Soft and Constrained Hypertree Width

MATTHIAS LANZINGER, Institute of Logic and Computation, TU Wien, Austria
CEM OKULMUS, Institute of Computer Science, Paderborn University, Germany
REINHARD PICHLER, Institute of Logic and Computation, TU Wien, Austria
ALEXANDER SELZER, Institute of Logic and Computation, TU Wien, Austria
GEORG GOTTLOB, Dipartimento di Matematica e Informatica, University of Calabria, Italy

Hypertree decompositions provide a way to evaluate Conjunctive Queries (CQs) in polynomial time, where the exponent of this polynomial is determined by the width of the decomposition. In theory, the goal of efficient CQ evaluation therefore has to be a minimisation of the width. However, in practical settings, it turns out that there are also other properties of a decomposition that influence the performance of query evaluation. It is therefore of interest to restrict the computation of decompositions by constraints and to guide this computation by preferences. To this end, we propose a novel framework based on candidate tree decompositions, which allows us to introduce soft hypertree width (shw). This width measure is a relaxation of hypertree width (hw); it is never greater than hw and, in some cases, shw may actually be lower than hw. Most importantly, shw preserves the tractability of deciding if a given CQ is below some fixed bound, while offering more algorithmic flexibility. In particular, it provides a natural way to incorporate preferences and constraints into the computation of decompositions. A prototype implementation and preliminary experiments confirm that this novel framework can indeed have a practical impact on query evaluation.

CCS Concepts: • **Theory of computation** → **Database query processing and optimization (theory)**; • **Information systems** → *Relational database query languages*; • **Mathematics of computing** → *Hypergraphs.*

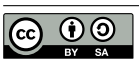
Additional Key Words and Phrases: hypergraph decomposition, hypertree width, constrained decompositions

ACM Reference Format:
Matthias Lanzinger, Cem Okulmus, Reinhard Pichler, Alexander Selzer, and Georg Gottlob. 2025. Soft and Constrained Hypertree Width. *Proc. ACM Manag. Data* 3, 2 (PODS), Article 114 (May 2025), 25 pages. https://doi.org/10.1145/3725251

1 Introduction

Since their introduction nearly 25 years ago, hypertree decompositions (HDs) and hypertree width (hw) have emerged as a cornerstone in the landscape of database research. Their enduring relevance lies in their remarkable ability to balance theoretical elegance with practical applicability. Hypertree width generalises α-acyclicity, a foundational concept for the efficient evaluation of conjunctive queries (CQs) [31]. Indeed, CQs of bounded hypertree width can be evaluated in polynomial time. Crucially, determining whether a CQ has $hw \leq k$ for fixed $k \geq 1$ and, if so, constructing an HD of width $\leq k$ can also be achieved in polynomial time [17].

Authors' Contact Information: Matthias Lanzinger, Institute of Logic and Computation, TU Wien, 1040 Wien, Austria, matthias.lanzinger@tuwien.ac.at; Cem Okulmus, Institute of Computer Science, Paderborn University, 33102 Paderborn, Germany, cem.okulmus@uni-paderborn.de; Reinhard Pichler, Institute of Logic and Computation, TU Wien, 1040 Wien, Austria, pichler@dbai.tuwien.ac.at; Alexander Selzer, alexander.selzer@tuwien.ac.at, Institute of Logic and Computation, TU Wien, 1040 Wien, Austria; Georg Gottlob, Dipartimento di Matematica e Informatica, University of Calabria, 87036 Arcavacata di Rende, Italy, georg.gottlob@sjc.ox.ac.uk.



ACM 2836-6573/2025/5-ART114
https://doi.org/10.1145/3725251

Over the decades, hypertree width has inspired the development of generalised hypertree width (ghw) [18] and fractional hypertree width (fhw) [22], each extending the boundaries of tractable CQ evaluation. These generalisations induce larger classes of tractable CQs. Notably, the hierarchy

$$fhw(q) \leq ghw(q) \leq hw(q)$$

holds for any CQ q, with the respective width measure determining the exponent of the polynomial bound on the query evaluation time. Consequently, practitioners naturally gravitate towards width measures that promise the smallest possible decomposition widths. Despite this motivation, HDs and hw retain a crucial distinction: hw is, to date, the most general width measure for it can be decided in polynomial time [17] if the width of a given query is below some fixed bound k. By contrast, generalised and fractional hypertree width, while often yielding smaller width values, suffer from computational intractability in their exact determination [16, 18]. In this work, we introduce a novel width notion—soft hypertree width (shw) – that overcomes this trade-off by retaining polynomial-time decidability while potentially reducing widths compared to hw.

But this is only a first step towards a larger goal. Query evaluation based on decomposition methods works via a transformation of a given CQ with low width into an acyclic CQ by computing "local" joins for the bags at each node of the decomposition. The acyclic CQ is then evaluated by Yannakakis' algorithm [31]. From a theoretical perspective, the complexity of this approach solely depends on the width, while the actual cost of the local joins and the size of the relations produced by these joins is ignored. To remedy this short-coming, Scarcello et al. [28] introduced *weighted HDs* to incorporate costs into HDs. More precisely, among the HDs with minimal width, they aimed at an HD that minimises the cost of the local joins needed to transform the CQ into an acyclic one and the cost of the (semi-)joins required by Yannakakis' algorithm. Promising first empirical results were obtained with this approach.

However, there is more to decomposition-based query optimisation than minimising the width. For instance, $hw = 2$ means that we have to compute joins of two relations to transform the given query into an acyclic one. Regarding width, it makes no difference if the join is a Cartesian product of two completely unrelated relations or a comparatively cheap join along a foreign key relationship. This phenomenon is, in fact, confirmed by Scarcello et al. [28] when they report on decompositions with higher width but lower cost in their experimental results. Apart from the pure join costs at individual nodes or between neighbouring nodes in the decomposition, there may be yet other reasons why a decomposition with slightly higher width should be preferred. For instance, in a distributed environment, it makes a huge difference, if the required joins and semi-joins are between relations on the same server or on different servers. In other words, apart from incorporating costs of the joins at a node and between neighbouring nodes in the HD, a more general approach that considers constraints and preferences is called for.

In this work, we propose a novel framework based on the notion of *candidate tree decompositions* (CTDs) from [16]. A CTD is a tree decomposition (TD) whose bags have to be chosen from a specified set of vertex sets (= the *candidate bags*). One can then reduce the problem of finding a particular decomposition (in particular, generalised or fractional hypertree decomposition) for given CQ q to the problem of finding a CTD for an appropriately defined set of candidate bags. In [16], this idea was used (by specifying appropriate sets of candidate bags) to identify tractable fragments of deciding, for given CQ q and fixed k, if $ghw(H) \leq k$ or $fhw(H) \leq k$ holds. However, as was observed in [16], it is unclear how the idea of CTDs could be applied to hw-computation, where parent/child relationships between nodes in a decomposition are critical.

