,7]. In view of the success of ordering-based characterizations of treewidth, we developed an ordering-based characterization of fractional hypertree width, and used it for our SMT encoding. The remainder of this section is devoted to the definition of this characterization and a proof of its correctness. Kamis, et al. [33] have suggested a similar characterization.

Let $H = (V, E)$ be a hypergraph with $n = |V|$ and $L = (v_1, \ldots, v_n)$ a linear ordering of the vertices of H. We define the *hypergraph induced by* L as $H_L^n = (V, E^n)$ where E^n is obtained from E by adding hyperedges successively as follows. We let $E^0 = E$, and for $1 \leq i \leq n$ we let $E^i = E^{i-1} \cup \{e_i\}$ where $e_i = \{\, v \in \{v_{i+1}, \ldots, v_n\} \mid$ there is some $e \in E^{i-1}$ containing v and $v_i \,\}$. We consider the binary relation $\mathit{Arc}_L = \{\, (v_i, v_j) \in V \times V \mid i < j$ and v_i and v_j are adjacent in $H_L^n \,\}$. We write $\mathit{Arc}_L(i) = \{v_i\} \cup \{\, v_j \mid (v_i, v_j) \in \mathit{Arc}_L \,\}$, hence $\mathit{Arc}_L(i) = \{v_i\} \cup e_i$.

The *fractional hypertree width of H with respect to a linear ordering L*, denoted $\mathit{fhtw}_L(H)$, is the largest fractional edge cover number with respect to H over all the sets $\mathit{Arc}_L(i)$, i.e.,

$$\mathit{fhtw}_L(H) = \max_{i=1}^{n} \mathit{fn}_H(\mathit{Arc}_L(i)).$$

We would like to emphasize that in this definition the fractional covers are considered with respect to the original hypergraph H, and not with respect to the induced hypergraph H_L^n.

Figure 1 illustrates these concepts on a small example.

Theorem 1. *The fractional hypertree width of a hypergraph H equals the smallest fractional width over all its linear orderings, i.e., $\mathit{fhtw}(H) = \min_L \mathit{fhtw}_L(H)$.*

We establish the theorem by means of two lemmas below. Before doing so, we introduce some additional terminology.

Let $H = (V, E)$ be a hypergraph and $E' \subseteq E$ an edge cover of H. An *E'-fractional* hypertree decomposition of H is a fractional hypertree decomposition $\mathcal{F} = (T, \chi, \gamma)$

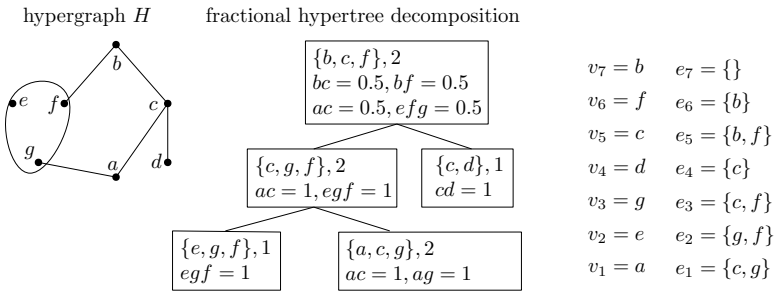


Fig. 1. An example illustrating a fractional hypertree decomposition of width 2 as well as the hyperedges e_i for $1 \leq i \leq 7$.

of H where each fractional cover $\gamma(t)$ assigns edges $e \in E \setminus E'$ the value 0. Similarly, the *E'-fractional* hypertree width of H with respect to a linear ordering L, denoted $\mathit{fhtw}_L(E', H)$, is computed by using only fractional edge covers that assign edges $e \in E \setminus E'$ the value 0, i.e.,

$$\mathit{fhtw}_L(E', H) = \max_{i=1}^{n} \mathit{fn}_{(V,E')}(\mathit{Arc}_L(i)).$$

The proof of the following lemma provides a decoding algorithm that efficiently computes a fractional hypertree decomposition from a given ordering.

Lemma 1. *Let $H = (V, E)$ be a hypergraph, $L = (v_1, \ldots, v_n)$ a linear ordering of V, and $E' \subseteq E$ an edge cover of H. Then H has an E'-fractional hypertree decomposition of width $\leq \mathit{fhtw}_L(E', H)$.*

Proof. We proceed by induction on n. If $n = 1$ the statement is trivially true. Now assume $n > 0$ and that the statement holds for all smaller n. Let $w = \mathit{fhtw}_L(E', H)$. Let $e_1, \ldots, e_n$ and $S_1, \ldots, S_n$ as in the definition of a fractional hypertree width of H with respect to the linear ordering L.

We obtain from H the hypergraph H_2 by deleting v_1 and adding the hyperedge e_1. Furthermore, we obtain from E' the edge cover $E'_2 \subseteq E(H_2)$ of H_2 by removing v_1 from every edge in E'.

Now $L_2 = (v_2, \ldots, v_n)$ is a linear ordering of H_2, and we observe that its width cannot be larger than the width of L, since the sequence of sets $\mathit{Arc}_{L_2}(i)$ for $1 \leq i \leq n-1$ is exactly the same as the sequence of sets $\mathit{Arc}_L(i)$ for $2 \leq i \leq n$. Hence $\mathit{fhtw}_{L_2}(E'_2, H_2) \leq \mathit{fhtw}_L(E', H) = w$.

By induction hypothesis, it follows that H_2 has an E'_2-fractional hypertree decomposition $\mathcal{F}_2 = (T_2, \chi_2, \gamma_2)$ of width $\leq w$. By definition of a tree decomposition, there must be a node $t_2 \in V(T_2)$ such that $e_1 \subseteq \chi_2(t_2)$. We define an E'-fractional hypertree decomposition $\mathcal{F} = (T, \chi, \gamma)$ of H as follows.

