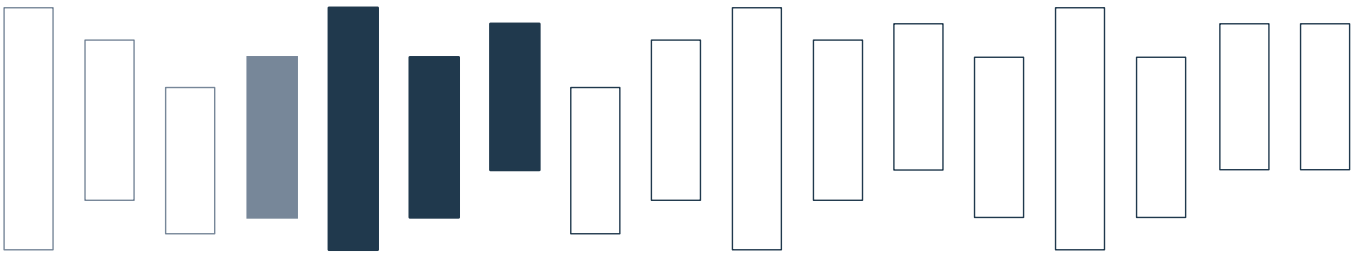
Technical Report AC-TR-20-001

January 2020

Computing Optimal Hypertree Decompositions

André Schidler and Stefan Szeider



This is the authors' copy of a paper that will appear in the proceedings of ALENEX'20, the SIAM Symposium on Algoriithm Engineering and Experiments, January 5–6, 2020, Salt Lake City, Utah, USA

www.ac.tuwien.ac.at/tr

Computing Optimal Hypertree Decompositions*

André Schidler† Stefan Szeider†

Abstract

We propose a new algorithmic method for computing the hypertree width of hypergraphs, and we evaluate its performance empirically. At the core of our approach lies a novel ordering based characterization of hypertree width which lends to an efficient encoding to SAT modulo Theory (SMT). We tested our algorithm on an extensive benchmark set consisting of real-world instances from various sources. Our approach outperforms state-of-the-art algorithms for hypertree width. We achieve a further speedup by a new technique that first solves a relaxation of the problem and subsequently uses the solution to guide the algorithm for solving the problem itself.

1 Introduction

1.1 Background The notion of a *hypertree decomposition* and the corresponding hypergraph invariant *hypertree width* were introduced by Gottlob et al. [14], who showed that many key problems arising in databases and constraint satisfaction can be solved in polynomial time when certain hypergraphs associated with the problem instances have bounded hypertree width. Since its introduction, hypertree width has become very popular, and it found many further applications, including Projected Solution Counting, Solution Enumeration (with polynomial delay), Constraint Optimization, and Combinatorial Auctions (see, e.g., the survey article by Gottlob, Greco, and Scarcello [12]). The fundamental nature of hypertree width is further underlined by the existence of combinatorial, game-theoretic, as well as logical characterizations [12].

Bounded hypertree width is witnessed by a *hypertree decomposition* of bounded width, a concept similar to a tree decomposition, but where bags are not required to be small but rather to admit small *(hyper)edge covers*. This way, large bags can still lead to hypertree decompositions of small width and, in turn, admit efficient solutions to the problem at hand. However, all the solution algorithms that utilize small hypertree width are exponential in the width. Hence the problem of computing hypertree decompositions of small width is of essential importance. Given a hypergraph H and an integer W, determining whether H has hypertree width $\leq W$ is NP-complete. The parameterized problem, where W is considered as the parameter, is $W[2]$-hard, i.e., not fixed-parameter tractable under complexity-theoretic assumptions [13]. Only when W is a constant, one achieves polynomial-time tractability, with the order of the polynomial depending on W (i.e., XP-tractability in terms of parameterized complexity). A first algorithm for computing hypertree width was proposed by Gottlob et al. in their original paper on hypertree width [14], a second, improved algorithm was later proposed and tested experimentally by Gottlob and Samer [15]. Both algorithms involve a step where all subsets of size W of the set of hyperedges are examined, in order to find small sets of hyperedges that form a separator of the hypergraph, which gives a lower bound of $\Omega(m^W)$ for the running time, where m denotes the number of hyperedges of the hypergraph. We will refer to both algorithms as *separator based algorithms*. This exponential lower bound on the running time for the separator based algorithms in terms of the hypertree width arises also in experimental settings, where these algorithms perform reasonably well on instances of small hypertree width, but struggle with instances of medium-sized hypertree width [15]. We could reproduce this expected behavior in our experiments where we used *TU Longo*, a recent implementation of the Gottlob-Samer algorithm.

1.2 Contribution We propose a new practical approach for computing the exact hypertree width of hypergraphs. We follow a logical approach which was initiated by Samer and Veith [19] for tree decompositions and was later successfully used for other (hyper)graph width measures, including clique-width [16], treecut width and treedepth [11], and fractional hypertree width [7]. For hypertree width, we had to introduce several new concepts and ideas to make this approach work. The general idea is to use a polynomial-time *encoding algorithm*, which takes as input a hypergraph H and an integer W, and produces a propositional formula $F(H, W)$, such that $F(H, W)$ is satisfiable if and only if the hypertree width of H is at most W. By trying systematically different values of W, we can determine the smallest W for which $F(H, W)$ is satisfiable, i.e., the hypertree width of H. Subsequently, we use a polynomial-time *decoding algorithm* which translates a satisfying assignment of $F(H, W)$ into hypertree decomposition of H of width W.

Our encoding utilizes an *ordering based characterization* of hypertree width, where we arrange the vertices in a linear ordering subject to certain constraints. Already the successful SAT-encodings for treewidth and fractional hyper-

*Supported by the Austrian Science Fund (FWF), projects P32441 and W1255, and by the German Research Foundation (DFG), project HO 1294/11-1.

†Algorithms and Complexity Group, TU Wien, Vienna, Austria.

tree width [19, 7] used ordering based characterizations of the corresponding width measures (for treewidth, this characterization uses the well-known characterization of graphs of bounded tree-width in terms of partial k-trees [4]). However, for hypertree width, an ordering based characterization is not straightforward. What makes it challenging to express hypertree width in terms of a linear ordering is the *Special Condition* in the definition of hypertree decompositions (see Section 2), which is formulated in terms of the *descendancy relation* in the decomposition tree. However, we succeeded in formulating the characterization in such a way that we could base a compact and efficient SAT encoding on it.

For bounding the width of a hypertree decomposition, we need to compute small hyperedge covers of vertex sets. For this purpose, we use in our encoding certain *arithmetic constraints* instead of propositional cardinality constraints [20] (as Samer and Veith did). During the solving process, these arithmetic constraints are mapped to propositional logic in an incremental fashion. This incremental encoding is handled by an *SMT (SAT Modulo Theory)* solver [2, 18], where a First-Order Logic solver (handling the arithmetic constraints) interacts with the SAT solver.

For our approach, the Special Condition is not making the problem easier. On the contrary, it makes the problem harder, as it blows up the encoding size significantly with a cubic number of additional clauses so that the solving time increases by an order of magnitude. Based on the observation that in practice the Special Condition often does not increase the width, we designed an approach that tries to avoid the usage of the full encoding or at least supports the full encoding with information gained from a relaxed encoding without the Special Condition.

