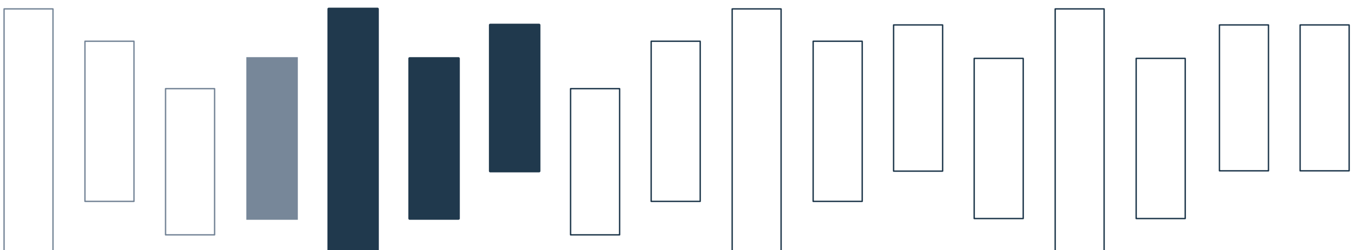
Technical Report AC-TR-20-010

September 2020

Breaking Symmetries with RootClique and LexTopSort

Johannes K. Fichte, Markus Hecher, and Stefan Szeider



This is the authors' copy of a paper that will appear in the proceedings of CP'20, the 26th International Conference on Principles and Practice of Constraint Programming. DOI 10.1007/978-3-030-58475-7_17

www.ac.tuwien.ac.at/tr

Technical Report AC-TR-20-010

Breaking Symmetries with RootClique and LexTopSort*

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

[1] School of Engineering Sciences, TU Dresden, Dresden, Germany
johannes.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
sz@ac.tuwien.ac.at

Abstract Bounded fractional hypertree width is the most general known structural property that guarantees polynomial-time solvability of the constraint satisfaction problem. Fichte et al. (CP 2018) presented a robust and scalable method for finding optimal fractional hypertree decompositions, based on an encoding to SAT Modulo Theory (SMT). In this paper, we provide an in-depth study of two powerful symmetry breaking predicates that allow us to further speed up the SMT-based decomposition: RootClique fixes the root of the decomposition tree; LexTopSort fixes the elimination ordering with respect to an underlying DAG. We perform an extensive empirical evaluation of both symmetry-breaking predicates with respect to the primal graph (which is known in advance) and the induced graph (which is generated during the search).

Keywords: Symmetry Breaking · Hypergraphs · Elimination Orderings · SAT Modulo Theory.

1 Introduction

Bounded *fractional hypertree width*, introduced by Grohe and Marx [24,25], is the most general known purely structural restriction that guarantees polynomial-time tractability of the CSP. It generalizes all previously introduced structural restriction, including treewidth [11,19], spread-cut width [10] and hypertree width [22]. However, in order to utilize bounded fractional hypertree width of a CSP instance for solving it efficiently, one needs to have a fractional hypertree decomposition of the constraint hypergraph of the CSP instance available, witnessing the bounded fractional hypertree width. Computing such a decomposition of smallest width is again an NP-hard task [18]. Nevertheless, previous work by Fichte et al. [16] showed that a practically feasible SMT (SAT Modulo Theory) encoding exists, which supports the computation of optimal fractional hypertree decompositions of constraint hypergraphs with several hundred of vertices.

* The work has been supported by the Austrian Science Fund (FWF), Grants Y698, 32441, and 32830, and the Vienna Science and Technology Fund, Grant WWTF ICT19-065. Hecher is also affiliated with the University of Potsdam, Germany.

Technical Report AC-TR-20-010

Contribution. In this paper, we introduce and study new symmetry breaking methods that speed up the SMT-approach for finding optimal fractional hypertree decompositions. Fractional hypertree decompositions are defined in terms of an (unrooted) decomposition tree, whose nodes are labeled with so-called bags of vertices of the decomposed hypergraph. However, the SMT encoding is based on a characterization of fractional hypertree width in terms of linear elimination orderings of the vertices of the decomposed constraint hypergraph. A hypergraph with n vertices has $n!$ linear elimination orderings, where many of these correspond to the same decomposition tree. Consequently, that there is much room for *symmetry breaking* (SB) strategies. We take a closer look at two symmetry breaking methods: *RootClique* and *TopSort*.

RootClique is based on the observation, that we can pick any clique of the primal graph of the decomposed hypergraph, and assume that this clique appears in the bag of the decomposition tree's root; hence we call it a root clique. This symmetry breaking predicate allows us to restrict the considered linear orderings to only those where vertices of the root clique appear at the very end.

A linear ordering of the vertices gives rise to a decomposition DAG on the same vertex set, whose arcs correspond to the edges of the induced primal graph of the decomposed hypergraph, oriented according to the linear ordering. LexTopSort is based on the observation that from the many linear orderings that are all topological sort of the same decomposition DAG, it suffices to consider only one of them.

For both symmetry breaking predicates, we consider *static* and *dynamic* variants. The static variants operate on the *primal graph* of the given hypergraph. The dynamic variant operates on the *induced primal graph*, which is obtained from the primal graph during the search by adding fill-in edges according to the computed elimination ordering. Whereas the static variants have the advantage that the symmetry breaking constraints can be computed in a preprocessing phase before the decomposition process starts, it has the disadvantage of having fewer edges available and thus breaks fewer symmetries. Our experiments show whether advantage or disadvantage dominates.

A static version of RootClique was initially suggested for tree decompositions by Bodlaender et al. [7] and then ported to fractional hypertree decompositions by Fichte et al. [16]. For tree decompositions, it is reasonable to take a largest clique as the root clique, as it has the best chance to break the most symmetries. For a hypergraph, it is not clear what makes a clique well-suited for symmetry breaking. In addition to the size of the root clique, we consider several other criteria such as the size of the root clique including its neighborhood, the size of the root clique not counting twin vertices, or the number of hyperedges being incident with a vertex of the root clique. We also introduce a dynamic variant of RootClique, which requires a nontrivial SMT encoding.

For the other symmetry breaking predicate, LexTopSort, it is the other way around: a dynamic version has been suggested for hypertree decompositions by Schidler and Szeider [35], and we introduce and test a first static variant.