1. We obtain T by adding a new node t_1 to T_2 and making it adjacent with t_2.
2. We set $\chi(t_1) = \{v_1\} \cup e_1 = S_1$ and $\chi(t) = \chi_2(t)$ for all other tree nodes t.

3. We choose for $\gamma(t_1)$ an E'-fractional edge cover of S_1 of smallest weight, which must be $\leq w$ since L was assumed to have weight w, and we set $\gamma(t) = \gamma_2(t)$ for all other tree nodes t.

We observe that (T, χ) satisfies all conditions of a tree decomposition, and conclude that $\mathcal{F}$ is indeed an E'-fractional hypertree decomposition of H of width $\leq w$. □

Lemma 2. *Let $H = (V, E)$ be a hypergraph, $E' \subseteq E$ an edge cover of H and $\mathcal{F} = (T, \chi, \gamma)$ an E'-fractional hypertree decomposition of H of width w. Then there is a linear ordering $L = (v_1, \ldots, v_n)$ of V such that $\mathit{fhtw}_L(E', H) \leq w$.*

Proof. As above we proceed by induction on n. We observe that the statement is trivially true if $n = 1$ or $|V(T)| = 1$. Now assume $n > 0$, $|V(T)| > 1$, and that the statement holds for all smaller n.

W.l.o.g., we may assume that for each leaf t of T there is some $v \in \chi(t)$ that does not belong to $\chi(t')$ for any other node $t' \in V(T) \setminus \{t\}$. Namely, if such a $v \in \chi(t)$ does not exist, then the properties of a tree decomposition imply that $\chi(t) \subseteq \chi(t'')$ for the unique neighbor t'' of t in T, and so all vertices and hyperedges covered at node t are also covered at node t'', and t can be omitted.

Based on the above assumption, we conclude that there must be some $v_1 \in V$ which belongs to $\chi(t)$ for a leaf t of T, but v_1 does not belong to $\chi(t')$ for any other node $t' \in V(T) \setminus \{t\}$.

Let $e_1 = \{\, v \in \{v_2, \ldots, v_n\} \mid \text{there is some } e \in E \text{ containing } v \text{ and } v_i \,\}$ and $S_1 = \{v_1\} \cup e_1$ (as in the definition of fractional hypertree width of H with respect to the linear ordering). Since $S_1 \subseteq \chi(t)$, $\gamma(t)$ gives an E'-fractional cover of S_1 with respect to H of weight $\leq w$, hence $\mathit{fn}_{(V,E')}(S_1) \leq w$.

We obtain the hypergraph $H_2 = (V_2, E_2)$ where $V_2 = V \setminus \{v_1\}$ and $E_2 = \{\, e \setminus \{v_1\} \mid e \in E \,\} \cup \{e_1\}$. We also define $E_2' = \{\, e \setminus \{v_1\} \mid e \in E' \,\}$ which is an edge cover of H_2. It is easy to see that from $\mathcal{F}$ we can obtain an E_2'-fractional hypertree decomposition $\mathcal{F}_2 = (T, \chi_2, \gamma_2)$ of H_2 of width $\leq w$ as follows.

1. We define $\chi_2(t) = \chi(t) \setminus \{v_1\}$, and $\chi_2(t') = \chi(t')$ for all other tree nodes t.
2. For every a hyperedge $e_2 \in E_2'$ we let $\gamma_2(t)[e_2] = \max\{\, \gamma(t)[e_1 \cup \{v_1\}] \mid e_1 \cup \{v_1\} \in E' \,\} \cup \{\, \gamma(t)[e_1] \mid e_1 \in E' \,\}$.

The induction hypothesis applies and hence we can conclude that there exists a linear ordering $L_2 = (v_2, \ldots, v_n)$ of $V(H_2)$ such that $\mathit{fhtw}_{L_2}(E_2', H_2) \leq w$. We now extend L_2 by adding v_1 at the first position and obtain the ordering $L = (v_1, \ldots, v_n)$. We have already observed above that $\mathit{fn}_{(V,E')}(S_1) \leq w$, hence $\mathit{fhtw}_L(E', H) \leq w$. □

Theorem 1 now follows by Lemmas 1 and 2 by taking $E' = E$.

4 SMT Encoding

In this section we describe an SMT encoding for the characterization of fractional hypertree decompositions as given in the previous section. The encoding is an adaptation of the Samer-Veith encoding of treewidth [42]. Given a hypergraph $H = (V, E)$ with

$V = \{v_1, \ldots, v_n\}$, we produce a formula $F(H, w)$ which is satisfiable if and only if the hypergraph V has a linear ordering L of V such that $\mathit{fhtw}_L(H) \leq w$.

The relation Arc_L can be computed in exactly the same way as Samer and Veith compute the "graph induced by the ordering." We therefore use the same notation and introduce Boolean *ordering variables* $o_{i,j}$ for $1 \leq i < j \leq n$ and Boolean *arc variables* $a_{i,j}$ for $1 \leq i, j \leq n$.

An ordering variable $o_{i,j}$ is true if and only if $i < j$ and v_i precedes v_j in L. Consequently, to enforce that L is indeed a linear ordering, we must ensure transitivity, which can be accomplished with the following clauses (here $o^*(i,j)$ stands for $o(i,j)$ if $i < j$ and $\neg o(j,i)$ otherwise):

$$[\neg o^*(i,j) \vee \neg o^*(j,k) \vee o^*(i,k)] \qquad \text{for } 1 \leq i,j,k \leq n \text{ and } i,j,k \text{ are distinct.}$$

The arc variables are used to represent the relation Arc_L for the ordering L represented by the ordering variables, where $a(i,j)$ is true if and only if $(v_i, v_j) \in \mathit{Arc}_L$, i.e., if $v_j \in \mathit{Arc}_L(i)$.