To that aim, we proceed in two phases, where we first use a relaxed encoding to compute a hypertree decomposition that does not necessarily satisfy the Special Condition. We then try to modify the decomposition so that the Special Condition is satisfied, without increasing the width. For this purpose, we have developed *two alternative approaches* and compared them empirically. The first approach tries to *greedily repair* the decomposition in order to satisfy the Special Condition. The second approach utilizes the full encoding but adds *stratification clauses* so that the solution to the relaxed encoding directs the solver. If the modification fails, we enter the second phase, where we use the full encoding, including clauses for the Special Condition, but without any stratification clauses.

The two-phase approach provides a considerable speedup compared to using just the full encoding alone.

We implemented these ideas in our solver *HtdSMT* and tested them thoroughly. For comparison purposes, we used *TU Longo*, a recent implementation of the Gottlob-Samer algorithm.

We were able to show that our multi-phase approach performs well in practice. Especially our stratified approach yielded encouraging results.

2 Preliminaries

A *hypergraph* is a pair $H = (V, H)$, consisting of a set V of *vertices* and a set E of *hyperedges* (sometimes just referred to as *edges*), each hyperedge is a subset of $V(H)$.

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 $v \in e$. The *size* of an edge cover is its cardinality.

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 to each $t \in V(T)$ a set $\chi(t) \subseteq V$ (called the *bag* at t) such that the following properties hold:

- T1 for each $v \in V$ there is some $t \in V(T)$ with $v \in \chi(t)$ ("v is covered by t"),
- T2 for each $e \in E$ there is some $t \in V(T)$ with $e \subseteq \chi(t)$ ("e is covered by t"),
- T3 for any three $t, t', t'' \in V(T)$ where t' lies on the path between t and t'', we have $\chi(t) \cap \chi(t'') \subseteq \chi(t')$ ("bags containing the same vertex are connected").

Throughout this paper, the term *node* refers to an element of $V(T)$ and *vertex* refers to an element of V.

We will use the following well-known fact (see, e.g., [5]).

FACT 2.1. *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 a hypergraph and its primal graph share exactly the same tree 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* $\lambda(t) \subseteq E(H)$ of $\chi(t)$. The *width* of $\mathcal{G}$ is the size of a largest edge cover $\lambda(t)$ over all $t \in V(T)$, and the generalized hypertree width *ghtw*(H) of H is the smallest width over all generalized hypertree decompositions of H. It is already NP-hard to decide whether a given hypergraph has generalized hypertree width ≤ 2 [10]. To make the recognition of hypergraphs of bounded width tractable, one needs to strengthen the definition of generalized hypertree width by adding the aforementioned Special Condition as follows.

A *hypertree decomposition* [14] of H is a generalized hypertree decomposition $\mathcal{G} = (T, \chi, \lambda)$ of H where T is a

rooted tree that satisfies in addition to T1–T3 also a certain Special Condition (T4). To formulate this condition, we call a vertex v to be *omitted* at a node $t \in V(T)$, if $v \notin \chi(t)$, but $\lambda(t)$ contains a hyperedge e with $v \in e$. The *Special Condition* now states the following:

T4 If a vertex v is omitted at t, then it must not appear in the bag $\chi(t')$ of any descendant node t' of t.

In other words, T4 states that if $t, t' \in V(T)$ are nodes such that t' is a descendant of t, then for each $e \in \lambda(t)$ we have $(e \setminus \chi(t)) \cap \chi(t') = \emptyset$. The *hypertree width* $\mathit{htw}(H)$ of H is the smallest width over all hypertree decompositions of H. Clearly $\mathit{ghtw}(H) \leq \mathit{htw}(H)$.

To avoid trivial cases, we consider only hypergraphs $H = (V, E)$ where each $v \in V$ is contained in at least one $e \in E$. Consequently, every considered hypergraph H has an edge cover, and the parameters $\mathit{ghtw}(H)$ and $\mathit{htw}(H)$ are always defined. If $|V| = 1$ then $\mathit{htw}(H) = \mathit{ghtw}(H) = 1$.

In the following, we also assume that each considered hypergraph H is *connected* (i.e., its primal graph is connected). One can easily adapt our results to disconnected hypergraphs by operating component-wise.

3 Ordering Based Characterization of Hypertree Width

The first SAT encoding of treewidth was suggested by Samer and Veith [19]. It used an ordering based characterization of treewidth. More recent SAT encodings of treewidth are also ordering based [1, 3]. Fichte et al. [7] extended this approach to computing the fractional hypertree width of hypergraphs, which only required a relatively little modification of the treewidth encoding.

For hypertree width, however, it turns out that expressing the special condition within an ordering based characterization requires new ideas. Let us consider an example that illustrates one of the difficulties. In the existing ordering based characterizations, each linear ordering $(v_{\pi(1)}, \ldots, v_{\pi(n)})$ of the vertices of the (hyper)graph corresponds to a decomposition tree with n nodes, with $v_{\pi(n)}$ corresponding to the root of the tree. The bag $\chi(v_{\pi(i)})$ associated with the tree node $v_{\pi(i)}$ contains all the vertices $v_{\pi(j)}$ with $\pi(j) \geq \pi(i)$ such that $v_{\pi(i)}$ and $v_{\pi(j)}$ belong to the same hyperedge, plus some additional *fill-in vertices* that are required for the connectedness condition T3.

For instance, if we consider the simple hypergraph $H = (V, E)$ where $V = \{v_1, \ldots, v_n\}$ and $E = \{V\}$, the ordering $(v_1, \ldots, v_n)$ would correspond to a decomposition tree which is a path, with v_n at the root. The bags are the sets $\chi(v_i) = \{v_i, \ldots, v_n\}$. As there is only one hyperedge, the edge cover for each v_i must be $\lambda(v_i) = \{V\}$, in particular $\lambda(v_n) = \{V\}$. However, as v_1 is omitted at every node v_i, $2 \leq i \leq n$, the special condition forbids the vertex v_1 to be present in any of the bags $\chi(v_i)$ for $1 \leq i \leq n-1$. Hence there is no hypertree decomposition that corresponds to this ordering in the conventional sense.

We fix this problem by adapting the ordering to allow the elimination of several vertices at once, not just one after the other. We accomplish this by introducing an *equivalence relation* $\equiv$ where equivalent vertices are eliminated simultaneously.

The second issue with hypertree decompositions is the fact that fill-in vertices are not just required to satisfy the connectedness condition T3, but also to satisfy the Special Condition T4. Fill-in vertices for the sake of T3 can be handled easily in a deterministic fashion, by adding certain edges that are enforced by the ordering (cf. Condition O3 below). Fill-in vertices for the sake of T4, however, are not enforced deterministically by the ordering. Even worse, adding a fill-in vertex for the sake of T4 might invalidate T3, and hence more fill-in vertices need to be added for the sake of T3, and so forth. Therefore, our ordering based characterization of hypertree width involves also a *DAG (directed acyclic graph)* D, which contains more information than can be deterministically derived from the ordering alone. The arcs of D can represent fill-in vertices of either kind.