We provide an extensive experimental evaluation of all the discussed variants and combinations of RootClique and LexTopSort within the basic SMT encoding for fractional hypertree width *frasmt* by Fichte et al. [16]. For the experiments, we ran all the mentioned variants on the two leading SMT solvers *z3* [32] and *optimathsat* [36]. Overall, RootClique seems to show better results than LexTopSort. However, combining the two techniques at the same time even further improved the performance and the number of solved instances. Notably, it seems that using both solvers *z3* and *optimathsat* in combination, where we preferred the latter for instances of higher fractional hypertree width, the resulting portfolio is quite close to the virtual best solver of our experiments.

2 Preliminaries

Hypergraphs. 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 is 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 $N_H(v) = (\bigcup E_H(v)) \setminus \{v\}$; 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 (with respect to H) 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 $\sum_{e \in E_H(v)} \gamma(e) \geq 1$. 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.

Hypertree Decompositions. 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,

Technical Report AC-TR-20-010

which was added to make the computation of the decomposition tractable [22]. The *generalized hypertree width* $ghtw(H)$ of H is the smallest width over all generalized hypertree decompositions of H. The *hypertree width* $htw(H)$ 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 $H = (V, E)$ with $E_H(v) \neq \emptyset$ 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 $\mathit{fhtw}(H) \leq ghtw(H) \leq htw(H) \leq tw(P(H))$.

Elimination Orderings. The first SAT encoding of treewidth was suggested by Samer and Veith [34]. It uses an ordering-based characterization of treewidth which is also used by more recent SAT encodings of treewidth [3,6]. Later, ordering-based encodings where used for hypertree width [35], generalized hypertree width [5], and fractional hypertree width [16,28].

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 $e_i = \{\, v \in \{v_{i+1}, \ldots, v_n\} \mid \text{there is some } e \in E^{i-1} \text{ containing } v \text{ and } v_i \,\}$. We consider the binary relation $Arc_L = \{\, (v_i, v_j) \in V \times V \mid i < j \text{ and } v_i \text{ and } v_j \text{ are adjacent in } H_L^n \,\}$. We write $Arc_L(i) = \{v_i\} \cup \{\, v_j \mid (v_i, v_j) \in Arc_L \,\}$, hence $Arc_L(i) = \{v_i\} \cup e_i$. We refer to $P(H_L^n)$, the primal graph of the induced hypergraph H_L^n, as the *induced primal graph*.

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 $Arc_L(i)$, i.e.,

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

Theorem 1 ([16]). *The fractional hypertree width of a hypergraph H equals the smallest fractional width over all its linear orderings: $\mathrm{fhtw}(H) = \min_L \mathrm{fhtw}_L(H)$.*

3 Symmetry Breaking for Elimination Orderings

In this section, we define all the symmetry breaking predicates, static and dynamic, and describe their encoding. Throughout this section, let $H = (V, E)$ be a fixed hypergraph.

Technical Report AC-TR-20-010

As laid out in the proof of Theorem 1, one can translate back and forth between linear orderings and fractional hypertree decomposition, preserving the fractional width. The translation from the linear ordering into a decomposition is canonical and deterministic. The translation in the other direction, however, depends on several choices. Let us briefly describe the translation.

Let $\mathcal{F} = (T, \chi, \gamma)$ be a fractional hypertree decomposition of H. First, we choose a node r of T as the root and consider a rooted version T_r of T. For each vertex $v \in V$, let $t = f(v)$ be the node of T_r with $v \in \chi(t)$ that is closest to the root r. This consideration yields a partial ordering $\leq_{\mathcal{F}}$ of V, where $u \leq_{\mathcal{F}} v$ if and only if $f(u)$ is a descendant of $f(v)$ in T_r. The proof of Theorem 1 [16] shows that any linear ordering L of V that refines $\leq_{\mathcal{F}}$, has the same fractional width as $\mathcal{F}$. We observe that the linear orderings that refine $\leq_{\mathcal{F}}$ are exactly the *topological sorts* of the DAG $D_{\mathcal{F}} = (V, A)$ where $A = \{ (u, v) \mid u \neq v$ and $u \leq_{\mathcal{F}} v \}$. Any topological sort L can be obtained from $D_{\mathcal{F}}$ by repeatedly deleting vertices without incoming arcs until all vertices have been deleted: L is then the set of vertices arranged by their succession of deletion.

When we fix the root and the topological sort, we have determined the linear ordering uniquely. We will fix the root with the RootClique symmetry breaking predicate, and we will fix the topological sort with the LexTopSort symmetry breaking predicate.

3.1 RootClique

The static RootClique predicate is based on the well-known fact that, if (T, χ) is a tree decomposition of a graph G and K a clique in G, then there exists a node $t \in V(T)$ with $V(K) \subseteq \chi(t)$ (see, e.g., [8]). Hence, when we choose any clique K in the primal graph $P(H)$, the static RootClique predicate requires that the root r is chosen among the nodes for which $V(K) \subseteq \chi(r)$ holds. This is not a full symmetry breaking, since a clique may appear in different bags. Hence, we suggest several strategies for choosing a clique that suits this purpose. In particular, we choose a clique in $P(H)$, maximizing

1. the size of the clique,
2. the size of the clique including its neighborhood $N_H(K)$ in $P(H)$,
3. the size of the clique not counting *twin vertices*, which are any two vertices u, v sharing the same neighborhood, i.e., where $N_H(u) = N_H(v)$, and
4. the number of hyperedges being incident with a vertex of the clique.

We also consider *k-hypercliques* which are cliques in $P(H)$ not intersecting with any hyperedge of H in more than k vertices. This concept was proposed by Fichte et al. [16], and was primarily intended to obtain lower bounds during the computation of fractional hypertree width. The dynamic RootClique predicate is based on the fact that each bag $\chi(t)$ is a clique in the *induced* primal graph, and conversely, every maximal clique in the induced primal graph corresponds to $\chi(t)$ for some node t. We uniquely determine the root of the decomposition tree by fixing the largest clique of the induced primal graph as the root clique.

Further details and the encodings are given after the next subsection.