We will overcome this problem by introducing a relaxed notion of HDs and hw, which we will refer to as *soft HDs* and *soft hw* (shw). The crux of this approach will be a relaxation of the so-called "special condition" of HDs (formal definitions of all concepts mentioned in the introduction will be

given in Section 2): it will be restrictive enough to guarantee a polynomial upper bound on the number of candidate bags and relaxed enough to make the approach oblivious of any parent/child relationships of nodes in the decomposition. We thus create a novel framework based on CTDs, that paves the way for several improvements over the original approach based on HDs and hw, while retaining the crucial tractability of deciding, for fixed k, if the width is $\leq k$ and, if so, computing a decomposition of the desired width.

First of all, shw constitutes an improvement in terms of the width, i.e., it is never greater than hw and there exist CQs q with strictly smaller shw then hw. The definition of soft HDs and soft hw lends itself to a natural generalisation by specifying rules for extending a given set of candidate bags in a controlled manner. We thus define a whole hierarchy of width measures $(shw^i)_{i\geq 0}$ with $shw^0 = shw$. Moreover, will prove that $shw^\infty = ghw$ holds. In other words, we extend the above mentioned hierarchy of width measures to the following extended hierarchy:

$$fhw(q) \leq ghw(q) = shw^\infty(q) \leq shw^i(q) \leq shw^0(q) = shw(q) \leq hw(q),$$

which holds for every CQ q. Basing our new width notion shw on CTDs gives us a lot of computational flexibility. In particular, in [16], a straightforward algorithm was presented for computing a CTD for a given set of candidate bags. We will show how various types of constraints (such as disallowing Cartesian products) and preferences (which includes the minimisation of costs functions in [28] as an important special case) can be incorporated into the computation of CTDs without destroying the polynomial time complexity. We emphasise that even though our focus in this work is on soft HDs and shw, the incorporation of constraints and preferences into the CTD computation equally applies to any other use of CTDs – such as the above mentioned computation of generalised and fractional hypertree decomposition in certain settings proposed in [16]. We have also carried out first experimental results with a prototype implementation of our approach. In short, basing query evaluation on soft HDs and incorporating constraints and preferences into their construction may indeed lead to significant performance gains. Moreover the computation of soft HDs in a bottom-up fashion via CTDs instead of the top-down construction [12, 20] or parallel computation of HDs [14] is in the order of milliseconds and does not create a new bottleneck.

Contributions. In summary, our main contributions are as follows:

- We introduce a new type of decompositions plus associated width measure for CQs – namely soft hypertree decompositions and soft hypertree width. We show that this new width measure shw retains the tractability of hw, it can never get greater than hw but, in some cases, it may get smaller. Moreover, we generalise shw to a whole hierarchy $(shw^i)_{i\geq 0}$ of width measures that interpolates the space between hw and ghw.
- The introduction of our new width measures is made possible by dropping the "special condition" of HDs. This allows us the use of candidate TDs, which brings a lot of computational flexibility. We make use of this added flexibility by incorporating constraints and preferences in a principled manner into the computation of decompositions. These ideas of extending the computation of decompositions is not restricted to shw but is equally beneficial to ghw and fhw computation.
- We have implemented our framework. Preliminary experiments with our prototype implementation have produced promising results. In particular, the use of soft HDs and the incorporation of constraints and preferences into the computation of such decompositions may indeed lead to a significant performance gain in query evaluation.

Structure. The rest of this paper is structured as follows: After recalling some basic definitions and results on hypergraphs in Section 2 we revisit the notion of candidate tree decompositions from [16] in Section 3. In Section 4, we introduce the new notion of soft hypertree width shw,

which is then iterated to shw^i in Section 5. The integration of preferences and constraints into soft hypertree decompositions is discussed in Section 6 and first experimental results with these decompositions are reported in Section 7. Further details are provided in the appendix and in the full version of this paper [24].

2 Preliminaries

Conjunctive queries (CQs) are first-order formulas whose only connectives are existential quantification and conjunction, i.e., $\forall, \vee$, and $\neg$ are disallowed. Likewise, Constraint Satisfaction Problems (CSPs) are given in this form. Hypergraphs have proved to be a useful abstraction of CQs and CSPs. Recall that a *hypergraph* H is a pair $(V(H), E(H))$ where $V(H)$ is called the set of vertices and $E(H) \subseteq 2^{V(H)}$ is the set of *(hyper)edges*. Then the hypergraph $H(\phi)$ corresponding to a CQ or CSP given by a logical formula ϕ has as vertices the variables occurring in ϕ. Moreover, every collection of variables jointly occurring in an atom of ϕ forms an edge in $H(\phi)$. From now on, we will mainly talk about hypergraphs with the understanding that all results equally apply to CQs and CSPs.

We write $I(v)$ for the set $\{e \in E(H) \mid v \in e\}$ of edges incident to vertex v. W.l.o.g., we assume that hypergraphs have no isolated vertices, i.e., every $v \in V(H)$ is in some edge. A set of vertices $U \subseteq V(H)$ induces the *induced subhypergraph* $H[U]$ with vertices U and edges $\{e \cap U \mid e \in E(H)\} \setminus \{\varnothing\}$.

Let H be a hypergraph and let u, v be two distinct vertices in H. A path from u to v is a sequence of vertices $w_1, \ldots, w_m$ in H, such that $w_0 = u$, $w_m = v$, and, for every $j \in \{0, \ldots, m-1\}$, the vertices w_j, w_{j+1} jointly occur in some edge of H. Now let $S \subseteq V(H)$. We say that two vertices u, v are $[S]$-*connected* if they are not in S and there is a path from u to v without any vertices of S. Two edges e, f are $[S]$-connected, if there are $[S]$-connected vertices $u \in e, v \in f$. An $[S]$-*component* is a maximal set of pairwise $[S]$-connected edges. For a set of edges $\lambda \subseteq E(H)$, we will simply speak of $[\lambda]$-components in place of $[\bigcup \lambda]$-components. We note that the term "$[S]$-connected" is slightly misleading, since – contrary to what one might expect – it means that the vertices of S are actually *disallowed* in a path that connects two vertices or edges. However, this terminology has been commonly used in the literature on hypertree decompositions since their introduction in [17]. To avoid confusion, we have decided to stick to it also here.