A tuple $(D, \prec, \equiv, \lambda)$ is a *generalized hypertree ordering* of a hypergraph $H = (V, E)$ if D is a DAG with vertex set V, $\prec$ is a topological ordering of D, $\equiv$ is an equivalence relation on V, and λ is a mapping that assigns to each vertex $v \in V$ a set $\lambda(v) \subseteq E$ of hyperedges, such that the following conditions are satisfied:

O1 For each pair $u \prec v$ of distinct vertices of H, if $u, v \in e$ for some $e \in E$, then D contains the arc (u, v).

O2 For each pair $u \prec v$ of distinct vertices of H, if $u \equiv v$, then D contains the arc (u, v).

O3 For any three vertices $u \prec v \prec w$ of H, if D contains the arcs (u, v) and (u, w), then it also contains the arc (v, w).

O4 For any three vertices $u \prec v \prec w$ of H, if $u \equiv v$ and D contains the arc (v, w), then it also contains the arc (u, w).

O5 For each vertex v of H, $\lambda(v)$ is an edge cover of the set

$$\chi_D(v) := [v]_{\equiv} \cup N_D^+(v).$$

Here, $[v]_{\equiv}$ denotes the equivalence class of $\equiv$ to which v belongs to, and $N_D^+(v)$ denotes the set of out-neighbors of v in D.

OBSERVATION 3.1. *If $u \equiv v$, then $\chi_D(u) = \chi_D(v)$.*

Proof. W.l.o.g, assume $u \prec v$. Then $u \equiv v$ implies that D contains the arc (u, v). Condition O4 implies that $N_D^+(v) \subseteq N_D^+(u)$, Condition O4 implies that $N_D^+(u) \subseteq N_D^+(v)$, hence $N_D^+(u) = N_D^+(v)$. Since $[u]_{\equiv} = [v]_{\equiv}$, the statement follows. □

Let $\mathit{Ord} = (D, \prec, \equiv, \lambda)$ be a generalized hypertree ordering of $H = (V, E)$. We define the *canonical tree* of Ord as the rooted tree T whose set of nodes is V and whose root r is the $\prec$-largest element of V. The edges of T are defined by letting the parent of each vertex $u \in V \setminus \{r\}$ to be the $\prec$-smallest vertex $v \in N_D^+(u)$.

The following observation shows that the tree T is well defined.

OBSERVATION 3.2. *If H is connected, then for each $v \in V \setminus \{r\}$ we have $N_D^+(v) \neq \emptyset$.*

Proof. Assume to the contrary that there is some $v \in V \setminus \{r\}$ with $N_D^+(v) = \emptyset$. Let G be the undirected graph with $V(G) = V$ and where $E(G)$ contains the edge $\{u, v\}$ whenever D contains the arc (u, v) or (v, u). Because of Condition O1, it follows that $P(H)$ is a spanning subgraph of G; hence G is connected. Let P be a shortest path in G that connects v with r, and let u be the $\prec$-smallest vertex on P. Since $N_D^+(v) = \emptyset$, it follows that $u \prec v$. Let x, y be the vertices that are adjacent with u on P. Since u is the $\prec$-smallest vertex on P, $u \prec x$ and $u \prec y$ follows. Hence D contains the arcs (u, x) and (u, y). Because of Condition O3, D contains one of the arcs (x, y) and (y, x); in any case, G contains the edge $s = \{x, y\}$. The edge s provides a shortcut for P, as we can remove u from P and connect x with y via s, obtaining a shorter path between v and r. This contradicts the assumption that P is a shortest path. Hence $N_D^+(v) \neq \emptyset$. □

We are now in the position to define hypertree orderings, which, as we will see, correspond to hypertree decompositions. A *hypertree ordering* of $H = (V, E)$ is a generalized hypertree ordering $\mathit{Ord} = (D, \prec, \equiv, \lambda)$ of H that satisfies the following *Special Condition*: For any two vertices u, v such that u is a descendant of v in the canonical tree T (consequently $u \prec v$) and any $e \in \lambda(v)$ we have $(e \setminus \chi(v)) \cap \chi(u) = \emptyset$.

The *width* of the (generalized) hypertree ordering $\mathit{Ord} = (D, \prec, \equiv, \lambda)$ is defined as $\max_{v \in V} |\lambda(v)|$.

THEOREM 3.1. *A hypergraph has (generalized) hypertree width W if and only if it has a (generalized) hypertree ordering of width W.*

In the next two sections, we will establish the theorem algorithmically, showing that we can efficiently translate between hypertree decompositions and hypertree orderings of the same width.

4 From Hypertree Decompositions to Hypertree Orderings

Let (T, χ, λ) be a hypertree decomposition of a hypergraph $H = (V, E)$ of width W.

We are going to define a hypertree ordering $\mathit{Ord} = (D, \prec, \equiv, \lambda')$ of the same width.

We call a vertex v to be *forgotten* at node $t \in V(T)$ if $v \in \chi(t)$ but $v \notin \chi(t')$ for the parent t' of t in T; all vertices in $\chi(r)$ are forgotten at r. Because of Conditions T1 and T3 of a tree decomposition, it follows that each vertex $v \in V$ is forgotten at exactly one node $t \in V(T)$ which we denote by $f(v)$.

Let $\equiv$ be the equivalence relation on V defined by letting two vertices u, v to be equivalent if and only if they are forgotten at the same node, i.e., if $f(u) = f(v)$.

Let $\prec^*$ be the partial order on V defined by $u \prec^* v$ if and only if $u \neq v$ and $f(u)$ is a descendant of $f(v)$ in T. We let $\prec$ to by an arbitrary but fixed total order that refines $\prec^*$.

OBSERVATION 4.1. *For any two vertices u, v such that $v \in \chi(f(u))$ and $u \not\equiv v$, we have $u \prec v$.*

Proof. Since v is not forgotten at $f(u)$, but belongs to $\chi(f(u))$, it must be forgotten at a node $f(v)$ which is an ancestor of $f(u)$, hence $u \prec^* v$, and consequently, $u \prec v$. □

To define D, we take as its arcs all the arcs (u, v) for $u, v \in V$ for which $u \prec v$ and $v \in \chi(f(u))$. By construction, $\prec$ is indeed a topological ordering of D. We observe that for all $u, v \in V$ with $u \equiv v$ we have $\chi_D(u) = \chi_D(v)$.

CLAIM 4.1. *D satisfies all the Conditions O1–O4 of a generalized hypertree ordering.*

Proof. To verify Condition O1, let $u, v \in V$ such that $u, v \in e$ for some $e \in E$ and $u \prec v$. Condition T2 implies that $u, v \in \chi(t)$ for some $t \in T$. Let $t_0 \in V(T)$ be a node with this property that is closest to the root. Consequently $t_0 = f(u)$ or $t_0 = f(v)$. Since $u \prec v$, $t_0 = f(u)$ follows of necessity. Thus we have $v \in \chi(f(u))$, and consequently D contains the arc (u, v).

To verify Condition O2, let $u, v \in V$ such that $u \equiv v$ and $u \prec v$. We have $f(u) = f(v)$, and $v \in \chi(u)$, hence D contains the arc (u, v).

To verify Condition O3, let $u, v, w \in V$ with $u \prec v \prec w$ such that D contains the arcs (u, v) and (u, w). From the definition of D it follows that $v, w \in \chi(f(u))$. The node $f(v)$ lies in T on the path between $f(u)$ and the root of T (possibly $f(u) = f(v)$). Since $v \prec w$, $w \in \chi(f(v))$. Consequently, D contains the arc (v, w).