Technical Report AC-TR-20-010

3.2 LexTopSort

We start by explaining the dynamic variant of LexTopSort. For better integration with RootClique, we use the inverse variant of topological sorting, where vertices without outgoing arcs are deleted, proving the ordering "from right to left." When there are several vertices without outgoing arcs, we choose the lexicographically smallest vertex next. This choice uniquely determines the linear ordering, which corresponds to the reflected lexicographically smallest topological sort. We denote for two vertices v_i, v_j by $\text{lex}(v_i, v_j)$ that v_i precedes v_j in the lexicographic ordering.

We enforce this restriction on the linear ordering $L = (v_1, \ldots, v_n)$ of V with the following predicate: *For any $1 \leq i < j \leq n$, if $lex(v_i, v_j)$, then there must be some $k \in \{i+1, \ldots, j\}$ such that the induced primal graph contains the edge* $\{v_i, v_k\}$. In other words, when we delete the lexicographically larger vertex v_j before we delete the lexicographically smaller vertex v_i, then v_i must have a neighbor v_k which has not been eliminated at that time, i.e., v_i has an outgoing arc in $D_{\mathcal{F}}$ to a vertex v_k that is still present when v_j is deleted.

Since the encoding of dynamic LexTopSort is expensive, we propose a new relaxed static version, which does not break all symmetries but can be encoded in a significantly more compact way. The static version is obtained by a small but influential change in the symmetry breaking predicate, by using the primal graph, not the induced primal graph: *For any $1 \leq i < j \leq n$, if $lex(v_i, v_j)$, then there must be some $k \in \{i+1, \ldots, j\}$ such that the primal graph contains the edge $\{v_i, v_k\}$.*

3.3 Encodings for Symmetry Breaking

In this section, we describe how RootClique and LextTopSort can be encoded within the SMT encoding for fractional hypertree width due to Fichte et al. [16], which we briefly review. To this end, let $H = (V, E)$ be a given hypergraph with $V = \{v_1, \ldots, v_n\}$ and w be a rational number. The encoding is an SMT formula that is satisfiable if and only if the hypergraph has a linear ordering L of V such that $\mathit{fhtw}_L(H) \leq w$. For computing the relation Arc_L, it uses Boolean *ordering variables* $o_{i,j}$ for $1 \leq i < j \leq n$ and Boolean *arc variables* $a_{i,j}$ for $1 \leq i, j \leq n$. Clauses are added that ensure that an ordering variable $o_{i,j}$ is true if and only if $i < j$ and v_i precedes v_j in L. In the following, we let $o^*(i,j)$ refer to $o(i,j)$ if $i < j$ and $\neg o(j,i)$ otherwise. 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 $v_j \in \mathit{Arc}_L(i)$. Finally, *weight variable* $w(i,e)$ for each $1 \leq i \leq n$ and $e \in E$ is used to represent 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.

Static RootClique (*s-RQ*). We encoded all the different variants for choosing cliques, and computed them by a solver in a separate solving step, executed before the actual decomposition. To this end, we use Boolean *clique variables* of

Technical Report AC-TR-20-010

the form $k(i)$ for each vertex $v_i \in V(H)$, where those vertices set to true form a clique K. All variants have in common, that a clique is computed as follows.

$[\neg k(i) \vee \neg k(j)]$ for any two vertices $v_i, v_j \in V(H)$ with $v_j \notin N_H(v_i)$.

Then, we considered different maximization constraints on top, resulting in different variants. In the following, we present variants for computing cliques that require adding different constraints to the constraint above.

Largest Clique (LQ). For obtaining a clique of size at least ℓ, we add the following constraints.

$[\sum_{v_i \in V(H)} k(i) \geq \ell]$

Largest Clique Including Neighbors (LQ+N). We also considered maximizing a clique, where we additionally count the neighbors of the clique.

$[\sum_{v_i \in V(H)} k(i) + |N_H(v_i)| \geq \ell]$

Largest Clique Excluding Twins (LQ-T). The following variant excludes twin vertices, when computing a maximal clique.

$[\sum_{v_i \in V(H)} k(i) - (\sum_{v_i \in V(H)} |\{v_j \in V(H) \mid j > i, N_H(v_j) = N_H(v_i)\}|) \geq \ell]$

Largest k-Hyperclique (k-Hy). For k-hypercliques, we need the following additional constraints.

$[\neg k(i_1) \vee \cdots \vee \neg k(i_k)]$ for any k vertices $v_{i_1}, \ldots, v_{i_k}$ of hyperedge $e \in E(H)$,
$[\sum_{v_i \in V(H)} k(i) \geq \ell]$

Clique with Largest Number of Used Hyperedges (LuH). This variant concerns only about maximizing the number of hyperedges that are adjacent to a clique.

$[\sum_{e \in E(H)} (\bigvee_{v_i \in e} k(i)) \geq \ell]$

Finally, after having computed a clique K, which can be obtained with any of the variants above, one can add the following constraints to the base encoding in order to actually break symmetries, statically guided by K. More precisely, the clique K is ensured to be eliminated before the other vertices (and considered the root of the decomposition) such that each vertex of K is eliminated in lexicographic order, i.e., according to L.

$[o^*(i, j)]$ for $v_j \in V(H) \setminus K$, $v_i \in K$,
$[o^*(i, j)]$ for $v_i, v_j \in K$, $v_i \neq v_j$, $\mathrm{lex}(v_i, v_j)$.

Technical Report AC-TR-20-010

Dynamic RootClique (d-RQ). While the main idea of this approach is similar to static RootClique, here we aim for a largest bag of the resulting decomposition to be the root node. However, this does not depend on the presence of a clique in the primal graph $P(H)$. Instead, we require such a clique in the induced primal graph of H. As a result, the SMT encoding for fractional hypertree decompositions based on elimination orderings is directly extended. For finding a largest bag, Boolean variables $B(i), b(i)$ for $1 \leq i \leq n$ and integer variables $d(i)$ for computing the degree of outgoing arcs in the induced primal graph of H are used. Intuitively, $B(i)$ indicates that $v_i \in V(H)$ is the lexicographically largest vertex in L that is contained in a largest bag. Consequently, smaller vertices of i are in this largest bag, whose member elements are indicated by variables $b(j)$.