A *tree decomposition* (TD) of hypergraph H is a tuple (T, B) where T is a tree and $B : V(T) \to 2^{V(H)}$ such that the following hold:

(1) for every $e \in E(H)$, there is a node u in T such that $e \subseteq B(u)$,
(2) and for every $v \in V(H)$, $\{u \in V(T) \mid v \in B(u)\}$ induces a non-empty subtree of T.

The vertex sets $B(u)$ are referred to as *bags* of the TD. The latter condition is referred to as the *connectedness condition*. We sometimes assume that a TD is rooted. In this case, we write T_u for $u \in V(T)$ to refer to the subtree rooted at u. For a subtree T' of T, we write $B(T')$ for $\bigcup_{u \in V(T')} B(u)$.

A *generalised hypertree decomposition* (GHD) of a hypergraph H is a triple (T, λ, B), such that (T, B) is a tree decomposition and $\lambda : V(T) \to 2^{E(H)}$ satisfies $B(u) \subseteq \bigcup \lambda(u)$ for every $u \in V(T)$. That is, every $\lambda(u)$ is an *edge cover* of $B(u)$. A *hypertree decomposition* (HD) of a hypergraph H is a GHD (T, λ, B), where the tree T is rooted and the so-called *special condition* holds, i.e., for every $u \in V(T)$, we have $B(T_u) \cap \bigcup \lambda(u) \subseteq B(u)$. Actually, in the literature on (generalised) hypertree decompositions, it is more common to use χ instead of B. Moreover, rather than explicitly stating B, χ, and λ as functions, they are usually considered as *labels* – writing B_u, λ_u, and χ_u rather than $B(u)$, $\chi(u)$, and $\lambda(u)$, and referring to them as B-, χ-, and λ-label of node u.

All these decompositions give rise to a notion of *width*: The width of a TD (T, B) is defined as $\max(\{|B(u)| : u \text{ is a node in } T\} - 1$, and the width of a (G)HD (T, λ, χ) is defined as $\max(\{|\lambda(u)| : u$ is a node in $T\}$. Then the *treewidth* $tw(H)$, the *hypertree-width* $hw(H)$, and the *generalised hypertree-width* $ghw(H)$ of a hypergraph H are defined as the minimum width over all TDs, HDs, and GHDs,

respectively, of H. The problems of finding the $tw(H)$, $hw(H)$, or $ghw(H)$ for given hypergraph H (strictly speaking, the decision problem of deciding whether any of these width notions is $\leq k$ for given k) are NP-complete [3, 21]. Deciding $tw \leq k$ or $hw(H) \leq k$ becomes tractable, if k is a fixed constant. In contrast, deciding $ghw(H) \leq k$ remains NP-complete even if we fix $k = 2$ [16, 18].

We note that, in the literature on tree decompositions, it is common practice to define TDs for graphs (rather than hypergraphs) in the first place. The TDs of a hypergraph $H = (V(H), E(H))$ are then defined as TDs of the so-called *Gaifman graph* of H, i.e., the graph $G = (V(G), E(G))$, with $V(G) = V(H)$, such that $E(G)$ contains an edge between two vertices u, v if and only if u, v jointly occur in an edge in $E(H)$. Clearly, every edge of a hypergraph H gives rise to a clique in the Gaifman graph G. Moreover, it is easy to verify that every clique of a graph has to be contained in some bag of a TD. Hence, for TDs, the Gaifman graph contains all the relevant information of the hypergraph. In contrast, for HDs and GHDs, we additionally have to keep track of the edges of the hypergraph, since only these are allowed to be used in the λ-labels. We have, therefore, preferred to define also TDs directly for hypergraphs (rather than via the Gaifman graph).

3 Revisiting Candidate Tree Decompositions

In this section, we revisit the problem of constructing tree decompositions by selecting the bags from a given set of candidate bags. This leads us to the notion of candidate tree decompositions (CTDs) and the CandidateTD problem, which we both define next.

Definition 1. *Let H be a hypergraph and let $\mathbf{S} \subseteq 2^{V(H)}$ be a set of vertex sets of H (the so-called* "candidate bags"*). A* candidate tree decomposition (CTD) *of $\mathbf{S}$ is a tree decomposition (T, B) of H, such that, for every node u of T, $B(u) \in \mathbf{S}$.*

In the CandidateTD problem, we are given a hypergraph H and a set $\mathbf{S} \subseteq 2^{V(H)}$ (= the set of candidate bags), and we have to decide whether there exists a tree decomposition (T, B) of H, such that, for every node $u \in T$, we have $B(u) \in \mathbf{S}$.

The idea of generating TDs from sets of candidate bags was heavily used in a series of papers by Bouchitté and Todinca [6–10] in a quest to identify a large class of graphs for which the treewidth (and also the so-called minimum fill-in, for details see any of the cited papers) can be computed in polynomial time. More specifically, this class of graphs G is characterised by having a polynomial number of minimal separators (i.e., subsets S of $V(G)$, such that two vertices u, v of G are in different connected components of the induced subgraph $G[V(G) \setminus S]$ but they would be in the same connected component of $G[V(G) \setminus S']$ for every proper subset $S' \subset S$). The key to this tractability result was to precisely identify and compute in polynomial time the set of candidate bags via the minimal separators and the so-called potential maximal cliques of a given graph. Ravid et al. [27] applied these ideas to efficiently enumerate TDs under a monotonic cost measure, such as treewidth.

It was shown in [16], that the problem of deciding whether a hypergraph has width $\leq k$ for various notions of decompositions (in particular, the generalised hypertree width *ghw* and the so-called fractional hypertree width *fhw*) can be reduced to the CandidateTD problem by an appropriate choice of the set S of candidate bags. However, it was also shown in [16] that some care is required as far as the form of the sought after TDs is concerned. In particular, it was shown in [16], that, in the unrestricted form given in Definition 1, the CandidateTD problem is NP-complete. In order to ensure tractability, the following restriction on TDs was formally defined in [16]:

Definition 2. *A rooted tree decomposition (T, B) is in* component normal form (CompNF)*, if for each node $u \in V(T)$, and for each child c of u, there is exactly one $[B(u)]$-component C_c such that $B(T_c) = \bigcup C_c \cup (B(u) \cap B(c))$.*