To verify Condition O4, let $u, v, w \in V$ with $u \prec v \prec w$ such that $u \equiv v$ and D contains the arc (v, w). From the definition of D we infer that $w \in \chi(f(v))$. Since $u \equiv v$, we have $w \in \chi(f(u))$, which in conjunction with $u \prec w$ implies that D contains the arc (u, w). □

We define a labeling λ' of V by letting $\lambda'(v) = \lambda(f(v))$ for each $v \in V$. Since $\chi_D(v) = \chi(f(v))$, $\lambda'(v)$ is an edge

cover of $\chi_D(v)$. We conclude that *Ord* is indeed a generalized hypertree ordering of H of width W.

It remains to verify that the Special Condition holds. Let T' be the canonical tree of *Ord*.

CLAIM 4.2. *For any two vertices $u_1, u_2 \in V$ such that u_1 is a descendant of u_2 in T' (consequently $u_1 \prec u_2$), either (i) $f(u_1) = f(u_2)$, or (ii) $f(u_1)$ is a descendant of $f(u_2)$ in T.*

Proof. We show that the claim holds in the case that u_2 is the parent of u_1; the general claim follows then by induction. Assume $f(u_1) \neq f(u_2)$. This implies $u_1 \not\equiv u_2$. However, since u_2 is the parent of u_1, D contains the arc (u_1, u_2), hence $u_2 \in \chi_D(u_1) = \chi(f(u_1))$. Thus u_2 must be forgotten in T at some node $f(u_2) \neq f(u_1)$ that lies on the path between $f(u_1)$ and the root of T. In other words, $f(u_1)$ is a descendant of $f(u_2)$. □

Since $\lambda'(u_i) = \lambda(f(u_i))$ and $\chi_D(u_i) = \chi(f(u_i))$, for $1 \leq i \leq 2$, it follows that Condition T4 of the hypertree decomposition with respect to $f(u_1)$ and $f(u_2)$ implies the special condition for the hypertree ordering *Ord* with respect to u_1 and u_2.

We conclude that this direction of the translation works as claimed.

5 From Hypertree Orderings to Hypertree Decompositions

Now, to see the reverse translation, let *Ord* $= (D, \prec, \equiv, \lambda)$ be a hypertree ordering of a hypergraph $H = (V, E)$ of width W. Let T be the canonical tree rooted at r.

CLAIM 5.1. *$\mathcal{G} = (T, \chi_D, \lambda)$ is a hypertree decomposition of H of width W.*

Proof. We verify that $\mathcal{G}$ satisfies all the Conditions T1–T4. Condition T1 holds, since $v \in \chi_D(v)$ for every $v \in V$. Condition T2 holds as well: for any $e \in E$, if v is the $\prec$-smallest vertex such that $v \in e$, then $e \subseteq \chi_D(v)$ follows by Property O1 of D.

To verify Condition T3, assume to the contrary that the condition is violated with respect to a vertex v. In other words, the set of nodes $\{u \in V(T) \mid v \in \chi(u)\}$ induces in T more than one connected subtrees $T_1, \ldots, T_k$. Let T_i be the subtree with $v \in V(T_i)$, and let $j \in \{1, \ldots, k\} \setminus \{i\}$. Let r_j be the root of T_j. Since $[v]_\equiv \subseteq V(T_i)$, we have $[v]_\equiv \cap V(T_j) = \emptyset$. Consequently, for every node $u \in V(T_j)$, we have $v \in \chi(u)$ but $u \notin [v]_\equiv$, hence D contains the arc (u, v). This implies that r_j is not the root of T, and so it has a parent p_j in T. By construction, $v \notin \chi(p_j)$. Since by definition of the canonical tree, p_j is the $\prec$-smallest vertex for which D contains the arc (r_j, p_j), but since D contains (r_j, v), we conclude that $p_j \prec v$. Hence, by Condition O3 of a hypertree ordering, it follows that D also contains the arc (p_j, v), but then $v \in \chi(p_j)$, a contradiction to our assumption. Hence we conclude that Condition T3 holds as well. Condition T4 holds as the Special Condition for *Ord* directly translates into T4 via the canonical tree. It remains to observe that by construction, the width of $\mathcal{G}$ is W. □

We conclude that the reverse direction of the translation works as well.

6 SMT Encoding

In this section, we describe how we can express the existence of a hypertree ordering of a hypergraph H of width $\leq W$ in terms of an *SMT formula*, which is a conjunction of propositional clauses and arithmetic constraints. Given H and W we proceed in two steps, first producing an SMT-formula $F'(H, W)$, which is satisfiable if an only if *ghtw*$(H) \leq W$, and secondly, adding additional clauses to obtain an SMT formula $F(H, W)$, which is satisfiable if an only if *htw*$(H) \leq W$.

6.1 Encoding Generalized Hypertree Width Let $H = (V, E)$ be the given hypergraph with n vertices $v_1, \ldots, v_n$.

For expressing the ordering $\prec$ we use $\binom{n}{2}$ Boolean variables $o_{i,j}$ for $1 \leq i < j \leq n$ (similarly as Samer and Veith [19] did for treewidth), where $o_{i,j}$ is true if and only if $i < j$ and $v_i \prec v_j$. We use the notation $o^*_{i,j}$ defined as follows:

$$o^*_{i,j} := \begin{cases} o_{i,j} & \text{if } i < j; \\ \overline{o}_{j,i} & \text{otherwise.} \end{cases}$$

To enforce that $\prec$ is indeed a linear ordering, we must ensure transitivity, which we establish by adding for all mutually distinct $1 \leq i, j, k \leq n$ the clause

$$\overline{o^*}_{i,j} \vee \overline{o^*}_{j,k} \vee o^*_{i,k}.$$

For expressing the equivalence relation $\equiv$ we use $\binom{n}{2}$ Boolean variables $e_{i,j}$ for $1 \leq i < j \leq n$, where $e_{i,j}$ is true if and only if $i < j$ and $v_i \equiv v_j$. We use the notation $e^*_{i,j}$ defined as follows:

$$e^*_{i,j} := \begin{cases} e_{i,j} & \text{if } i < j; \\ e_{j,i} & \text{otherwise.} \end{cases}$$

Also here we must ensure transitivity (within each equivalence class, not over the entire set V as for $o_{i,j}$ above), which we accomplish by adding for all mutually distinct $1 \leq i, j, k \leq n$ the clause

$$\overline{e^*}_{i,j} \vee \overline{e^*}_{j,k} \vee e^*_{i,k}.$$

For expressing the DAG D we use $n(n-1)$ Boolean variables $a_{i,j}$, for distinct $1 \leq i, j \leq n$, where $a_{i,j}$ is true if and only if D contains the arc (v_i, v_j). First we ensure that $\prec$ is a

topological ordering of D by adding for each pair of distinct $1 \leq i, j \leq n$ the clause

$$\overline{o^*}_{i,j} \vee \overline{a}_{j,i}.$$

Furthermore, we add clauses that correspond to Conditions O1–O4 of the definition of a generalized hypertree ordering. That is, for Condition O1 we add for all distinct $1 \leq i, j \leq n$, such that some $e \in E$ contains both v_i and v_j, the clause

$$\overline{o^*}_{i,j} \vee a_{i,j}.$$

For Condition O2, we add for all distinct $1 \leq i, j \leq n$ the clause

$$\overline{o^*}_{i,j} \vee \overline{e^*}_{i,j} \vee a_{i,j}.$$

For Condition O3, we add for all three mutually distinct $1 \leq i, j, k \leq n$ the clause

$$\overline{o^*}_{j,k} \vee \overline{a}_{i,j} \vee \overline{a}_{i,k} \vee a_{j,k}.$$