The following constraints model this construction, where the degree variables are computed and only one largest bag is allowed.

$$[d(i) = \textstyle\sum_{1 \leq j \leq n, j \neq i} a(i,j)] \quad \text{for } 1 \leq i \leq n,$$
$$[\neg B(i) \vee \neg B(j)] \quad \text{for } 1 \leq i < j \leq n,$$
$$[\textstyle\bigvee_{v_i \in V(H)} B(i)].$$

We ensure that if for vertex v_i there is a lex-smaller vertex v_j, where there is no arc from v_i to v_j, v_i cannot be the largest vertex in a largest bag. Further, for vertex v_i with $B(i)$ it is not allowed that there is a larger bag (of larger degree) with a lexicographically larger vertex v_j.

$$[\neg o^*(j,i) \vee a(i,j) \vee \neg B(i)] \quad \text{for } 1 \leq i, j \leq n,\ i \neq j,$$
$$[\neg o^*(i,j) \vee B(j) \vee \neg B(i) \vee d(j) \leq d(i)] \quad \text{for } 1 \leq i, j \leq n,\ i \neq j.$$

For fixing the order of the elements within the bag and in relation to elements outside this bag, we compute the elements of this largest bag as follows.

$$[\neg b(i) \vee \neg o^*(j,i) \vee \neg B(j)] \quad \text{for } 1 \leq i, j \leq n$$
$$[\neg B(i) \vee \neg o^*(j,i) \vee b(j)] \quad \text{for } 1 \leq i, j \leq n.$$

Then, we fine-tune the symmetry breaking by setting the order within this (largest) bag and in relation to the other vertices. This is similar to symmetry breaking for RootClique, but depending on the elements of the largest bag.

$$[\neg b(i) \vee b(j) \vee o^*(i,j)] \quad \text{for } 1 \leq i < j \leq n,$$
$$[\neg b(i) \vee \neg b(j) \vee o^*(i,j)] \quad \text{for } 1 \leq i < j \leq n,\ \text{lex}(v_i, v_j).$$

LexTopSort. For encoding LexTopSort, we need the following additional SMT variables. Boolean variables $s(i,j)$ for each $1 \leq i,j \leq n$ with $i \neq j$ are used to represent that v_i has v_j as the lex-smallest vertex with an arc from v_i to v_j in the induced primal graph. For connected hypergraphs H, obviously, such a vertex v_j has to exist for every vertex v_i, except for the smallest vertex in the

Technical Report AC-TR-20-010

ordering L. Being the smallest vertex v_i is represented with Boolean variables $l(i)$ (for every $1 \leq i \leq n$).

For both static and dynamic LexTopSort, we need to encode that there is only one such smallest vertex. Further, we need to make sure that if for three vertices v_i, v_j, v_k with $\text{lex}(v_i, v_j)$ either we have that $o^*(i,j)$ (i is eliminated before j), or v_k is not the lex-smallest vertex of v_i, or otherwise it is guaranteed that j is eliminated before k. Intuitively, this ensures that either the succession of elimination coincides with L or deleting the lexicographically larger vertex v_j is allowed since v_k is present in $D_{\mathcal{F}}$ when v_j is deleted.

$$[\neg l(i) \vee \neg l(j)] \qquad \text{for } 1 \leq i < j \leq n,$$
$$[o^*(i,j) \vee \neg s(i,k) \vee o^*(j,k)] \qquad \text{for } 1 \leq i,j,k \leq n,\ \text{lex}(v_i, v_j).$$

Then, we add one of the following two blocks of constraints, depending on the static or dynamic variant of LexTopSort.

Static LexTopSort (s-LT). The static variant ensures that for every vertex v_i that either v_i is the smallest vertex or v_i has a lex-smallest vertex. Then, for two neighbors v_j, v_k of v_i, if $s(i,k)$, v_j cannot be eliminated before v_k in $D_{\mathcal{F}}$.

$$[\textstyle\bigvee_{\{v_i,v_j\} \in E(P(H))} s(i,j) \vee l(i)] \qquad \text{for } v_i \in V(H),$$
$$[\neg o^*(j,k) \vee \neg s(i,k)] \qquad \text{for } 1 \leq i,j,k \leq n,\ j \neq k, \{v_j, v_k\} \subseteq N_H(v_i).$$

Dynamic LexTopSort (d-LT). Conceptually, dynamic LexTopSort is similar to static LexTopSort, although the variants show major differences in runtime as we will see in Section 4.2. First, for every vertex v_i either v_i is the smallest vertex or there is a lex-smallest vertex for v_i. Then, if v_i has v_j as the lex-smallest vertex, we require an arc from v_i to v_j in the induced primal graph of H. Similar to static LexTopSort, if there are two candidates v_j and v_k for being the lex-smallest vertex of v_i, it is prohibited to take v_k if v_j is eliminated before v_k.

$$[\textstyle\bigvee_{i \neq j} s(i,j) \vee l(i)] \qquad \text{for } v_i \in V(H),$$
$$[\neg s(i,j) \vee a(i,j)] \qquad \text{for } 1 \leq i,j \leq n,\ i \neq j,$$
$$[\neg a(i,j) \vee \neg a(i,k) \vee \neg o(j,k) \vee \neg s(i,k)] \qquad \text{for } 1 \leq i,j,k \leq n,\ i \neq j,\ i \neq k,\ j \neq k.$$

Combining RootClique with LexTopSort. For combining RootClique with LexTopSort, we have to take care that the ordering L is in line with the vertices of the root clique being lexicographically first.

Further, for static RootClique, where we have a (static) clique K prior the actual solving with the SMT encoding, we can easily fix the smallest vertex of the LexTopSort encoding as follows.

$$[l(i)] \qquad \text{for } v_i \in K, \text{ if for every } v_j \in K \text{ with } j \neq i, \text{ we have } \text{lex}(v_i, v_j).$$

Technical Report AC-TR-20-010

4 Implementation and Experiments

We ported *frasmt* to *Python 3.8.* and implemented the different strategies for symmetry breaking. The source code of our solver *frasmt* is readily available at github.com/daajoe/frasmt and detailed results as well as analysis are online at Zenodo [12]. In our implementation, we support two SMT solvers, namely *z3 4.8.7* [32] as well as *optimathsat 1.6.4* [36]. Hence, we have two configurations: *frasmt_z3*, which uses the SMT solver *z3* and *frasmt_om*, which uses the SMT solver *optimathsat*. As it turns out, while one solver is overall better than the other, both solvers complement each other quite well. To demonstrate this finding, our results also show a portfolio variant *frasmt_z3+om* that uses both solvers, where for instances of fractional hypertree width larger than 4 solver *optimathsat* is invoked and below, solver *z3* is invoked. Further, for computing the cliques that are used in the static variant of RootClique, we applied a solver called *clingo 5.4.0* [20], which is an extension of SAT solvers and allows for incremental solving as well as optimization without manual cardinality constraints. For obtaining these cliques, we relied on the any-time algorithm of *clingo*. We allowed this solver to use up to 500 seconds, which showed almost the same results as using no internal time limit and –in the worst case– spending the total runtime on symmetry breaking only. Indeed, for symmetry breaking, we then used the best clique according to the optimization criteria of the clique variant that could be computed within these 500 seconds. However, we observed that it is indeed crucial to allow some time for symmetry breaking since the vanilla configuration of *frasmt_z3* is almost identical to *frasmt*, which spends only 10 seconds on limited approaches of symmetry breaking.