A straightforward encoding of the definitions of Arc_L gives rise to the following clauses:

$$[\neg o(i,j) \vee a(i,j)] \wedge [o(i,j) \vee a(j,i)] \qquad \text{for } \{v_i, v_j\} \in E(P(H)) \text{ and } i < j,$$

$$[\neg a(i,j) \vee \neg a(i,l) \vee \neg o(j,l) \vee a(j,l)] \wedge [\neg a(i,j) \vee \neg a(i,l) \vee o(j,l) \vee a(l,j)] \qquad \text{for } 1 \leq i,j,l \leq n,\ i \neq j,\ i \neq l, \text{ and } j < l,$$

$$[\neg a(i,j) \vee \neg a(i,l) \vee a(j,l) \vee a(l,j)] \qquad \text{for } 1 \leq i,j,k \leq n,\ i \neq j,\ i \neq k \text{ and } j < k,$$

$$[\neg a(i,i)] \qquad \text{for } 1 \leq i \leq n.$$

Now, instead of cardinality constraints as used for treewidth, we use here real-valued weight variables representing the fractional covers. In fact, this makes the overall SMT encoding for fractional hypertree width even simpler and more compact than the SAT encoding for treewidth.

More precisely, we introduce a *weight variable* $w(i,e)$ for each $1 \leq i \leq n$ and $e \in E$, representing the weight of e in a fractional edge cover $\gamma_L(i)$ of the set $\mathit{Arc}_L(i)$, where L is the ordering represented by the ordering variables.

To ensure that $\gamma_L(i)$ is indeed a fractional edge cover of $\mathit{Arc}_L(i)$, we add the following two constraints; the first checks that all the vertices in $\mathit{Arc}_L(i) \setminus \{v_i\}$ are covered by $\gamma_L(i)$, the second checks that v_i is covered by $\gamma_L(i)$:

$$[\neg a(i,j) \vee \textstyle\sum_{e \in E_H(v_j)} w(i,e) \geq 1] \qquad \text{for } 1 \leq i \neq j \leq n,$$

$$[\textstyle\sum_{e \in E_H(v_i)} w(i,e) \geq 1] \qquad \text{for } 1 \leq i \leq n.$$

Finally, we ensure that the weights of the fractional covers $\gamma_L(i)$ are at most w, $1 \leq i \leq n$, by means of the following constraints:

$$[\textstyle\sum_{e \in E} w(i,e) \leq w] \qquad \text{for } 1 \leq i \leq n.$$

This completes the construction of the formula $F(H, w)$. The formula $F(H, w)$ has $\mathcal{O}(n(n+m))$ variables where $\mathcal{O}(n^2)$ are Boolean variables and $\mathcal{O}(nm)$ are real variables, and $\mathcal{O}(n^3)$ clauses, where only $\mathcal{O}(n^2)$ are used for restricting the width.

In view of the construction of the formula and by Theorem 1 we obtain the following result.

Theorem 2. *A hypergraph H has fractional hypertree width $\leq w$ if and only if $F(H, w)$ is satisfiable.*

In view of the remark from the end of Section 2, we conclude that by replacing the real variables with integer variables yields an encoding for generalized hypertree width.

5 Preprocessing

In this section, we formulate several preprocessing methods. Some of them originate in the context of treewidth [4] and we adapted them for our purposes. It turned out that in some cases the preprocessing techniques decrease the encoding size significantly. This not only speeds up the solving process but also extends the scope of our method to larger instances.

We exhaustively apply the following preprocessing rules R1–R4 in their order of occurrence.

R1: Contained Hyperedges A hyperedge that is a subset of another hyperedge can be safely removed.

Proposition 1. *Let $H = (V, E)$ be a hypergraph, $e, f \in E$ be hyperedges such that $e \subsetneq f$, then fhtw$(H) =$ fhtw$((V, E \setminus \{e\}))$.*

Proof. Consider a fractional hypertree decomposition $\mathcal{F} = (T, \chi, \lambda)$ of H with $\lambda(e) > 0$. We define a fractional hypertree decomposition $\mathcal{F}' = (T, \chi, \lambda')$ of the same width by setting $\lambda'(e') = \lambda(e')$ for $e' \in E \setminus \{e, f\}$ and $\lambda'(f) = \lambda(f) + \lambda(e)$. □

R2: Biconnected Components A hypergraph H is *connected* if for any two vertices $u, v \in V$ there exist vertices $v_1, \ldots, v_k \in V$ such that $u = v_1$, $v = v_k$ and v_i and v_{i+1} are adjacent in H for $1 \leq i \leq k - 1$. H is *biconnected* if $H - v$ is connected for every $v \in V$. A *biconnected component* of H is a maximal biconnected hypergraph $H' = (V', E')$ with $V' \subseteq V$ and $E' \subseteq E$. Observe that two biconnected components of H can have at most one vertex in common.

We can split any hypergraph into biconnected components and compute the fractional hypertree width of each component separately.

Proposition 2. *Let H be a hypergraph and $H_1, \ldots, H_\ell$ its biconnected components. Then* fhtw$(H) = \max_{i=1}^{k}$ fhtw(H_i).

We omit the easy proof due to space restrictions.

R3: Deletion of Vertices of Degree 1 A vertex of degree 1 (i.e., a vertex occurring in only one hyperedge) can be safely deleted.

Proposition 3. *Let $H = (V, E)$ be a hypergraph and $v \in V$ be a vertex of degree one, i.e., $|E_H(v)| = 1$, and* fhtw$(H - v) \geq 1$. *Then* fhtw$(H) =$ fhtw$(H - v)$.