For Condition O4, we add for all three mutually distinct $1 \leq i, j, k \leq n$ the clause

$$\overline{o^*}_{i,j} \vee \overline{o^*}_{j,k} \vee \overline{e^*}_{i,j} \vee \overline{a}_{j,k} \vee a_{i,k}.$$

In order to encode Condition O5, we first need a representation of the bags $\chi_D(v_i)$ and edge covers $\lambda(v_i)$, for $1 \leq i \leq n$. Accordingly, we introduce a Boolean variable $b_{i,j}$ for each pair of (not necessarily distinct) $1 \leq i, j \leq n$, which is true if and only if $v_j \in \chi_D(v_i)$. For all distinct $1 \leq i, j \leq n$ we add the three clauses

$$\overline{a}_{i,j} \vee b_{i,j}, \quad \overline{e^*}_{i,j} \vee b_{i,j}, \quad \overline{b}_{i,j} \vee a_{i,j} \vee e^*_{i,j},$$

representing the equivalence $a_{i,j} \vee e^*_{i,j} \leftrightarrow b_{i,j}$, and for all $1 \leq i \leq n$ we add the unit clause $b_{i,i}$.

In order to represent the edge covers, we introduce for each $1 \leq i \leq n$ and each $e \in E$ an integer-valued *weight variable* $w_{i,e}$, which has the value 1 if $e \in \lambda(v_i)$ and the value 0 otherwise. Although this could be done also by Boolean variables, we use integer-valued variables as we can use arithmetic constraints for ensuring that each vertex in a bag is covered (Condition O5) and for bounding the width of the ordering. For encoding Condition O5 in a compact way, we use the fact that for each vertex $v \in V$ there is a vertex $v_i \in V$, the $\prec$-smallest vertex in $[v]_{\equiv}$, with $\chi_D(v) = \chi_D(v_i) = N_D^+(v_i) \cup \{v_i\}$. Hence it suffices to check that $\lambda(v_i)$ covers all the vertices $v_j \in N_D^+(v_i)$ and the vertex v_i. We accomplish this by adding for all distinct $1 \leq i, j \leq n$ the arithmetic constraint

$$\overline{a}_{i,j} \vee \sum_{e \in E, v_j \in e} w_{i,e} \geq 1,$$

and for all $1 \leq i \leq n$ the arithmetic constraint

$$\sum_{e \in E, v_i \in e} w_{i,e} \geq 1.$$

Finally, for bounding the width, we add for each $1 \leq i \leq n$ the arithmetic constraint

$$\sum_{e \in E} w_{i,e} \leq W.$$

All the clauses and arithmetic constraints added so far ensure that the formula is satisfiable if and only if H has a generalized hypertree decomposition of width $\leq W$. It remains to encode the Special Condition.

6.2 Encoding the Special Condition For expressing the Special Condition, we need to represent the canonical tree. To this aim, we introduce n^2 Boolean variables $p_{i,j}$ for $1 \leq i, j \leq n$ where for $i \neq j$, $p_{i,j}$ is true if and only if v_j is the parent of v_i in the canonical tree, and $p_{i,i}$ is true if and only if v_i is the root of the canonical tree. We add the clause

$$\bigvee_{i=1}^{n} p_{i,i}$$

stating that at least one node v_i is the root of the canonical tree, and we add for all distinct $1 \leq i, j \leq n$ the clause

$$\overline{o^*}_{i,j} \vee \overline{p}_{i,i}$$

stating that the root must be the $\prec$-largest vertex. Next, we add for each $1 \leq i \leq n$ the clause

$$\bigvee_{j=1}^{n} p_{i,j}$$

stating that each node has a parent in the canonical tree or is the root. Finally, we enforce that the parent of v_i is the $\prec$-smallest vertex $v_j \in N_D^+(v_i)$ by adding for any three distinct $1 \leq i, j, k \leq n$ the clause

$$\overline{a}_{i,j} \vee \overline{o^*}_{j,k} \vee \overline{p}_{i,k}.$$

We represent the descendancy relation of the canonical tree with $n(n-1)$ Boolean variables $d_{i,j}$ for distinct $1 \leq i, j \leq n$, where $d_{i,j}$ is true if v_j is a descendant of v_i in the canonical tree. We obtain the descendancy relation as the transitive closure of the parent relation. The base case is handled by adding for all distinct $1 \leq i, j \leq n$ the clause

$$\overline{p}_{i,j} \vee d_{i,j},$$

and the inductive step is handled by adding for all three distinct $1 \leq i, j, k \leq n$ the clause

$$\overline{d}_{i,j} \vee \overline{d}_{j,k} \vee d_{i,k}.$$

To enforce that the $d_{i,j}$-variables represent the smallest relation that contains the transitive closure of the parent relation, we add for any three distinct $1 \leq i, j, k \leq n$ the clause

$$\overline{p}_{i,j} \vee d_{j,k} \vee \overline{d}_{i,k},$$

stating that if the parent is not a descendant, than neither are the children, as well as for any two distinct $1 \leq i, j \leq n$ the clause

$$\overline{o}^*_{i,j} \vee \overline{d}_{j,i},$$

stating that the descendancy relation respects the ordering *Ord*. In addition, we introduce n^2 auxiliary variables $c_{i,j}$, for $1 \leq i, j \leq n$, where $c_{i,j}$ is true if $v_j \in \bigcup_{e \in \lambda(v_i)} e$. We enforce this by adding for all $1 \leq i, j \leq n$ and $e \in E$ such that $v_j \in e$ the clause

$$(w_{i,e} = 0) \vee c_{i,j},$$

and the arithmetic constraint

$$\overline{c}_{i,j} \vee \sum_{e \in E, v_j \in e} w_{e,i} \geq 1.$$

We can now enforce the Special Condition by adding for all $1 \leq i, j \leq n$ and all $k \in \{1, \ldots, n\} \setminus \{i, j\}$ the clause

$$\overline{d}_{i,j} \vee \overline{c}_{j,k} \vee \overline{b}_{i,k} \vee b_{j,k}.$$

This now completes the encoding of H having hypertree width $\leq W$.

7 Symmetry Breaking

In general, there can be many hypertree orderings that correspond to the same (generalized) hypertree decomposition. In Section 4, when we translated a hypertree decomposition into a hypertree ordering, we had the choice of picking any total order $\prec$ that refines the partial order $\prec^*$. However, if we fix a lexicographic order of the vertices, say $v_1, \ldots, v_n$, then there exists a *unique* linear order $\prec$ that refines $\prec^*$. Algorithmically, we can think of obtaining $\prec$ by successively deleting vertices until all vertices have been deleted, by choosing the next vertex to delete as the lexicographically smallest among all the $\prec^*$-smallest vertices, similarly to the *Topological Sorting* procedure. Since we have exactly one choice at each step, the obtained $\prec$ is unique.