4.1 Benchmark Setup

We compared the different strategies of symmetry breaking with respect to the goal of finding the best variant. To this end, we configured the following setup.

Measure and Resources. In order to draw conclusions about the efficiency of the compared solvers, we mainly inspected wall clock times. We set a timeout of 7200 seconds and limited available RAM to 16 GB per instance. Resource limits were set and enforced by the tool *runsolver* [33].

Benchmark Instances. We considered a selection of 2191 instances collected by Fischl et al. [17] (publicly available at [15]) from various sources, consisting of hypergraphs that originate from CSP instances and conjunctive database queries. The instances and their original sources are summarized in Table 1. The instances contain up to 2993 vertices and 2958 hyperedges.

Benchmark Hardware. Solvers were executed on a cluster of 12 nodes. Each node is equipped with two Intel Xeon E5-2650 CPUs consisting of 12 physical cores each at 2.2 GHz clock speed, 256 GB RAM and 1 TB hard disc drives (*not* an SSD) Seagate ST1000NM0033. The results were gathered on Ubuntu 16.04.1 LTS machines with disabled hyperthreading on kernel 4.4.0-139.

Benchmark Set	Type	#Instances	Origin
`DaimlerChrysler`	Industrial	15	
`Grid2D`	Grid	12	
`ISCAS'89`	Competition	24	[23]
`MaxSAT`	MaxSAT	35	[5]
`csp_application`	XCSP	1090	[2]
`csp_random`	XCSP	863	[2]
`csp_other`	Misc	82	
`CQ`	Conjunctive Queries	156	[1,4,21,26,30,37]
$\sum$	Hyperbench	2191	[17]

Table 1. The benchmark sets consisting of the type, the number of instances as well as the origin, of the instances we considered in our experiments. Note that some benchmark sets are overlapping and therefore the numbers do not add up to 2191.

Technical Report AC-TR-20-010

Compared Solvers. We mainly compare variants of *frasmt_z3* and *frasmt_om* to see the influence of symmetry breaking. The vanilla configuration *frasmt_z3* has the same features as the best reported configuration [16] of *frasmt*, where no extensive symmetry breaking is used. The results of *frasmt_z3* and *frasmt* are almost identical (small differences may occur due to *Python* version upgrade), which is why we refrained from further adding additional data to our plots and tables. We also considered the recent solver *triangulator* [29]. While *triangulator* overall is extremely fast on about half of the instances (about 1190), we observed that the solver quickly runs out of main memory on most of the other instances, which still persists if increasing main memory to 64GB. In consequence, we only report results for the more recent solver *triangulator-msc*, which uses *cplex* and is available at github.com/Laakeri/triangulator-msc. However, we follow other recent work on symmetry breaking [9] and stress that the main goal of our experiments is to demonstrate the *benefit* of our symmetry breaks, not to compare the speed of our approach to other algorithms with different techniques.

4.2 Benchmark Results

We discuss the following three aspects, where we first elaborate on the variants of RootClique. Then, we cover the performance of LexTopSort, including the combination with RootClique, followed by static vs. dynamic symmetry breaking.

Computing Static RootCliques. Figure 1 depicts a cactus plot of the variants *s-RQ* for static RootClique, as presented in the previous section. In this figure, the x-axis refers to the number of instances, where for each solver the runtime (y-axis) is sorted in ascending order. Therefore, this plot provides an overview of the variants over all instances. In this plot, we mainly focused on showing the variants for solver *z3*, which showed overall the best performance, since the results for *om*, while different compared to *z3*, draw a similar picture. Surprisingly, the

Technical Report AC-TR-20-010

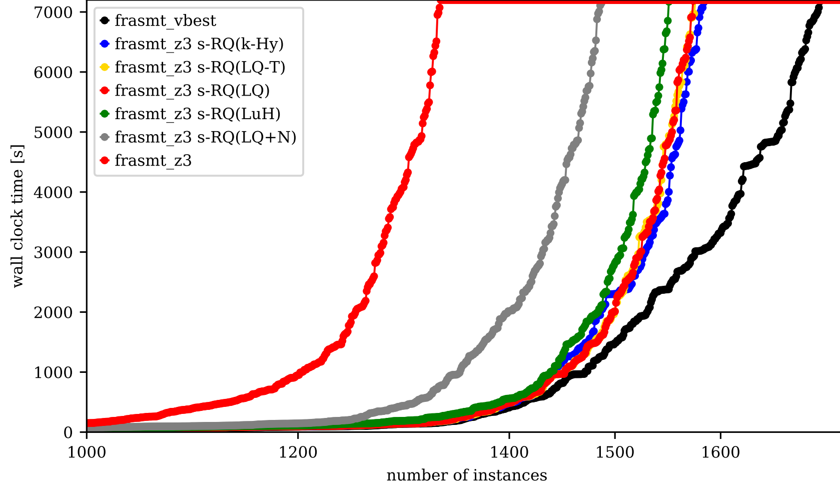


Figure 1. A cactus plot of the static variants of RootClique, compared to the vanilla configuration *frasmt_z3*. The x-axis shows the number of instances and the y-axis depicts wall clock runtimes in seconds, which are sorted in ascending order, but for each solver configuration individually. Solver "*frasmt* _vbest" refers to the virtually best solver by taking for each instance the best result among all displayed solver (configurations). The legend is ordered from best to worst (from right to left in the plot).

k-hyperclique variant *(k-Hy)* shows the best results[4], which is, however, almost as good as the two variants *LQ-T* and *LQ* for computing largest cliques. While the variant on aiming for the largest clique without twin vertices *(LQ-T)* seems to have a slight advantage over going for the largest clique *(LQ)*, the differences are minor. The fact that k-hyperclique performs best was, however, surprising. On the other hand, if one also considers the variant *LuH* for preferring cliques with the largest number of used hyperedges, it seems that k-hypercliques might form a good compromise. Notably, if also considering LexTopSort, the situation changes slightly. It turns out that the variant *LQ-T* performs better than using the k-hyperclique, followed by *LQ*.