Proof. We know that $\mathit{fhtw}(H) \geq \mathit{fhtw}(H - v)$. For showing $\mathit{fhtw}(H) \leq \mathit{fhtw}(H - v)$, we take a fractional hypertree decomposition $\mathcal{F} = (T, \chi, \lambda)$ of $H - v$ and modify $\mathcal{F}$ to obtain a fractional hypertree decomposition $\mathcal{F}' = (T', \chi', \lambda')$ of H. In particular, there has to exist node t in T with $\chi(t) = e \setminus \{v\}$, where $e \in E$ such that $v \in e$. Then, we construct $\mathcal{F}'$ by taking $\mathcal{F}$, adding a fresh node t' as a child node of t to T', and assigning $\chi(t') = e$ and $\lambda'(t') = 1$. Since $\mathit{fhtw}(H) \geq 1$, the claim sustains. □

R4: Simplicial Vertices Let $H = (V, E)$ be a hypergraph. A vertex $v \in V$ is a *simplicial vertex* of H if the neighborhood of v forms a clique in the primal graph of H.

We can remove a simplicial vertex v as long we maintain $\mathit{fn}_H(N_H[v] \cup \{v\})$ as a lower bound for the fractional hypertree width.

Proposition 4. *Let $H = (V, E)$ be a hypergraph and v a simplicial vertex of H. Then, $\mathit{fhtw}(H) = \max(\mathit{fhtw}(H - v), \mathit{fn}_H(N_H[v] \cup \{v\}))$.*

Proof. We proceed similarly to the proof of Proposition 3, where we modified a fractional hypertree decomposition $\mathcal{F}$ for $H - v$ in order to obtain one for H. Here, however, the fresh decomposition node t' contains $N_H[v] \cup \{v\}$ in its bag. □

6 Symmetry Breaking and Lower Bounds with Cliques

In this section we present the utilization of cliques in the primal graph for two purposes. First, we can choose any clique (i.e., a complete subgraph) in the primal graph and put the vertices of the clique at the end of the ordering. This can be seen as a symmetry breaking method that decreases the search space. In particular, it helps to speed up the optimality check (i.e., the $F(H, w)$ call when $w = \mathit{fhtw}(H) - 1$), as here the full search space needs to be explored. A similar technique has previously been used for a SAT encoding of treewidth [4].

For a hypergraph $H = (V, E)$ we call a set $S \subseteq V$ a *hyperclique* if S is a complete subgraph of the primal graph $P(H)$.

The next proposition ensures that we can indeed force a hyperclique to be the last in the ordering without effecting the fractional hypertree width.

Proposition 5. *Let $H = (V, E)$ and be a hypergraph and $S = \{v_1, \ldots, v_\ell\}$ a hyperclique in H. Then, there is an ordering $L = (\ldots, v_1, \ldots, v_\ell)$ in which the vertices of S appear at the end, such that $\mathit{fhtw}_L(H) = \mathit{fhtw}(H)$.*

Proof. Let $\mathcal{F} = (T, \chi, \lambda)$ be a fractional hypertree decomposition of H of width $\mathit{fhtw}(H)$. By Fact 1 there is a node t in T with $S \subseteq \chi(t)$. We consider T to be rooted in t and construct a linear ordering L according to the proof of Lemma 2. Since we always pick vertices from bags from leaves of T, we will be left with t as the last tree node, and hence the vertices from $\chi(t)$ will be picked last. As a result, we obtain an ordering L, where vertices V' appear at the end and $\mathit{fhtw}_L(H) = \mathit{fhtw}(H)$. □

Hypercliques can also be used to obtain a lower bound on the fractional hypertree width. If we want to compute the treewidth of a graph, and we know that the graph contains a clique on k vertices, then by Fact 1 we immediately know that the treewidth of

the graph must be at least $k-1$. However, in the context of hypergraphs and fractional hypertree width, we need to take the fractional edge cover number of the clique into account. Consider for instance a hypergraph $H = (V, \{V\})$. It is easy to see that $\mathit{fhtw}(H) = 1$, although V forms a hyperclique. However, we still can show the following:

Proposition 6. *Let $H = (V, E)$ be a hypergraph and S a hyperclique of H. Then, $\mathit{fhtw}(H) \geq \mathit{fn}_H(S)$.*

Proof. Assume any fractional hypertree decomposition $\mathcal{F} = (T, \chi, \lambda)$ of H. Since S is a hyperclique, Fact 1 provides us a node t in $\mathcal{F}$ whose bag contains S, i.e., $\chi(t) \supseteq S$. Then, by definition of fractional hypertree decompositions, every vertex of S is covered in t. As a result, the weight $\lambda(t)$ is at least $\mathit{fn}_H(S)$, and $\mathit{fhtw}(H) \geq \mathit{fn}_H(S)$. □

We performed some experiments that suggest that the symmetry breaking works better with a hyperclique S with large $\mathit{fn}_H(S)$ than with a hyperclique S' where this number is small (e.g., a clique that is contained in a single hyperedge), even when S' is larger than S. Hence a hyperclique S with large $\mathit{fn}_H(S)$ serves two purposes: it facilitates symmetry breaking and provides us with a lower bound on the fractional hypertree width. However, the computation of a hyperclique S where $\mathit{fn}_H(S)$ is maximal is a very hard problem, hence we propose the notion of a k-hyperclique as a compromise.

A hyperclique S of a hypergraph $H = (V, E)$ is a *k-hyperclique* if no hyperedge of H intersects with S in more than k vertices. Intuitively, small values of k prevent large hyperedges, whereas bigger values provides us with flexibility, resulting in potentially larger cliques.

As already discussed, for symmetry breaking we rely on appropriate cliques. We aim to (i) fix parts of the ordering L of all the vertices of the clique (preferably large), (ii) to influence other bags as much as possible. The chances of L influencing other bags increase when we do not have large hyperedges, i.e., one hyperedge does not cover all the vertices of a large clique. Therefore, there is a tradeoff between finding large cliques, and avoiding large hyperedges.