To enforce in the SMT encoding that $\prec$ is this unique ordering, we introduce for all $1 \leq i < j \leq n$ and each $k \in \{1, \ldots n\} \setminus \{i, j\}$ a Boolean variable $\ell_{i,j,k}$. The variable $\ell_{i,j,k}$ is true if and only if $v_j \prec v_k \prec v_i$, and there is an arc (v_k, v_i). This means that we can pick v_j in the deletion procedure sketched above before v_i, although v_i is lexicographically smaller than v_j, since the arc (v_k, v_j) prohibits us choosing v_i earlier.

Consequently, for all $1 \leq i < j \leq n$ and all $k \in \{1, \ldots n\} \setminus \{i, j\}$ we add the following clauses

$$\overline{o}^*_{i,k} \vee \overline{a}_{k,i} \vee \ell_{i,j,k}, \quad \overline{\ell}_{i,j,k} \vee o^*_{j,k}, \quad \overline{\ell}_{i,j,k} \vee a_{k,i},$$

whose conjunction is logically equivalent to the equivalence $\ell_{i,j,k} \leftrightarrow o^*_{j,k} \wedge a_{k,i}$. Finally, for all $1 \leq i < j \leq n$, we add the clause

$$o_{i,j} \vee a_{j,i} \vee \bigvee_{k \in \{1,\ldots,n\} \setminus \{i,j\}} \ell_{i,j,k},$$

which states that if $v_j \prec v_i$ although $i < j$, then there must be an arc from v_j to v_i or an arc from v_k to v_i for some $k > j$.

In our experiments (see Section 9), we compared the performance of our encoding with and without symmetry breaking. It turned out that symmetry breaking gives a significant speedup for the computation of hypertree width via the full encoding. Interestingly, for the encoding of generalized hypertree width, the symmetry breaking clauses impaired the performance.

8 Utilizing Generalized Hypertree Decompositions

Due to the often considerably higher efforts of computing a hypertree decomposition compared to its generalized variant, we developed an alternative approach for computing hypertree decompositions. Instead of using the full encoding $F(H, W)$ for hypertree width as described above, we use the encoding $F'(H, W)$ for generalized hypertree width, and try to utilize a satisfying assignment of it to obtain a hypertree decomposition of the same width. Only if this fails, we use the full encoding $F(H, W')$, with $W' \geq W$. Next, we discuss our two strategies for utilizing generalized hypertree decompositions.

8.1 Greedy Repair The first strategy is to generate an optimal generalized hypertree ordering using the SMT encoding $F'(H, W)$, and transform it into a generalized hypertree decomposition (T, χ, λ) of width W utilizing the method laid out in Section 5. Then, we modify the tree T by trying out each of its nodes as the root. For each such choice, we extend the labelings χ and λ monotonically, possibly adding vertices to bags and adding hyperedges to edge covers, until we either obtain a correct hypertree decomposition of width W, or fail:

1. As long as Condition T4 is violated, i.e., if for some $t \in V(T)$ there is a vertex v that is omitted at t but is contained in $\chi(t')$ for some descendant t' of t, we add v to $\chi(t)$. Adding v to $\chi(t)$ might cause Condition T3 to be violated, or that $\lambda(t)$ is no longer an edge cover of $\chi(t)$.

2. As long as Condition T3 is violated, i.e., if for $t, t', t'' \in V(T)$ where t' lies on the path between t and t'', we

have some $v \in \chi(t) \cap \chi(t'')$, but $v \notin \chi(t')$, we add v to $\chi(t')$. Adding v to $\chi(t')$ might cause Condition T4 to be violated, or that $\lambda(t')$ is no longer an edge cover of $\chi(t')$.

3. As long as there exists a tree node t such that $\lambda(t)$ is not an edge cover of $\chi(t)$, we add hyperedges to $\lambda(t)$ until $\chi(t)$ is covered. Adding hyperedges to $\lambda(t)$ might cause the size of $\lambda(t)$ to exceed W.

4. We fail if $|\lambda(t)| > W$ for some $t \in V(T)$; otherwise we start over at Step 1.

We note that as long as any condition is violated, bags are extended, and if all conditions hold and the width bound W has not been exceeded, we have found a valid hypertree decomposition.

8.2 Stratified Encoding With the second strategy, we also start by generating a hypertree ordering using the SMT encoding. We now utilize some information from a satisfying assignment τ of $F'(H, W)$ and add additional *stratification clauses* $S(\tau)$ to $F(H, W)$. If it still holds that $F(H, W) \wedge S(\tau)$ is satisfiable, then $htw(H) = W$; the converse, however, might not be true any more. So, if $F(H, W) \wedge S(\tau)$ is unsatisfiable, then this approach has failed. The clauses in $S(\tau)$ reflect some information regarding the generalized hypertree ordering $Ord = (D, \prec, \equiv, \lambda)$ of H that is represented by τ. Now the search for a satisfying assignment for $F(H, W)$ is guided by this information.

We implemented and tested the following variants $S_i(\tau)$ for stratification clauses, ordered from the most restrictive to the least and restrictive.

Fixing the ordering. $S_1(\tau)$ is the conjunction of all the unit clauses $o^*_{i,j}$, where τ sets $o^*_{i,j}$ to true, for $1 \leq i, j \leq n$. Therefore, $S_1(\tau)$ forces the solver, to maintain the full ordering.

Fixing the tree structure. $S_2(\tau)$ is the conjunction of all the clauses $p_{i,j} \vee p_{j,i}$, where $j \neq i$ and τ sets $p_{i,j}$ to true, for $1 \leq i, j \leq n$. This suggests a structure for the canonical tree, but allows the solver to choose the root and orientation.

Fixing the bags. $S_3(\tau)$ is the conjunction of all the unit clauses $b_{i,j}$, where τ sets $b_{i,j}$ to true, for $1 \leq i, j \leq n$. The bags are thereby initialized with the same vertices as in the generalized hypertree decomposition. The solver can extend the bags and change the ordering.

Fixing the arcs. $S_4(\tau)$ is the conjunction of all the clauses $a_{i,j} \vee a_{j,i}$, where τ sets $a_{i,j}$ to true, for $1 \leq i, j \leq n$. In other words, $S_4(\tau)$ forces the solver to keep the arcs from D but allows to invert their orientation.

For each of these stratification strategies, the formula $F(H, W) \wedge S_i(\tau)$ can generally be solved faster than $F(H, W)$. In our experiments, S_4 performed best, as we will discuss in detail in the next section.

9 Experiments and Results

9.1 Experimental Setup We implemented the ideas and concepts presented above in our solver *HtdSMT*[1]. We implemented both encoder and decoder in Python 2.7. Their running time is relatively uncritical, as the main work is carried out by the SMT solver. In principle, any SMT solver can be used. We tested *optimathsat*[2] and *Z3*[3]. Due to better results with the former, we used it in the final version. Data between encoder and SMT solver is passed using files to avoid the overhead of the Python interface.