Combining RootClique with LexTopSort. Before we discuss the combination of RootClique with LexTopSort, we briefly elaborate on the performance of the variants of LexTopSort without RootClique. It seems that especially dynamic LexTopSort worsens the picture. In more detail, both static and dynamic LexTopSort without RootClique show a rather bad performance, which is sometimes even worse than the vanilla configuration *frasmt_z3*. This observation is underlined by Figure 2, which shows a cactus plot of variants of LexTopSort and combinations with variants of RootClique. The best variants use full static symmetry breaking only (*s-LT*, *s-RQ*). While the result for *frasmt_z3* suggests not much difference between full static symmetry breaking and static RootClique only (*s-RQ*), the

[4] For comparability with *frasmt*, we used $k = 6$ (the option reported best [16]).

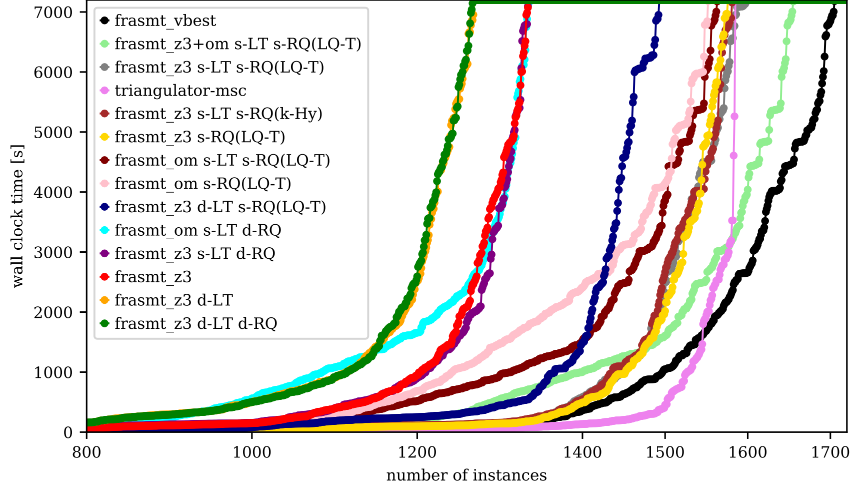


Figure 2. A cactus plot showing (combinations of) both RootClique and LexTopSort symmetry breaking, where the x-axis refers to the number of instances, and the y-axis depicts wall clock runtimes in seconds. Runtimes are sorted in ascending order for each solver configuration individually. Solver "*frasmt* _vbest" refers to the virtually best solver by taking for each instance the best result among all displayed *frasmt* configurations. The legend is ordered from best to worst (from right to left in the plot).

results for *frasmt_om* reveal that indeed with static LexTopSort, one can further improve the results obtained by static RootClique only. This might also be emphasized due to the fact that the combination of solvers *z3* and *optimathsat* provides significant improvements compared to both single configurations. Notably, *triangulator-msc* is very fast, but overall *frasmt_z3* solves more instances. Table 2 reports on the number of solved instances and total runtimes for the larger benchmark sets, where timeouts count as 7200 seconds, and results are detailed and grouped by fractional hypertree width.

Static vs. Dynamic Symmetry Breaking. The results of the previous paragraphs seem to be rather bad for the dynamic variants of symmetry breaking. However, this is not too surprising if one considers that the encoding size of dynamic LexTopSort is in $\mathcal{O}(n^3)$ and that for RootClique the encoding is in $\mathcal{O}(n^2)$, where n is the number of vertices contained in the hypergraph. Still, against all odds the cactus plot of Figure 2 already depicts dynamic variants, whose curve is sometimes below other static variants and even below our best variant *frasmt_z3+om*. Further, Figure 3 shows two scatter plots comparing runtimes instance-by-instance of the best variant of the previous paragraph (*frasmt_z3+om*, x-axis) with (y-axis) both the best variant (left) of dynamic symmetry breaking and the worst dynamic variant (right). Both dynamic variants show that, while *frasmt_z3+om* performs better on plenty of instances, there are still instances on the bottom left of the plots, where the dynamic variants are faster. Further, some instances on the

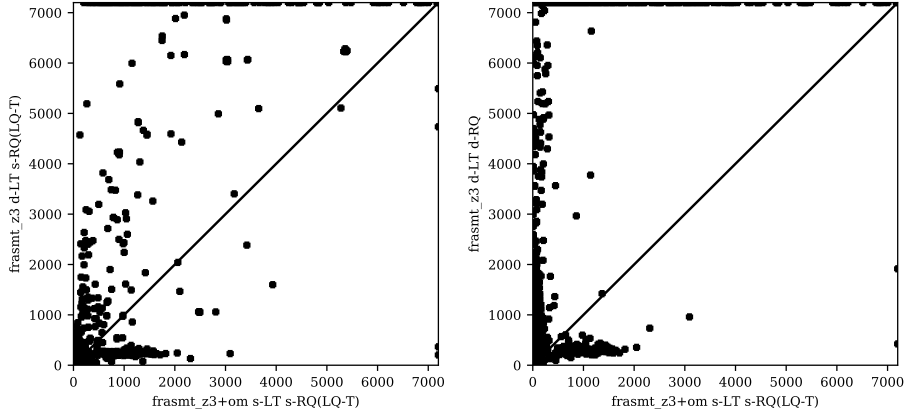


Figure 3. Scatter plots comparing runtimes of instances (in seconds) one-by-one of our best configuration (x-axis) with the best dynamic symmetry breaking method (y-axis, left) and with the worst dynamic configuration (y-axis, right).

bottom right of both plots cannot be solved by *frasmt_z3+om*, which are solved by dynamic variants.