In order to address this tradeoff we compute maximum cardinality k-hyperclique. We can detect a k-hyperclique of size at least ℓ for given hypergraph $H = (V, E)$ by means of a SAT encoding. Here k is assumed to be a small constant. For each vertex v we introduce a Boolean variable x_v, which is true if v belongs to the k-hyperclique. We then add the following constraints:

$[\neg x_{v_1} \vee \neg x_{v_2}]$	for any two vertices $v_1, v_2 \in V$ with $v_2 \notin N[v_1]$;
$[\neg x_{v_1} \vee \cdots \vee \neg x_{v_k}]$	for any k vertices $v_1, \ldots, v_k$ belonging to hyperedge $e \in E$;
$[\sum_{v \in V} x_v \geq \ell]$	cardinality constraint for enforcing clique size at least ℓ.

In the next section we will provide more details on how we have implemented the search for k-hypercliques.

7 Experimental Work

We performed a series of experiments on various publicly available benchmark sets, in order to obtain the fractional hypertreewidth of these instances, to evaluate whether our

SMT-based approach fits well to obtain exact values on the width, and to investigate how well our approach scales. The source code of our SMT-based decomposer[4], benchmarks, and detailed results[5] are publicly available.

7.1 Implementation

We implemented our encoding into our prototypical decomposer *FraSMT*. We used *Python 2.7.14* [40] based on an Anaconda[6] distribution, which includes dependency handling for binaries packages. We used the graph library *networkX 2.1* [30], the answer-set programming solver *clingo 5.2.2* (*gringo 5.2.2* and *clasp 3.3.3*) [22], and the SMT solver *Z3 4.6.2* [38]. Our implementation consists of two separate tools: a validator and a decomposer.

Validator. The first part is a reusable validator that validates computed fractional hypertree decompositions or related decompositions such as tree decompositions and hypertree decompositions. The validator takes as input an extended version of the format used for the treewidth track of the Parameterized Algorithms and Computational Experiments Challenge (PACE) [14]. Since the graph library *networkX* does not support hypergraphs, we implemented hypergraph classes and classes that allow for a primal graph view on such a hypergraph. Both classes implement a *networkX*-like hypergraph API. Although it suffices to use rational numbers to compute the fractional hypertree width [27,28], we still represent the maximum width by a real value since *Z3* does neither support rationals nor reals of arbitrary precision. As we may have a precision loss due to the internal representation of the reals [12], we check for width $w + \varepsilon$ for some small $\varepsilon \geq 0$. By default we set ε to 0.001.

Decomposer and its Configurations. The second and main part is the decomposer *FraSMT*, which implements the preprocessing techniques, the SMT encoding, invoking the SMT solver, as well as reconstructing a decomposition from the solver assignments and outputting a decomposition (if possible; for details see below). Our decomposer *always* reduces contained hyperedges and splits a hypergraph into biconnected components and computes the width of each component separately. We optionally run finding and deleting degree 1 vertices as well as simplicial vertices. We refer to configurations that include this preprocessing with a string that contains "P", whereas "p" indicates that this preprocessing technique is disabled (see Table 1). Further, our decomposer computes as a preprocessing step large k-hypercliques for symmetry breaking and for obtaining lower bounds, as discussed above. For this task we employ the answer-set programming (ASP) solver *clingo*, which supports (implicit) incremental solving and unsatisfiable core shrinking [1]. We use this to aim at a k-hyperclique of maximum cardinality. However, we limited the solving time (ten seconds) to determine such a clique. Still, at any time during the optimization (as long as at least one k-hyperclique has been computed), the solver is able to provide a large k-hyperclique. The ASP solver supports a natural

[4] See: github.com/daajoe/frasmt
[5] See: Benchmark repository [16] and results/raw data [17].
[6] See: https://conda.io/docs/user-guide/install/download.html

encoding of cardinality constraints, and allows for incrementally computing maximum cardinality cliques among all k-hypercliques for $3 \leq k \leq \ell$ for some fixed ℓ. We thereby start with $k = \ell$ and then proceed (aiming at better lower bounds) by incrementally decreasing k by adding the necessary constraints to the ASP solver in multiple shots [22]. We then use such a resulting large hyperclique to apply symmetry breaking in our encoding as described in Proposition 5 and we use hypercliques to obtain additional lower bounds for the encoding. Moreover, we take the maximum width over the previously computed components and feed this value into the next computation. In that way we might obtain unsatisfiability and cannot output a decomposition, however, we cut the search space for the SMT solver as the solver does not necessarily need to find an exact solution in order to avoid an easy-hard-easy behavior. In the following configurations we use symmetry breaking as well as lower bounds. Finally, we implemented the encoding via a direct *Python* interface to the solver using additional features of *Z3*.

Other Solvers. In order to obtain results for hypertree width of our instances, we used a backtracking-based implementation *det-k-decomp* by Gottlob and Samer [26]. Since this implementation can only check for hypertree width of size at most k of an instance, we added a simple progression step on top[7], which for every iteration reduces the result of *det-k-decomp* by 1 to check optimality.

7.2 Benchmark Instances

We considered a selection of 2191 instances, which contain hypergraphs that originate from CSP instances and conjunctive database queries from various sources. The hypergraphs contain up to 2993 vertices and 2958 hyperedges. The first set `Daimler-Chrysler` consists of 15 instances, the second set `Grid2D` consists of 12 instances, and the third set `ISCAS'89` consists of 24 instances on circuits [26]. Moreover, the benchmarks contain 35 instances in the set `MaxSAT` [6] and two sets (`csp_application` and `csp_random`) of instances from the well known XCSP benchmarks [3] with less than 100 constraints such that all constraints are extensional. The set `csp_application` contains 1090 instances and the set `csp_random` contains 863 instances. Further, the set `csp_other` contains 82 instances, which have been collected for works on hypertree decompositions[8]. The set `CQ` consists of 156 instances from various conjunctive queries [2,5,23,29,34,43]. About a quarter of the instances are graphs. Although *fhtw* and *tw* coincide on graphs, these instances are still well-suited as benchmarks as they provide a challenge for the decomposer. All instances have been collected by Fischl et al. [19] (publicly available at [16]). We gratefully acknowledge him for providing this large collection of instances.