We used nodes with two Intel Xeon E5540 CPUs with 2.53 GHz each. Each node ran Ubuntu 16.04, Python 2.7.12, gcc 5.5.0 20171010 and optimathsat 1.6.2. Each run was limited to 8 GB of RAM and thirty minutes runtime.

9.2 Benchmark Instances We tested our solver against the 100 public instances of the 2019 PACE (Parameterized Algorithms and Computational Experiments) challenge[4]. The instances are a subset of the Hyperbench collection[5] [9]. The PACE organizers [6] selected the instances based on the expected runtime as follows. A heuristic method solved the instances with a relaxed Special Condition. The instances were then divided into the categories *easy* (solved within 60 seconds), *medium* (solved in 300 to 900 seconds), and *hard* (not solved within the 7200 second time limit). The gaps in the definitions are deliberate. Finally, 10 easy, 30 medium, and 60 hard instances were randomly selected. For all instances, we were able to solve, the hypertree width turned out to be equal to the generalized hypertree width.

9.3 Comparison with the Gottlob-Samer Algorithm To provide a comparison with the state-of-the-art, we compared our algorithm with *TU Longo*[6], a recent implementation of the Gottlob-Samer algorithm [15]. *TU Longo* participated besides *HtdSMT* in the 2019 PACE challenge [6], where (according to preliminary results) it reached second place in both the exact and heuristic track. In the exact track, an earlier version of *HtdSMT* reached the first place. For our experiments we used the same setup and instances as the PACE competition, and provide more detailed information and results on variants of our approach to provide further insight on the performance of individual techniques.

[1] https://github.com/ASchidler/frasmt_pace/ (experimental results of this paper are based on Commit *5354ef8*)
[2] http://optimathsat.disi.unitn.it
[3] https://z3prover.github.io
[4] https://doi.org/10.5281/zenodo.3354607
[5] http://hyperbench.dbai.tuwien.ac.at
[6] https://doi.org/10.5281/zenodo.3236369

9.4 Results In Tables 1, 2, 3, and 5 we provide primary data of our experiments:

1. the number of instances out of the selected 100 PACE instances a particular decomposition algorithm could solve within the time limit of 30 minutes;

2. the average running time per instance over the instances solved by all the methods listed in the table.

Table 1 compares the performance of the three different strategies implemented in *HtdSMT*. The Stratified Encoding can solve the most instances, while the Greedy Repair approach is the fastest method. Although the stratified encoding may seem superior, we need all three methods to solve the maximum number of instances, as each method can solve instances, that none of the others can solve. Together, all three methods can solve a total of 75 different instances.

Approach	Solved	Av. Time [s]
Greedy Repair	61 / 100	25.55
Stratified Encoding	72 / 100	45.14
HTD Encoding	62 / 100	130.84

Table 1: Comparison of different solving strategies discussed in this paper.

Table 2 shows a comparison of the four different stratification strategies, as discussed in Section 8.2. S_4 outperforms the other methods in both runtime and number of solved instances and is, therefore, the method we use for *HtdSMT*.

The SMT solver in our tests was able to find a *generalized* hypertree ordering for 80 instances. The success rate of the stratified encoding, under the precondition that a generalized hypertree ordering is available, is therefore remarkably high (approximately 90 %).

Stratification Strategy	Solved	Av. Time [s]
$S_4(\tau)$ - Arcs	72 / 100	90.04
$S_1(\tau)$ - Ordering	69 / 100	85.59
$S_3(\tau)$ - Bags	69 / 100	90.90
$S_2(\tau)$ - Tree	68 / 100	86.06

Table 2: Comparison of different stratification strategies, as discussed in Section 8.2.

In Table 3, we compare *HtdSMT* with *TU Longo*. For this comparison, *HtdSMT* executed the three solving strategies serially: first, the greedy repair, then the stratified encoding and finally the full encoding. This modified version was able to solve 73 instances within the 30-minute time limit. We see that *HtdSMT* can solve considerably more instances in a shorter amount of time compared to *TU Longo*.

Name	Solved	Av. Time [s]
HtdSMT	73 / 100	7.56
TU Longo	35 / 100	138.63

Table 3: Comparison of different solvers.

It is interesting to see the performance differences between *HtdSMT* and *TU Longo* with respect to the magnitude of the hypertree width of the given instance. Table 4 shows the results of the different strategies and solvers grouped by hypertree width. *TU Longo* performs best for lower widths, but the performance decreases drastically the higher the width becomes. In contrast, our algorithm *HtdSMT* performs equally well on higher widths.

	Hypertree Width				
Solver	2	3	4	5	6
Stratified Encoding	8	9	7	29	19
Full Encoding	7	9	9	26	11
Greedy Repair	7	9	7	26	12
TU Longo	9	11	8	7	0

Table 4: Comparison of different solvers and solving strategies grouped by hypertree width.

The *cactus plot* in Figure 1 provides further information on how the various solving approaches scale with respect to the given time limit.

Finally, Table 5 compares the performance of the SMT encodings with and without symmetry breaking (see Section 7). For hypertree decompositions, symmetry breaking gives us a significant speedup (approximately 15 %). For generalized hypertree decompositions, however, the addition of the symmetry breaking clauses has a negative effect. This effect can be explained by the significant increase in the *relative* encoding size, as can be seen from Table 5.

Name	Solved	Av. Time [s]	Av. Size [MB]
HTD SB	62 / 100	150.22	254.20
HTD	62 / 100	175.39	161.01
GHTD SB	73 / 100	77.05	132.27
GHTD	80 / 100	46.91	39.08

Table 5: Impact of symmetry breaking (SB) in conjunction with computing hypertree decompositions (HTD) and generalized hypertree decompositions (GHTD). The column *Size* shows the average size of the encoding file taken over all 100 instances.