Algorithm 1: CompNF Candidate Tree Decomposition of $\mathbf{S}$

input: Hypergraph H and a set $\mathbf{S} \subseteq 2^{V(H)}$.
output: "Accept", if there is a CompNF CTD for $\mathbf{S}$
"Reject", otherwise.

```
1  blocks = all blocks headed by any S ∈ S ∪ {∅}
2  foreach (S, C) ∈ blocks do
3  |  if C = ∅ then basis(S, C) ← ∅
4  |  else basis(S, C) ← ⊥     /* a block is satisfied iff its basis is not ⊥ */
5  repeat
6  |  foreach (S, C) ∈ blocks that are not satisfied do
7  |  |  foreach X ∈ S ∖ {S} do
8  |  |  |  if X is a basis of (S, C) then
9  |  |  |  |  basis(S, C) ← X
10 until no blocks changed
11 if basis(∅, V(H)) ≠ ⊥ then
12 |  return Accept
13 return Reject
```

Alternatively, we could define CompNF by making use of the fact that the TDs of a hypergraph H are exactly the TDs of its Gaifmann graph $G(H)$ (see Section 2). The CompNF condition given in Definition 2 can then be rephrased as requiring that, for a node u and child node c in a rooted TD (T, B), that the vertices in $B(T_c)$ must not be separated by the vertex set $B(u) \cap B(c)$.

It was shown in [16] that the CandidateTD problem becomes tractable if we ask for the existence of a CTD *in CompNF*. We note that also the algorithms presented in the works of Bouchitté and Todinca (see, in particular, [8, 9]) as well as in [27] implicitly only consider TDs in CompNF. From now on, we consider the CandidateTD problem only in this restricted form. By slight abuse of notation, we will simply refer to it as the CandidateTD problem, with the understanding that we are only looking for CTDs in CompNF. Gottlob et al. [16] presented a poly-time algorithm, which we recall in a slightly adapted form in Algorithm 1, for deciding the CandidateTD problem.

To discuss Algorithm 1 in detail, we first have to define the terminology used in the algorithm, namely the notions of a "block", a "basis", and what it means that a block is "satisfied". We call a pair (S, C) of *disjoint* subsets of $V(H)$ a *block* if C is a maximal set of $[S]$-connected vertices of H or $C = \emptyset$. We say that (S, C) is *headed* by S. For two blocks (S, C) and (X, Y) define $(X, Y) \leq (S, C)$ if $X \cup Y \subseteq S \cup C$ and $Y \subseteq C$. Note that the notion of a *block* was already introduced in the works of Bouchitté and Todinca [6–10] and later used by Ravid et al. [27]. In this paper (following [16]), blocks are defined slightly more generally in the sense that Bouchitté and Todinca only considered blocks (S, C) where S is a minimal separator of the given graph (rather than an arbitrary, possibly even empty, subset of the vertices) and C is not allowed to be empty.

A block (S, C) is *satisfied* if a CompNF TD of $H[S \cup C]$ exists with root bag S. If $C = \emptyset$, satisfaction is trivial. Finally, for a block (S, C) and $X \subseteq V(H)$ with $X \neq S$, let $(X, Y_1), \ldots, (X, Y_\ell)$ be all the blocks headed by X with $(X, Y_i) \leq (S, C)$. Then we say that X *is a basis of* (S, C) if the following conditions hold:

(1) $C \subseteq X \cup \bigcup_{i=1}^{\ell} Y_i$.
(2) For each $e \in E(H)$ such that $e \cap C \neq \emptyset$, $e \subseteq X \cup \bigcup_{i=1}^{\ell} Y_i$.

(3) For each $i \in [\ell]$, the block (X, Y_i) is satisfied.

The concepts of a *block* and of a *basis of a block* are related to a CTD (in CompNF) in the following way: Recall from Definition 2 that if u, c are parent-child nodes in a TD, then there is exactly one $[B(u)]$-component C_c such that $B(T_c) = \bigcup C_c \cup (B(u) \cap B(c))$. Now consider the subtree T' of T that consists of the node u plus the entire subtree T_c. Moreover, consider the block (S, C) with $S = B[u]$ and $C = \bigcup C_c \setminus B(u)$. Then this block is indeed *satisfied* by taking as CTD of $H[S \cup C]$ the subtree T' of T (where the bag of each node in T' is exactly the bag of the corresponding node in T). The goal of Algorithm 1 is to satisfy progressively larger blocks, increasing $|S \cup C|$. If $(\varnothing, V(H))$ is satisfied, a CTD for H exists, as checked in Line 11.

The relationship between the notions of *block*, *basis*, and *satisfaction* of a block is summarised by the following property that was proved in [16]: *Let (S, C) be a block and let X be a basis of (S, C), then (S, C) is satisfied.* The proof idea of this property is as follows: let $(X, Y_1), \ldots, (X, Y_\ell)$ be all the blocks headed by X with $(X, Y_i) \leq (S, C)$. By the third condition of a basis, the block (X, Y_i) is satisfied for each $i \in [\ell]$. Hence, there exists a CTD (T_i, B_i) of each subhypergraph $H[X \cup Y_i]$, such that the bag of the root is X for all theses CTDs. We can merge all these root nodes into a single node to form a single TD (T', B') with the T_i's as subtrees. A CTD for $H[S \cup C]$ is then obtained by taking as root a node with bag S and appending the root of T' (with bag X) plus the entire tree T'. The aim of the repeat-loop in Algorithm 1 (Lines 5-10) is precisely to check if we can mark yet another block (S, C) as satisfied by identifying a basis for it. Initially, in the foreach-loop on Lines 2-4, only the blocks with empty C-part are assigned the trivial basis $\varnothing$ and thus marked as satisfied. Finally, on Lines 11-13, the algorithm returns "Accept" (i.e., a CTD of H exists) if the block $(\varnothing, V(H))$ is marked as satisfied via a non-empty basis. Otherwise, it returns "Reject". The bottom-up approach in Algorithm 1, constructing a CTD by combining subtrees, also appears similarly in prior work [9, 27].

For the polynomial-time complexity of Algorithm 1, the crucial observation is that the number b of blocks (S, C) is bounded by $|\mathbf{S}| * |V(H)|$, i.e., for each $S \in \mathbf{S}$, there cannot be more components C than vertices in H. As a coarse-grained upper bound on the complexity of Algorithm 1, we thus get $b^4 * ||H||$, i.e., each of the 3 levels of nested loops has at most b iterations and the cost of checking the basis-property on Line 8 can be bounded by b times the size of (some representation of) H.