5 Conclusion and Future Work

In this work, we analyzed different strategies for symmetry breaking in characterizations of fractional hypertree width. While we focused on this particular width parameter, the overall idea of our methods immediately apply to the computation of other width parameters, such as treewidth. Hence, we expect that our findings can be used to gain significant improvements for other ordering-based encodings [34]. Our two methods RootClique and LexTopSort for eliminating symmetries seem to be good companions, since a combination of both state-of-the-art SMT solvers *z3* [32] and *optimathsat* [36] reached the virtually best configuration. Still, we only considered the best variant of RootClique in this configuration, and it seems there is potential for algorithm selection involving machine-learning tools like autofolio [31]. Overall, we perceived static symmetry breaking strategies as superior to dynamic techniques. While this might not be too surprising, we observed that some instances could still be solved faster with dynamic techniques. We see future work in analyzing the impact of fractional hypertree width for practical solving (counting) similar to treewidth [13,14,27] and in the context of other measurements such as bag size and domain size.

Technical Report AC-TR-20-010

Set	Solver	Σ	*fhtw* range group				time[h]
			max(*fhtw*)	0-2	3-4	>4	
csp_other	frasmt_vbest	46	**7.0**	**28**	**5**	13	103.57
	frasmt_z3 s-RQ(LQ-T)	**44**	6.0	**28**	**5**	**11**	**104.62**
	frasmt_z3 s-LT s-RQ(LQ-T)	43	6.0	27	**5**	**11**	106.09
	frasmt_z3 s-LT s-RQ(k-Hy)	43	6.0	27	**5**	**11**	106.69
	frasmt_z3	43	6.0	**28**	**5**	10	110.17
	frasmt_z3 s-LT d-RQ	41	6.0	27	**5**	9	111.02
	frasmt_z3+om s-LT s-RQ(LQ-T)	40	**7.0**	27	**5**	8	112.63
	frasmt_z3 d-LT s-RQ(LQ-T)	37	6.0	26	4	7	118.04
	frasmt_om s-LT s-RQ(LQ-T)	37	**7.0**	26	3	8	118.76
	frasmt_om s-RQ(LQ-T)	36	6.0	26	3	7	119.69
	frasmt_z3 d-LT	36	6.0	26	4	6	124.27
	frasmt_om s-LT d-RQ	34	6.0	26	3	5	123.96
	frasmt_z3 d-LT d-RQ	34	6.0	25	4	5	125.89
	triangulator-msc	25	$5.\bar{3}$	17	4	4	139.92
csp_application	frasmt_vbest	674	**7.0**	**43**	397	234	775.44
	frasmt_z3 s-LT s-RQ(LQ-T)	**648**	**7.0**	**43**	**396**	209	835.76
	frasmt_z3 s-RQ(LQ-T)	646	**7.0**	**43**	**396**	207	**824.72**
	frasmt_z3 s-LT s-RQ(k-Hy)	641	**7.0**	**43**	**396**	202	842.05
	frasmt_z3+om s-LT s-RQ(LQ-T)	640	**7.0**	**43**	**396**	201	827.50
	frasmt_z3	559	**7.0**	**43**	370	146	974.08
	frasmt_z3 s-LT d-RQ	554	**7.0**	**43**	365	146	980.45
	frasmt_om s-LT s-RQ(LQ-T)	552	**7.0**	41	310	201	1003.93
	triangulator-msc	551	**7.0**	**43**	288	**220**	973.13
	frasmt_z3 d-LT s-RQ(LQ-T)	549	**7.0**	40	326	183	997.30
	frasmt_om s-RQ(LQ-T)	535	**7.0**	40	293	202	1040.55
	frasmt_z3 d-LT	498	**7.0**	40	312	146	1113.74
	frasmt_z3 d-LT d-RQ	498	**7.0**	40	312	146	1114.50
	frasmt_om s-LT d-RQ	482	**7.0**	38	293	151	1146.85
csp_random	triangulator-msc	**860**	$6.\bar{6}$	**54**	**39**	**767**	**203.21**
	frasmt_vbest	835	**9.0**	**54**	**39**	742	365.57
	frasmt_om s-RQ(LQ-T)	830	**9.0**	**54**	38	738	444.29
	frasmt_z3+om s-LT s-RQ(LQ-T)	826	**9.0**	**54**	**39**	733	419.60
	frasmt_om s-LT s-RQ(LQ-T)	824	**9.0**	**54**	37	733	424.79
	frasmt_z3 s-LT s-RQ(LQ-T)	757	**9.0**	**54**	**39**	664	478.49
	frasmt_z3 d-LT s-RQ(LQ-T)	755	**9.0**	**54**	**39**	662	488.93
	frasmt_z3 s-LT s-RQ(k-Hy)	747	**9.0**	**54**	**39**	654	478.59
	frasmt_z3 s-RQ(LQ-T)	734	**9.0**	**54**	**39**	641	490.29
	frasmt_om s-LT d-RQ	670	**9.0**	**54**	31	585	691.12
	frasmt_z3 s-LT d-RQ	590	**9.0**	**54**	23	513	808.34
	frasmt_z3 d-LT d-RQ	585	**9.0**	**54**	19	512	830.63
	frasmt_z3 d-LT	584	**9.0**	**54**	19	511	830.89
	frasmt_z3	582	**9.0**	**54**	18	510	824.62
Σ	frasmt_vbest	1706	**9.0**	**271**	446	989	1247.01
	frasmt_z3+om s-LT s-RQ(LQ-T)	**1657**	**9.0**	270	**445**	942	1362.19
	frasmt_z3 s-LT s-RQ(LQ-T)	1600	**9.0**	270	**445**	885	1423.34
	triangulator-msc	1585	7.0	259	336	**990**	**1322.97**
	frasmt_z3 s-LT s-RQ(k-Hy)	1583	**9.0**	270	**445**	868	1430.35
	frasmt_z3 s-RQ(LQ-T)	1576	**9.0**	**271**	**445**	860	1422.65
	frasmt_om s-LT s-RQ(LQ-T)	1564	**9.0**	267	355	942	1549.95
	frasmt_om s-RQ(LQ-T)	1553	**9.0**	266	339	948	1607.00
	frasmt_z3 d-LT s-RQ(LQ-T)	1493	**9.0**	266	374	853	1607.46
	frasmt_om s-LT d-RQ	1337	**9.0**	264	332	741	1966.01
	frasmt_z3 s-LT d-RQ	1336	**9.0**	270	398	668	1903.86
	frasmt_z3	1335	**9.0**	**271**	398	666	1912.94
	frasmt_z3 d-LT	1269	**9.0**	266	340	663	2073.04
	frasmt_z3 d-LT d-RQ	1268	**9.0**	265	340	663	2075.15

Table 2. Detailed results on the number of solved instances grouped by fractional hypertree width of the solved instance. Runtimes are cumulated wall clock times in hours, where timeouts count as 7200 seconds.