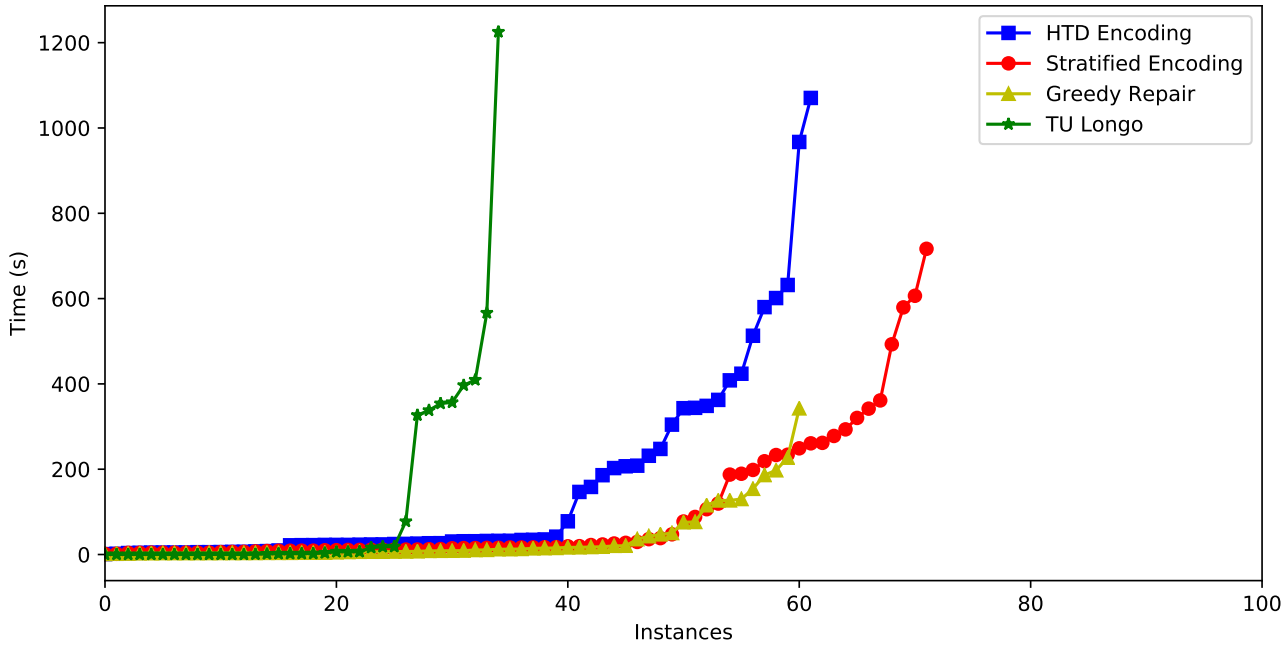


Figure 1: Comparison of different solving approaches.

10 Concluding Remarks

We have presented a new approach for practically computing the hypertree width of hypergraphs and evaluated them experimentally on benchmark instances. Our approach uses a new ordering based characterization of hypertree width, which leads to an efficient SMT encoding. For further speedup, we developed a two-phase approach, which first computes a generalized hypertree decomposition, and utilizes parts of its structure for computing the hypertree width. We realized this alternative approach in two ways: with a Greedy Repair algorithm, and with a Stratified Encoding. The latter is a novel encoding technique, where a satisfying assignment for a partial encoding is used to generate stratification clauses for guiding the solver on the full encoding. Our results are highly encouraging. Our methods outperform the state-of-the-art algorithm, solving 73 out of 100 benchmark instances compared to 35. In particular, on instances of medium to large hypertree width, our approach outperforms the existing approach.

As future work, we have several ideas for further improvements. For instance, the preprocessing methods developed by Fichte et al. [7] for fractional hypertree width could be adapted for a use within *HtdSMT*. Especially the hyper-clique heuristic should provide a performance increase for generalized hypertree width so provide the means for speeding up the Greedy Repair and Stratified Encoding approaches; it doesn't apply directly to hypertree width, as it implicitly determines the root of the decomposition tree.

Another interesting avenue for future research is to scale our approach to larger hypergraphs through SAT-based local improvement methods which have recently provided encouraging results for branchwidth and treewidth [17, 8].

References

[1] Max Bannach, Sebastian Berndt, and Thorsten Ehlers. Jdrasil: A modular library for computing tree decompositions. In Costas S. Iliopoulos, Solon P. Pissis, Simon J. Puglisi, and Rajeev Raman, editors, *16th International Symposium on Experimental Algorithms, SEA 2017, June 21-23, 2017, London, UK*, volume 75 of *LIPIcs*, pages 28:1–28:21. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[2] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185, pages 825–885. IOS Press, 2009.

[3] Jeremias Berg and Matti Järvisalo. 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*, pages 328–335. IEEE Computer Society, 2014.

[4] Hans L. Bodlaender. Discovering treewidth. In *Proceedings of the 31st Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'05)*, volume 3381 of *Lecture Notes in Computer Science*, pages 1–16. Springer Verlag, 2005.

[5] Hans L. Bodlaender and Rolf H. Möhring. The pathwidth and treewidth of cographs. *SIAM J. Discrete Math.*, 6(2):181–188, 1993.

[6] Muhammad Ayaz Dzulfikar, Johannes Klaus Fichte, and Markus Hecher. The PACE 2019 parameterized algorithms and computational experiments challenge: The fourth iteration. In *14th International Symposium on Parameterized and Exact Computation, IPEC 2019*, 2020 forthcoming.

[7] Johannes Klaus Fichte, Markus Hecher, Neha Lodha, and Stefan Szeider. An SMT approach to fractional hypertree width. In *Proc. CP'18*, volume 11008 of *Lecture Notes in Computer Science*, pages 109–127. Springer Verlag, 2018.

[8] Johannes Klaus Fichte, Neha Lodha, and Stefan Szeider. SAT-based local improvement for finding tree decompositions of small width. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, pages 401–411. Springer, 2017.

[9] Wolfgang Fischl, Georg Gottlob, Davide Mario Longo, and Reinhard Pichler. Hyperbench: A benchmark and tool for hypergraphs and empirical findings. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, New York, NY, USA, 2019*, pages 464–480. Assoc. Comput. Mach., New York, 2019.

[10] Wolfgang Fischl, Georg Gottlob, and Reinhard Pichler. General and Fractional Hypertree Decompositions. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems - SIGMOD/PODS '18*, pages 17–32, New York, New York, USA, 2018. ACM Press.

[11] Robert Ganian, Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. SAT-encodings for treecut width and treedepth. In Stephen G. Kobourov and Henning Meyerhenke, editors, *Proceedings of ALENEX 2019, the 21st Workshop on Algorithm Engineering and Experiments*, pages 117–129. SIAM, 2019.

[12] Georg Gottlob, Gianluigi Greco, and Francesco Scarcello. Treewidth and hypertree width. In Lucas Bordeaux, Youssef Hamadi, and Pushmeet Kohli, editors, *Tractability: Practical Approaches to Hard Problems*, pages 3–38. Cambridge University Press, 2014.

[13] Georg Gottlob, Martin Grohe, Nysret Musliu, Marko Samer, and Francesco Scarcello. Hypertree decompositions: Structure, algorithms, and applications. In Dieter Kratsch, editor, *Proceedings of the 31st International Workshop on Graph-Theoretic Concepts in Computer Science (WG'05)*, volume 3787 of *Lecture Notes in Computer Science*, pages 1–15. Springer Verlag, 2005.

[14] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *J. of Computer and System Sciences*, 64(3):579–627, 2002.

[15] Georg Gottlob and Marko Samer. A backtracking-based algorithm for hypertree decomposition. *J. Exp. Algorithmics*, 13:1:1.1–1:1.19, February 2009.

[16] Marijn Heule and Stefan Szeider. A SAT approach to clique-width. *ACM Trans. Comput. Log.*, 16(3):24, 2015.

[17] Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. A SAT approach to branchwidth. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 179–195. Springer Verlag, 2016.

[18] David Monniaux. A survey of satisfiability modulo theory. In Vladimir P. Gerdt, Wolfram Koepf, Werner M. Seiler, and Evgenii V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing - 18th International Workshop, CASC 2016, Bucharest, Romania, September 19-23, 2016, Proceedings*, volume 9890 of *Lecture Notes in Computer Science*, pages 401–425. Springer Verlag, 2016.

[19] Marko Samer and Helmut Veith. 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*, volume 5584 of *Lecture Notes in Computer Science*, pages 45–50. Springer Verlag, 2009.

[20] Carsten Sinz. Towards an optimal cnf encoding of boolean cardinality constraints. In Peter van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*, volume 3709 of *Lecture Notes in Computer Science*, pages 827–831. Springer Verlag, 2005.