4 Soft Hypertree Width

So far, it is not known whether there is a set of candidate bags $\mathbf{S}_{H,k}$ for a hypergraph H such that there is a CTD for $\mathbf{S}_{H,k}$ if and only if $hw(H) \leq k$. However, in [14], it was shown that there always exists a HD of minimal width such that all bags of nodes c with parents p are of the form $B_c = (\bigcup \lambda_c) \cap \left(\bigcup C_p\right)$ where C_p is a $[\lambda_p]$-component of H. In principle, this gives us a concrete way to enumerate a sufficient list of candidate bags. The number of all such bags is clearly polynomial in H: there are at most $|E(H)|^{k+1}$ sets of at most k edges, and each of them cannot split H into more than $|E(H)|$ components. The only point that is unclear when enumerating such a list of candidate bags, is how to decide beforehand whether two sets of edges λ_c, λ_p are in a parent/child relationship (and if so, which role they take). However, we observe that the parent/child roles are irrelevant for the polynomial bound on the number of candidate bags. Hence, we may drop this restriction and instead simply consider all such combinations induced by any two sets of at most k edges. Concretely, this leads us to the following definitions.

Definition 3 (The set $\text{Soft}_{H,k}$). *For hypergraph H, we define $\text{Soft}_{H,k}$ as the set that contains all sets of the form*

$$B = (\bigcup \lambda_1) \cap (\bigcup C) \tag{1}$$

where C is a $[\lambda_2]$-component of H and λ_1, λ_2 are sets of at most k edges of H.

Definition 4 (Soft Hypertree Width). *A* soft hypertree decomposition *of width k for hypergraph H is a candidate tree decomposition for* $\text{Soft}_{H,k}$. *The* soft hypertree width (*shw*) *of hypergraph H is the minimal k for which there exists a soft hypertree decomposition of H.*

This measure naturally generalises the notion of hypertree width in a way that remains tractable to check (for fixed k), but removes the need for the special condition.

Theorem 1. *Let $k \geq 1$. Deciding, for given hypergraph H, whether $shw(H) \leq k$ holds, is feasible in polynomial time in the size of H. The problem even lies in the highly parallelisable class LogCFL.*

Proof Sketch. In [17], it is shown that checking if $hw(H) \leq k$ holds for fixed $k \geq 1$ is in LogCFL. The key part of the proof is the construction of an alternating Turing machine (ATM) that runs in Logspace and Ptime. The ATM constructs an HD in a top-down fashion. In the existential steps, one guesses a λ-label of the next node in the HD. Given the label λ_c of the current node and the label λ_p of its parent node, one can compute the set of $[\lambda_c]$-components that lie inside a $[\lambda_p]$-component. In the following universal step, the ATM has to check recursively if all these $[\lambda_c]$-components admit an HD of width $\leq k$.

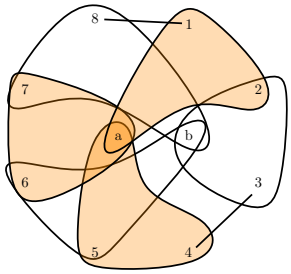
This ATM can be easily adapted to an ATM for checking if $shw(H) \leq k$ holds. The main difference is, that rather than guessing the label λ_c of the current node (i.e., a collection of at most k edges), we now simply guess an element from $\text{Soft}_{H,k}$ as the bag B_c of the current node. The universal step is then again a recursive check for all components of B_c that lie inside a $[B_p]$-component, if they admit a candidate TD of width $\leq k$. The crucial observation is that we only need Logspace to represent the bags $B \in shw(H)$, because every such bag is uniquely determined by the label λ_c and a $[\lambda_p]$-component. Clearly, λ_c can be represented by up to k (pointers to) edges in $E(H)$, and a $[\lambda_p]$-component can be represented by up to $k+1$ (pointers to) edges in $E(H)$, i.e., up to k edges in λ_p plus 1 edge from the $[\lambda_p]$-component. Clearly, the latter uniquely identifies a component, since no edge can be contained in 2 components. □

The main argument in the proof of Theorem 1 was that the ATM for checking $hw(H) \leq k$ can be easily adapted to an algorithm for checking $shw(H) \leq k$. Actually, the straightforward adaptation of existing *hw*-algorithms to *shw*-algorithms is by no means restricted to the rather theoretical ATM of [17]. The parallel algorithm log-k-decomp from [14] performs significant additional effort to control orientation of subtrees in order to guarantee the special condition. This is unnecessary for *shw*-computation, where the orientation of subtrees is irrelevant. Instead, we may follow the philosophy of the much simpler BalancedGo algorithm in [19] for *ghw*-computation. By a standard argument (see e.g., [14, Lemma 3.14]), a CTD for Soft always contains a *balanced separator*. Hence, one can simply adapt BalancedGo algorithm to only consider separators in Soft to obtain another algorithm for checking *shw* that is suitable for parallelisation. In Section 6, we will extend the CTD-framework by constraints and preferences. It will turn out that we can thus also capture the opt-k-decomp approach from [28], that integrates a cost function into the computation of HDs. To conclude, the key techniques for modern decomposition algorithms are also applicable to *shw*-computation. Additionally, just as with other popular hypergraph width measures, it is possible to relate *shw* to a variant of the Robber and Marshals game (refer to the full version [24]).

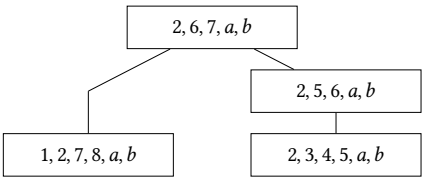
The relationship of $shw(H)$ with $hw(H)$ and *ghw* is characterised by the following result:

Theorem 2. *For every hypergraph H, the relationship $ghw(H) \leq shw(H) \leq hw(H)$ holds. Moreover, there exist hypergraphs H with $shw(H) < hw(H)$.*

Proof. By [14], if $hw(H) = k$, then there exists a hypertree decomposition of width k such that every bag is of the form from Equation (1). That is, for every HD (T, λ, χ), we immediately get a candidate TD (T, χ) for $\text{Soft}_{H,k}$. Hence, $shw(H) \leq hw(H)$. Furthermore, every bag in $\text{Soft}_{H,k}$ is



(a) Hypergraph H_2 with $ghw(H_2) = shw(H_2) = 2$ and $hw(H_2) = 3$. Some edges are coloured for visual clarity.



(b) A soft hypertree decomposition of H_2 with width 2.

Fig. 1. (a) A hypergraph H_2 and (b) its soft hypertree decomposition.

subset of a union of k edges, hence $\rho(B) \leq k$ for any B from Equation (1). Thus also $ghw(H) \leq shw(H)$. In the example below, we will present a hypergraph H with $shw(H) < hw(H)$. □

We leave as an interesting open question for future work if the gap between shw and hw can become arbitrarily large. However, in the next section, we will introduce a natural iteration of shw to get a whole hierarchy $(shw^i)_{i\geq 0}$ of width notions with $shw^0 = shw$ and $shw^\infty = ghw$. We are able to show that the gap between shw^i and hw can become arbitrarily big for every $i \geq 1$, however it remains open whether this already holds for the case $i = 0$, i.e., shw itself.