7.3 Benchmark Setting

Hardware. Our results were gathered on Ubuntu 16.04 LTS Linux machines kernel 4.13.0-3 on GCC 5.4.1, both post-Spectre and post-Meltdown kernels[9]. We ran

[7] github.com/daajoe/detkdecomp
[8] https://www.dbai.tuwien.ac.at/proj/hypertree/benchmarks.zip
[9] See: spectreattack.com

config	N	$t[s]$ median	avg	std
FraSMT (C6P)	**1449**	1189	3124	3299
FraSMT (C4P)	1434	1187	3192	3326
FraSMT (C4p)	1282	1760	3461	3432
FraSMT (c0p)	1106	7200	4019	3398
det-k-decomp	838	7200	4672	3357

Table 1. Overview on the number N of instances for which the respective decomposer configuration outputted the exact (fractional) hypertree width of the instance within the given timeout. Configuration: c/C represents disabled or enabled symmetry breaking and lower bound techniques, respectively. p/P represents disabled or enabled preprocessing techniques. 0 represents that finding cliques was disabled. 4 and 6 represent that we used the ASP solver to search for a k-hyperclique of maximum cardinality with $k \in \{4, 6\}$. However, due to the imposed timeout, the solver might also just use an ℓ-hyperclique where $3 \leq \ell \leq k$. t median (avg, std) represents the median (average, standard deviation) of the runtime in seconds of the decomposer over all instances of our benchmark instances, including the timeouts.

fhtw	1	$(1, 2]$	$(2, 3]$	$(3, 4]$	$(4, 5]$	$(5, 6]$	$(6, 7]$	$(7, 8]$	$(8, 9]$
N	145	123	198	255	308	273	65	81	1

Table 2. Distribution of the fractional hypertree width over the solved instances.

the experiments on a cluster of 16 nodes. Each node is equipped with two Intel Xeon E5-2640v4 CPUs consisting of 10 physical cores each at 2.4 GHz clock speed and 160 GB RAM. Hyper threading was disabled.

Setup and Limits. In order to draw conclusions about the efficiency of *FraSMT*, we mainly inspected the wall clock time. We set a timeout of 7200 seconds and limited available RAM to 8 GB per instance. Resource limits where enforced by *runsolver* [41]. Due to hardware resource limitations we conducted only one run per instance and configuration. However, we benchmarked a few instances with multiple runs and observed no significant difference.

7.4 Results

We used a tool to gather data and control the benchmark generation, evaluation, and cluster setting [31]. We publicly provide all experimental data [17], including raw data such as all command line flags used, system sampling (RAM/sysload), standard output and standard error during the run.

Solved Instances/Runtime. Table 1 provides basic statistics on the benchmarks. The table contains the tested configurations of our decomposer and the number of solved instances for which we obtained the (fractional) hypertree width and present total

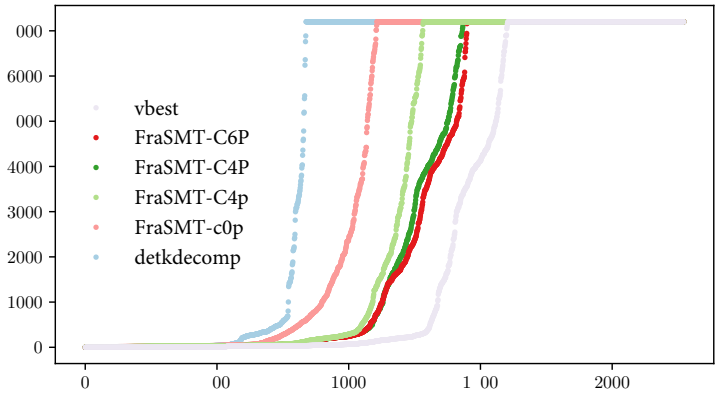


Fig. 2. Runtime in seconds on the considered benchmark instance. vbest refers to the virtual best solver. The x-axis labels consecutive integers that identify instances. The instances are ordered by running time, individually for each solver.

(average, minimum) runtime of the decomposer. We include timeouts of 7200 seconds into the average and median. Figure 2 illustrates runtime results for the tested decomposer configurations as cactus plot. We solved instances that have up to 1453 vertices, up to 891 hyperedges, and up to hyperedges of size 16. The best configuration, namely *FraSMT* (C6P), was capable of decomposing 1451 out of the total number of 2191 instances. Using k-hypercliques with $k = 4$ instead of $k = 6$ for symmetry breaking solved 1435 instances. Without preprocessing, *FraSMT* was able to solve 1283 instances. Without preprocessing or symmetry breaking, *FraSMT* could obtain 1107 fractional hypertree decompositions of exact width. Solver *det-k-decomp* was able to solve 838 instances, although both underlying methods are exact and *det-k-decomp* computes the less general parameter hypertree width in the same time. We further observe that by analyzing the virtual best solver (vbest), there are few instances a single best configuration cannot solve but can be solved by different configurations.