References

1. 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
2. Audemard, G., Boussemart, F., Lecoutre, C., Piette, C.: XCSP3: an XML-based format designed to represent combinatorial constrained problems. http://xcsp.org (2016)
3. 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)
4. 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
5. Berg, J., Lodha, N., Järvisalo, M., Szeider., S.: MaxSAT benchmarks based on determining generalized hypertree-width. Tech. rep., MaxSAT Evaluation 2017 (2017)
6. 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)
7. Bodlaender, H.L., Fomin, F.V., Koster, A.M.C.A., Kratsch, D., Thilikos, D.M.: On exact algorithms for treewidth. In: Proceedings of the 14th Annual European Symposium on Algorithms (ESA'06). Lecture Notes in Computer Science, vol. 4168, pp. 672–683. Springer Verlag (2006)
8. Bodlaender, H.L., Möhring, R.H.: The pathwidth and treewidth of cographs. SIAM J. Discrete Math. **6**(2), 181–188 (1993)
9. Codish, M., Miller, A., Prosser, P., Stuckey, P.J.: Constraints for symmetry breaking in graph representation. Constraints An Int. J. **24**(1), 1–24 (2019)
10. 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)
11. Dechter, R.: Bucket elimination: a unifying framework for reasoning. Artificial Intelligence **113**(1-2), 41–85 (1999)
12. Fichte, J.K., Hecher, M., Szeider, S.: Analyzed Benchmarks on Experiments for frasmt v2.0.0 (Dataset). Zenodo (Jul 2020), https://doi.org/10.5281/zenodo.3950097
13. Fichte, J.K., Hecher, M., Thier, P., Woltran, S.: Exploiting database management systems and treewidth for counting. In: Komendantskaya, E., Liu, Y.A. (eds.) Proceedings of the 22nd International Symposium on Practical Aspects of Declarative Languages (PADL'20). Lecture Notes in Computer Science, vol. 12007, pp. 151–167. Springer Verlag, New Orleans, LA, USA (2020)
14. Fichte, J.K., Hecher, M., Zisser, M.: An improved GPU-based SAT model counter. In: Schiex, T., de Givry, S. (eds.) Proceedings of the 25th International Conference on Principles and Practice of Constraint Programming (CP'19). pp. 491–509. Springer Verlag, Stamford, CT, USA (Sep 2019)

Technical Report AC-TR-20-010

Technical Report AC-TR-20-010

15. Fichte, J.K., Hecher, M., Lodha, N., Szeider, S.: A Benchmark Collection of Hypergraphs. Zenodo (Jun 2018), https://doi.org/10.5281/zenodo.1289383
16. Fichte, J.K., Hecher, M., Lodha, N., Szeider, S.: An SMT approach to fractional hypertree width. In: CP. Lecture Notes in Computer Science, vol. 11008, pp. 109–127. Springer (2018)
17. Fischl, W., Gottlob, G., Longo, D.M., Pichler, R.: HyperBench: a benchmark of hypergraphs. http://hyperbench.dbai.tuwien.ac.at (2017)
18. Fischl, W., Gottlob, G., Pichler, R.: General and fractional hypertree decompositions: Hard and easy cases. In: den Bussche, J.V., Arenas, M. (eds.) Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'18). pp. 17–32. Assoc. Comput. Mach., New York, Houston, TX, USA (Jun 2018)
19. Freuder, E.C.: A sufficient condition for backtrack-bounded search. J. of the ACM **29**(1), 24–32 (1982)
20. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Multi-shot ASP solving with clingo. Theory Pract. Log. Program. **19**(1), 27–82 (2019)
21. 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)
22. Gottlob, G., Leone, N., Scarcello, F.: Hypertree decompositions and tractable queries. J. of Computer and System Sciences **64**(3), 579–627 (2002)
23. Gottlob, G., Samer, M.: A backtracking-based algorithm for hypertree decomposition. J. Exp. Algorithmics **13**, 1:1.1–1:1.19 (Feb 2009)
24. 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)
25. Grohe, M., Marx, D.: Constraint solving via fractional edge covers. ACM Transactions on Algorithms **11**(1), Art. 4, 20 (2014)
26. 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)
27. Hecher, M., Thier, P., Woltran, S.: Taming high treewidth with abstraction, nested dynamic programming, and database technology. In: Pulina, L., Seidl, M. (eds.) Proceedings of the 23rd International Conference on Theory and Applications of Satisfiability Testing (SAT'20). pp. 343–360. Springer Verlag (2020)
28. 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)
29. Korhonen, T., Berg, J., Järvisalo, M.: Solving graph problems via potential maximal cliques: An experimental evaluation of the bouchitté-todinca algorithm. ACM Journal of Experimental Algorithmics **24**(1), 1.9:1–1.9:19 (2019)
30. 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)
31. Lindauer, M., Hoos, H.H., Hutter, F., Schaub, T.: Autofolio: An automatically configured algorithm selector. J. Artif. Intell. Res. **53**, 745–778 (2015)
32. 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)

33. Roussel, O.: Controlling a solver execution with the runsolver tool. J on Satisfiability, Boolean Modeling and Computation **7**, 139–144 (2011)
34. 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)
35. Schidler, A., Szeider, S.: Computing optimal hypertree decompositions. In: Blelloch, G., Finocchi, I. (eds.) Proceedings of ALENEX 2020, the 22nd Workshop on Algorithm Engineering and Experiments. pp. 1–11. SIAM (2020)
36. Sebastiani, R., Trentin, P.: Optimathsat: A tool for optimization modulo theories. J. Autom. Reasoning **64**(3), 423–460 (2020)
37. Transaction Processing Performance Council (TPC): TPC-H decision support benchmark. Tech. rep., TPC (2014), http://www.tpc.org/tpch/default.asp

Technical Report AC-TR-20-010