Example 1. *Let us revisit the hypergraph H_2 from [2], which was presented there to show that ghw can be strictly smaller than hw. The hypergraph is shown in Figure 1a. It consists of the edges $\{1, 8\}, \{3, 4\}, \{1, 2, a\}, \{4, 5, a\}, \{6, 7, a\}, \{2, 3, b\}, \{5, 6, b\}, \{7, 8, b\}$ and no isolated vertices. It is shown in [2] that $ghw(H_2) = 2$ and $hw(H_2) = 3$. We now show that also $shw(H_2) = 2$ holds.*

A candidate tree decomposition for $\mathrm{Soft}_{H_2,2}$ is shown in Figure 1b. Let us check that $\mathrm{Soft}_{H_2,2}$ contains all the bags in the decomposition. The bags $\{1, 2, 7, 8, a, b\}$ and $\{2, 3, 4, 5, a, b\}$ are the union of 2 edges and thus clearly in the set. The bag $\{2, 6, 7, a, b\}$ is induced by $\lambda_2 = \{\{3, 4\}, \{2, 3, b\}\}$. There is only one $[\lambda_2]$-component C, which contains all edges of $E(H) \setminus \lambda_2$. Hence, $\bigcup C$ contains all vertices in $V(H) \setminus \{3\}$. We thus get the bag $\{2, 6, 7, a, b\}$ as $\left(\bigcup \lambda_1\right) \cap \left(\bigcup C\right)$ with $\lambda_1 = \{\{2, 3, b\}, \{6, 7, a\}\}$. The remaining bag $\{2, 5, 6, a, b\}$ is obtained similarly from $\lambda_2 = \{\{1, 8\}, \{1, 2, a\}\}$ which yields a single component C with $\bigcup C = H(H) \setminus \{1\}$. We thus get the bag $\{2, 5, 6, a, b\}$ as $\left(\bigcup \lambda_1\right) \cap \left(\bigcup C\right)$ with $\lambda_1 = \{\{1, 2, a\}, \{5, 6, b\}\}$.

We have chosen the hypergraph H_2 above, since it has been the standard (and only) example in the literature of a hypergraph with $ghw = 2$ and $hw = 3$. It illustrates that the relaxation to shw introduces useful new candidate bags. A more elaborate hypergraph (from [1]) with $ghw(H) = shw(H) = 3$ and $hw(H) = 4$ is provided in Appendix A.1.

5 Even Softer Hypertree Width

In Definition 3, we have introduced a formalism to define a collection $\mathrm{Soft}_{H,k}$ of bags that one could use in TDs towards the definition of a soft hypertree width of hypergraph H. We now show how this process could be iterated in a natural way to get "softer and softer" notions of hypertree width. The key idea is to extend, for a given hypergraph H, the set $E(H)$ of edges by subedges that one could possibly use in a λ-label of a soft HD to produce the desired χ-labels as $\chi = \bigcup \lambda$. It will turn out that we thus get a kind of interpolation between shw and ghw.

We first introduce the following useful notation:

Definition 5. *Let A, B be two sets of sets. We write $A \boxtimes B$ as shorthand for the set of all pairwise intersections of sets from A with sets from B, i.e.: $A \boxtimes B := \{ a \cap b \mid a \in A, b \in B \}$*

We now define an iterative process of obtaining sets $E^{(i)}$ of subedges of the edges in $E(H)$ and the corresponding collection $\text{Soft}^i_{H,k}$ of bags in TDs that one can construct with these subedges.

Definition 6. *Consider a hypergraph H. For $i \geq 0$, we define sets $E^{(i)}$ and $\text{Soft}^i_{H,k}$ as follows: For $i = 0$, we set $E^{(0)} = E(H)$ and $\text{Soft}^0_{H,k} = \text{Soft}_{H,k}$ as defined in Definition 3. For $i > 0$, we set $E^{(i)} = E^{(i-1)} \boxtimes \text{Soft}^{i-1}_{H,k}$ and we define $\text{Soft}^i_{H,k}$ as the set that contains all sets of the form*

$$B = (\bigcup \lambda_1) \cap (\bigcup C)$$

where C is a $[\lambda_2]$-component of H, λ_1 is a set of at most k elements of $E^{(i)}$, and λ_2 is a set of at most k elements of $E(H)$.

We can then extend Definition 4 as follows: A soft hypertree decomposition of order *i of width k of H is a candidate tree decomposition for $\text{Soft}^i_{H,k}$. Also, the* soft hypertree width of order *i of H (denoted $shw^i(H)$) is the minimal k, s.t. there is a soft hypertree decomposition of order i of width k of H.*

Intuitively, $E^{(i)}$ represents all the "interesting" subedges of the edges in $E(H)$ relative to the possible bags in $\text{Soft}^{i-1}_{H,k}$, i.e., every element $B \in \text{Soft}^{i-1}_{H,k}$ can be obtained as $B = e'_1 \cup \cdots \cup e'_\ell$ with $\ell \leq k$ and $e'_j \in E^{(i)}$ for every j. In this way, we can iteratively refine the set of considered bags in a targeted way to arrive at an interpolation of *ghw*. This idea will be made precise in Theorem 7 below. First, we establish certain monotonicity properties of $E^{(i)}$ and $\text{Soft}^i_{H,k}$.

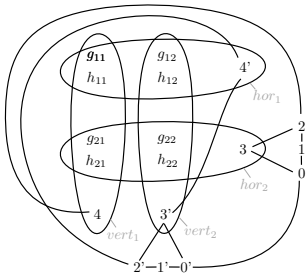
Lemma 3. *Let H be a hypergraph and $i \geq 0$. Moreover, let $E^{(i)}$ and $\text{Soft}^i_{H,k}$ be defined according to Definition 6. Then the following subset relationships hold:*

$$E^{(i)} \subseteq E^{(i+1)}, \quad E^{(i)} \subseteq \text{Soft}^i_{H,k}, \quad \text{and} \quad \text{Soft}^i_{H,k} \subseteq \text{Soft}^{i+1}_{H,k}$$