(Fractional) Hypertree Width. We computed the fractional hypertree width for our benchmarks using *FraSMT* and the hypertree width using *det-k-decomp*. The sets contain a few identical instances that occur in multiple sets. Even though we provide here only an overview on all instances, we decided to keep the duplicate instances to analyze the benchmark sets as provided from the original source for easier comparability. We provide detailed statistics online [17]. Using *det-k-decomp* we obtained the hypertree width for 838 instances. Table 2 provides the distribution of the number of instances and their respective fractional hypertree width. Considering all sets, 33% of the instances have fractional hypertree width below 4, 60% of the instances have fractional hypertree width below 6. Overall we were able to obtain the exact width for 66% of the instances.

Fractional Hypertree Width vs. Hypertree Width. When considering the obtained fractional hypertree width and hypertree width for these instances in our benchmark set that have been solved by both methods, the best *FraSMT* configuration and *det-k-decomp*,

we observed a difference between the two width measures on 221 instances. The maximum difference was 2, and among these 221 instances the median difference was 0.6. However, since *det-k-decomp* could decompose significantly fewer instances and by construction works better on instances of small width, we expect the difference between *fhtw* and *htw* to be significantly higher on the remaining instances.

8 Discussion and Conclusions

Our SMT-based encoding for fractional hypertree width, in combination with preprocessing and symmetry breaking methods, and its implementation, enable us to compute the exact fractional hypertree width for many realistic instances. This provides a significant step for making the theoretical notion of bounded fractional hypertree width, the most general known structural restriction for CSP that guarantees tractability, accessible for a practical use.

Our results show that a large majority of our considered benchmarked instances have low fractional hypertree width (below 10). However, we were unable to compute the exact width for about 33% of the instances. Consequently, we think that upper bound computations either using heuristics for hypertree width or modifying our encoding to obtain only upper bounds are of interest for future investigations.

Interestingly, we obtained the exact fractional hypertree width for more instances using our decomposer *FraSMT* than the exact hypertree width using *det-k-decomp*, although our decomposer determined the more general parameter. An important factor is the extensive preprocessing and symmetry breaking methods, which are not present in *det-k-decomp*, as these methods resulted in 16% more solved instances for our decomposition technique. However, even without preprocessing or symmetry breaking our approach solved more instances than *det-k-decomp*.

Since our results are limited to small and medium-sized hypergraphs up to about 1400 vertices, 900 hyperedges, and hyperedges of small size, heuristics or combinations of heuristics and exact methods might be interesting for practical purposes. Our techniques can be very helpful to evaluate the accuracy of heuristics. Efficient and precise heuristics would enable us to obtain a broad picture about available instances in CSP which might lead to a usage of fractional hypertree decompositions for solving actual CSP instances, in particular, for problems such as model counting in CSP. We believe that our methods can be extended to compute the "fractional FAQ-width" which, when bounded, renders the Functional Aggregate Query (FAQ) problem tractable [33].

The focus of this paper was the exact computation of fractional hypertree width. However, we would like to point out that with our approach one can also compute good upper bounds on the fractional hypertree width by just skipping the expensive optimality check. We have reasons to believe that this will scale to significantly larger instances, since a similar behaviour has been observed in related work [18,35]. We are interested to address this potential of our method systematically in future work.

References

1. Alviano, M., Dodaro, C.: Anytime answer set optimization via unsatisfiable core shrinking. Theory Pract. Log. Program. **16**(5-6), 533–551 (2016)

2. Arocena, P.C., Glavic, B., Ciucanu, R., Miller, R.J.: The ibench integration metadata generator. In: Li, C., Markl, V. (eds.) Proceedings of Very Large Data Bases (VLDB) Endowment. vol. 9:3, pp. 108–119. VLDB Endowment (Nov 2015), https://github.com/RJMillerLab/ibench
3. Audemard, G., Boussemart, F., Lecoutre, C., Piette, C.: XCSP3: an XML-based format designed to represent combinatorial constrained problems. http://xcsp.org (2016)
4. Bannach, M., Berndt, S., Ehlers, T.: Jdrasil: A modular library for computing tree decompositions. In: Iliopoulos, C.S., Pissis, S.P., Puglisi, S.J., Raman, R. (eds.) 16th International Symposium on Experimental Algorithms, SEA 2017, June 21-23, 2017, London, UK. LIPIcs, vol. 75, pp. 28:1–28:21. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2017)
5. Benedikt, M., Konstantinidis, G., Mecca, G., Motik, B., Papotti, P., Santoro, D., Tsamoura, E.: Benchmarking the chase. In: Geerts, F. (ed.) Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'17). pp. 37–52. Assoc. Comput. Mach., New York, Chicago, Illinois, USA (2017), https://github.com/dbunibas/chasebench
6. Berg, J., Lodha, N., Järvisalo, M., Szeider., S.: MaxSAT benchmarks based on determining generalized hypertree-width. Tech. rep., MaxSAT Evaluation 2017 (2017)
7. Berg, J., Järvisalo, M.: SAT-based approaches to treewidth computation: An evaluation. In: 26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014, Limassol, Cyprus, November 10-12, 2014. pp. 328–335. IEEE Computer Society (2014)
8. Bodlaender, H.L.: A partial k-arboretum of graphs with bounded treewidth. Theoretical Computer Science **209**(1-2), 1–45 (1998)
9. Bodlaender, H.L., Möhring, R.H.: The pathwidth and treewidth of cographs. SIAM J. Discrete Math. **6**(2), 181–188 (1993)
10. Carbonnel, C., Cooper, M.C.: Tractability in constraint satisfaction problems: a survey. Constraints **21**(2), 115–144 (2016)
11. Cohen, D., Jeavons, P., Gyssens, M.: A unified theory of structural tractability for constraint satisfaction problems. J. of Computer and System Sciences **74**(5), 721–743 (2008)
12. Committee, M.S.: IEEE standard for floating-point arithmetic. IEEE Std 754-2008 pp. 1–70 (Aug 2008)
13. Dechter, R.: Tractable structures for constraint satisfaction problems. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming, vol. I, chap. 7, pp. 209–244. Elsevier (2006)
14. Dell, H., Komusiewicz, C., Talmon, N., Weller, M.: The PACE 2017 parameterized algorithms and computational experiments challenge: The second iteration. In: Lokshtanov, D., Nishimura, N. (eds.) Proceedings of the 12th International Symposium on Parameterized and Exact Computation (IPEC'17). pp. 30:1—30:13. LIPIcs (2017)
15. Durand, A., Mengel, S.: Structural tractability of counting of solutions to conjunctive queries. Theoretical Computer Science **57**(4), 1202–1249 (2015)
16. Fichte, J.K., Hecher, M., Lodha, N., Szeider, S.: A Benchmark Collection of Hypergraphs (Jun 2018). https://doi.org/10.5281/zenodo.1289383
17. Fichte, J.K., Hecher, M., Lodha, N., Szeider, S.: Analyzed benchmarks and raw data on experiments for FraSMT (Jun 2018). https://doi.org/10.5281/zenodo.1289429
18. Fichte, J.K., Lodha, N., Szeider, S.: SAT-based local improvement for finding tree decompositions of small width. In: Gaspers, S., Walsh, T. (eds.) Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10491, pp. 401–411. Springer (2017)
19. Fischl, W., Gottlob, G., Longo, D.M., Pichler, R.: HyperBench: a benchmark of hypergraphs. http://hyperbench.dbai.tuwien.ac.at (2017)
20. Fischl, W., Gottlob, G., Pichler, R.: Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems, houston, tx, usa, june 10-15, 2018. In: den