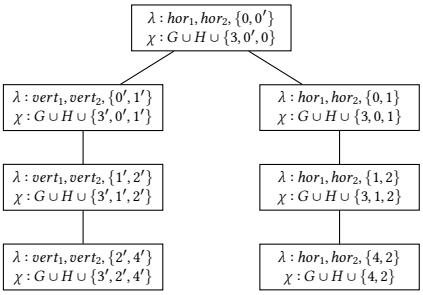
Example 2. *To illustrate the more complex dynamics of iterated shw, we present a modification[1] of a particular hypergraph by Adler [1]. Our modified hypergraph H'_3 is shown in Figure 2a. It has vertices $G = \{g_{11}, g_{12}, g_{21}, g_{22}\}$, $H = \{h_{11}, h_{12}, h_{21}, h_{22}\}$, and $V = \{0, 1, 2, 3, 4, 0', 1', 2', 3', 4'\}$. Importantly, there are edges $\{w, v\}$ for every $w \in G \cup H$ and $v \in V$. Intuitively this means that, to achieve low width, a bag in a decomposition always has to cover $G \cup H$ since, otherwise, the bag would not split the hypergraph into more than one component. This hypergraph satisfies $ghw(H'_3) = shw^1(H'_3) = 3$ and $shw(H'_3) = hw(H'_3) = 4$. A GHD of width 3 for H'_3 is shown in Figure 2b. We observe that every λ-label contains either the two "horizontal" edges hor_1 and hor_2, or the two "vertical" edges ver_1 and ver_2 to cover $G \cup H$. Crucially, the root bag $\chi_r = G \cup H \cup \{3, 0', 0\}$ is not in $\text{Soft}^0_{H'_3,3}$ (neither are the bags of the two closest descendants on the right-hand side of the root). We refer the reader to [1] for further discussion of why there cannot be a GHD of width 3 without the discussed bags. The critical observation for why these bags are not in $\text{Soft}^0_{H'_3,3}$ is that there is no way to separate $4'$ from the rest of the hypergraph with three edges. Hence, in Equation (1), for $k = 3$, any λ_p would induce only a single component that contains $4'$, leaving no way to retrieve the required bags from Figure 2b.*

The bags in question are however all in $\text{Soft}^1_{H'_3,3}$. It suffices to show that $hor'_1 := hor_1 \setminus \{4'\}$ is in $E^{(1)}$, as then all bags on the right-hand side of Figure 2b are unions of at most three edges in $E^{(1)}$. Figure 2c illustrates why hor'_1 is indeed in $E^{(1)}$ and how it intuitively connects to a special condition violation (SCV) in the respective GHD. Observe that hor_1 in the λ-label of the root contains $4'$ but the

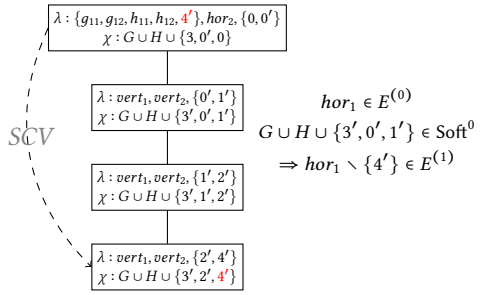
[1]The original hypergraph from [1] lacks the edge $\{3', 4'\}$ and is discussed in detail in Appendix A.1.



(a) The hypergraph H_3' with $ghw(H_3') = shw^1(H_3') = 3$ and $shw(H_3') = hw(H_3') = 4$. The image omits the additional edges $\{\{w, v\} \mid w \in G \cup H, v \in V\}$.



(b) A generalised hypertree decomposition of H_3' with width 3 and all bags in $\text{Soft}^1_{H_3',3}$. The root bag $G \cup H \cup \{3, 0', 0\}$ is not in $\text{Soft}_{H_3',3}$.



(c) Illustration of how the special condition violation (SCV) at the root bag relates to our Soft hierarchy.

Fig. 2. Illustrations for Example 2

bag at the root does not. Hence, the special condition is violated, because $4'$ *appears in a bag further down in the decomposition. In fact, this leaf node is the first descendant (in this case, actually the only descendant) of the root with* $4'$ *occurring in the bag. On the other hand,* $4'$ *is missing from all the bags on the path between the root and this leaf node. Hence, intersecting any bag in* $\text{Soft}^0_{H_3',3}$ *with any of the bags along the path allows us to eliminate* $4'$*. In particular, if we do so by intersecting* $hor_1 \in E^{(0)}$ *with the bag* $G \cup H \cup \{3', 0', 1'\} \in \text{Soft}^0_{H_3',3}$ *(i.e., the right child of the root) we get* hor_1'*. This proves that* hor_1' *is in* $E^{(1)}$ *and it also illustrates more concretely what we intuitively described when stating that* $E^{(i)}$ *contains all the "interesting" subedges with respect to* $\text{Soft}^{i-1}_{H,k}$*.*

Now that we have seen that $\text{Soft}^{i+1}_{H,k}$ extends $\text{Soft}^i_{H,k}$, a natural next question is, how expensive is it to get from $\text{Soft}^i_{H,k}$ to $\text{Soft}^{i+1}_{H,k}$. In particular, by how much does $\text{Soft}^{i+1}_{H,k}$ increase compared to $\text{Soft}^i_{H,k}$. The following lemma establishes a polynomial bound, if we consider k as fixed.

Lemma 4. *Let* H *be a hypergraph and* $i \geq 0$ *and* $k \geq 1$*. Then we have:*

$$|\text{Soft}^{i+1}_{H,k}| \leq |\text{Soft}^i_{H,k}|^{4k+6}.$$

The following result on the complexity of recognizing low $shw^i(H)$ is an immediate consequence.

Theorem 5. *Let* i *and* k *be fixed positive integers. Then, given a hypergraph* H*, it can be decided in polynomial time, if* $shw^i(H) \leq k$ *holds.*

Proof. It follows from Lemma 4, that we can compute $\text{Soft}^i_{H,k}$ in polynomial time (for fixed i, k). Hence, by the tractability of the CandidateTD problem recalled in Section 3, deciding $shw^i(H) \leq k$ is then also feasible in polynomial time. □

Since Definition 6 defines a width measure for every natural number i, we also wish to analyse the behaviour in the limit as i approaches infinity. To that end we define $\text{Soft}^\infty_{H,k} := \bigcup_{i\in\mathbb{N}} \text{Soft}^i_{H,k}$ and shw^∞ as in Definition 6 for candidate bags $\text{Soft}^\infty_{H,k}$. We show that $shw^i(H)$ in fact converges towards $ghw(H)$. To this end, we first show that, for given hypergraph H, $\left(\text{Soft}^i_{H,k}\right)_{i\geq 0}$ and, therefore, also $\left(shw^i(H)\right)_{i\geq 0}$, converges towards a fixpoint. In a second step, we then show that, for $shw^i(H)$, this fixpoint is actually $ghw(H)$.