Bussche, J.V., Arenas, M. (eds.) Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018. pp. 17–32. ACM (2018)
21. Freuder, E.C.: A sufficient condition for backtrack-bounded search. J. of the ACM **29**(1), 24–32 (1982)
22. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Multi-shot ASP solving with clingo. CoRR **abs/1705.09811** (2017), http://arxiv.org/abs/1705.09811
23. Geerts, F., Mecca, G., Papotti, P., Santoro, D.: Mapping and cleaning. In: Cruz, I., Ferrari, E., Tao, Y. (eds.) Proceedings of the IEEE 30th International Conference on Data Engineering (ICDE'14). pp. 232–243 (March 2014)
24. Gottlob, G., Leone, N., Scarcello, F.: Hypertree decompositions and tractable queries. J. of Computer and System Sciences **64**(3), 579–627 (2002)
25. Gottlob, G., Leone, N., Scarcello, F.: On tractable queries and constraints. In: Database and Expert Systems Applications, 10th International Conference, DEXA '99, Florence, Italy, August 30 - September 3, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1677, pp. 1–15 (1999)
26. Gottlob, G., Samer, M.: A backtracking-based algorithm for hypertree decomposition. J. Exp. Algorithmics **13**, 1:1.1–1:1.19 (Feb 2009)
27. Grohe, M., Marx, D.: Constraint solving via fractional edge covers. In: Proceedings of the of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2006). pp. 289–298. ACM Press (2006)
28. Grohe, M., Marx, D.: Constraint solving via fractional edge covers. ACM Transactions on Algorithms **11**(1), Art. 4, 20 (2014)
29. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. Web Semantics: Science, Services and Agents on the World Wide Web **3**(2), 158–182 (2005)
30. Hagberg, A.A., Schult, D.A., Swart, P.J.: Exploring network structure, dynamics, and function using networkx. In: Gäel Varoquaux, T.V., Millman, J. (eds.) Proceedings of the 7th Python in Science Conference (SciPy'08). pp. 11–15. Pasadena, CA, USA (Aug 2008)
31. Kaminski, R., Schneider, M., Rabener, T., et al.: benchmark-tool (2017), https://github.com/potassco/benchmark-tool
32. Khamis, M.A., Ngo, H.Q., Rudra, A.: FAQ: questions asked frequently. In: Milo, T., Tan, W. (eds.) Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016. pp. 13–28. Assoc. Comput. Mach., New York (2016)
33. Khamis, M.A., Ngo, H.Q., Rudra, A.: FAQ: questions asked frequently. CoRR **abs/1504.04044** (2017), http://arxiv.org/abs/1504.04044v6, full version of [32]
34. Leis, V., Gubichev, A., Mirchev, A., Boncz, P., Kemper, A., Neumann, T.: How good are query optimizers, really? Proceedings of Very Large Data Bases (VLDB) Endowment **9**(3), 204–215 (Nov 2015)
35. Lodha, N., Ordyniak, S., Szeider, S.: A SAT approach to branchwidth. In: Creignou, N., Berre, D.L. (eds.) Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9710, pp. 179–195. Springer Verlag (2016)
36. Lodha, N., Ordyniak, S., Szeider, S.: SAT-encodings for special treewidth and pathwidth. In: Gaspers, S., Walsh, T. (eds.) Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10491, pp. 429–445. Springer Verlag (2017)
37. Marx, D.: Approximating fractional hypertree width. TALG **6**(2), Art. 29, 17 (2010)

38. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems Tools (TACS'08). Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer Verlag (2008)
39. Rose, D.J.: On simple characterizations of k-trees. Discrete Math. **7**, 317–322 (1974)
40. van Rossum, G.: Python tutorial. Cs-r9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam (May 1995)
41. Roussel, O.: Controlling a solver execution with the runsolver tool. J on Satisfiability, Boolean Modeling and Computation **7**, 139–144 (2011)
42. Samer, M., Veith, H.: Encoding treewidth into SAT. In: Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5584, pp. 45–50. Springer Verlag (2009)
43. Transaction Processing Performance Council (TPC): TPC-H decision support benchmark. Tech. rep., TPC (2014), http://www.tpc.org/tpch/default.asp