Lemma 6. *Let H be a hypergraph, let $n = \max(|V(H)|, |E(H)|)$, and let k be a positive integer. Then there exists an $\alpha \leq 3n$, such that*

$$\text{Soft}^\alpha_{H,k} = \text{Soft}^\infty_{H,k}$$

Theorem 7. *For every hypergraph H, we have $shw^\infty(H) = ghw(H)$*

Proof Sketch. The proof starts from a GHD of H and argues that, after sufficiently many iterations of defining $E^{(i+1)}$ from $E^{(i)}$, all bags χ_u of the GHD are available as candidate bags. To this end, we make use of the following property from [16], Lemma 5.12: Let $e \in \lambda_u$ with $e \nsubseteq \chi_u$ for some node u of the GHD and let u' be a node with $e \subseteq \chi_{u'}$. Moreover, let $u = u_0, \ldots, u_\ell = u'$ be the path from u to u' in the GHD. Then the following property holds: $e \cap \chi_u = e \cap \left(\bigcap_{j=1}^{\ell} \bigcup \lambda_{u_j}\right)$.

The proof of the theorem then comes down to showing that all these subedges of edges $e \in E(H)$ are contained in $E^{(p)}$, where p is the maximal length of paths in the GHD. □

We thus get yet another bound on the number of iterations required for soft-HDs to bring shw^d down to ghw.

Corollary 8. *Let H be a hypergraph with $ghw(H) = k$ and suppose that there exists a width k GHD of H whose depth is bounded by d'. Then $shw^d(H) = ghw(H)$ with $d \leq 2d'$ holds.*

In [1] it was shown that the gap between so-called marshal width (a lower bound on ghw) and hw can become arbitrarily big. By adapting that proof, we can show that also the gap between shw^1 and hw can become arbitrarily big. The details are worked out in Appendix B.2.

Theorem 9. *There exists a family of hypergraphs $(H_n)_{n\geq 3}$ satisfying $shw^1(H_n) + n \leq hw(H_n)$.*

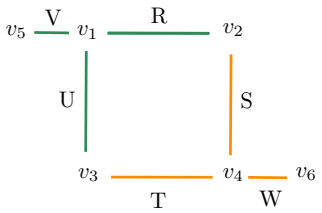
6 Constrained Hypertree Decompositions

Although hypertree decompositions and their generalisations have long been a central tool in identifying the asymptotic worst-case complexity of CQ evaluation, practical applications often demand more than simply a decomposition of low width. While width is the only relevant factor for the typically considered complexity upper bounds, structural properties of the decomposition can critically influence computational efficiency in practice.

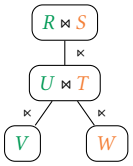
Example 3. *Consider the conjunctive query*

$$q = R(v_1, v_2) \wedge S(v_2, v_4) \wedge T(v_3, v_4) \wedge U(v_1, v_3) \wedge V(v_1, v_5) \wedge W(v_4, v_6).$$

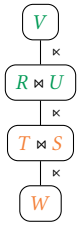
Assume furthermore, that we are in a distributed setting with vertical partitioning. In particular, relations R, U, V are on one node, whereas S, T, W are on another. The query hypergraph, together with the partitioning is illustrated in Figure 3a. This query has multiple simple HDs of minimal width, with little natural reason to prefer one over the other. Some example computations resulting from various HDs of minimal width are illustrated below in (b)-(c).



(a) The query hypergraph H. Edge colours represent the partition of the respective relation.



(b) The computations for answering the query according to a natural HD of H.



(c) The computations for answering the query according to a different natural HD of H.

However, in a distributed setting where we want to minimise the communication between different nodes, we might prefer some HDs over others. In our example Figure 3c might be preferable. Here the bottom half of the decomposition is evaluated purely on the orange node, with only the result of $(T \bowtie S) \ltimes W$ being communicated to the other node. On the other hand, in Figure 3b two of the joins are across partitions, and the semi-join between them might add further communication depending on where it is best to compute the joins. Since HDs as in Figure 3c allow for simpler distributed query answering, some applications might only want HDs that are constrained to such HDs, where partitions are split up over disjoint subtrees of the decomposition. Naturally the full details of a practical scenario would require further details, e.g., whether $T \bowtie S$ is expected to be very large, but this simplified setting already illustrates the fundamental need for decompositions that follow certain constraints.

This issue is one of the simplest cases that highlights the necessity of imposing additional structural constraints on decompositions. By integrating such constraints, we can align decompositions more closely with practical considerations beyond worst-case complexity guarantees. We thus initiate the study of *constrained* decompositions. Specifically, for a constraint $\mathcal{C}$ and width measure *width*, we study $\mathcal{C}$-*width*(H), the least *width* over all decompositions that satisfy the constraint $\mathcal{C}$. In the first place, we thus study constrained *shw*. But we emphasise that the results in this section apply to any notion of decomposition and width that can be computed via CTDs.

To guide the following presentation, we first identify various interesting examples of constraints for a TD (T, B), that we believe might find use in applications:

Connected covers Applications for query evaluation naturally want to avoid Cartesian products in the reduction from a hypertree decomposition to an acyclic query. This motivates constraint ConCov, that holds exactly for those CTDs where every bag B_u has an edge cover λ_u of size $|\lambda_u| \leq k$, such that the edges in λ_u form a connected subhypergraph.

Shallow Cyclicity The *cyclicity depth* of a CTD for H is the least d such that the bag B_u of every node u at depth greater than d can be covered by a single edge of H. The constraint $\mathsf{ShallowCyc}_d$ is satisfied by a CTD, if it has cyclicity depth at most d. Intuitively, this constraint captures having a cyclic "core" to the query with acyclic parts attached to it. Such structure with low cyclicity depth can be naturally leveraged for efficient query answering by reducing the relations in the high-width nodes through semi-joins with the attached acyclic parts.

Partition Clustering As discussed in Example 3, in distributed scenarios, where relations are partitioned across the network, query evaluation using a decomposition could benefit substantially from being able to evaluate entire subtrees of the CTD at a single partition. We can enforce this through a constraint of the following form: Let $\rho : E(H) \to \mathit{Partitions}$ label each edge of the hypergraph with a partition. The constraint PartClust holds in a CTD (T, B), if there exists a function $f : V(T) \to \mathit{Partitions}$ such, that for every node u in T, we have:

(1) B_u has a candidate cover using edges with label $f(u)$.

Algorithm 2: $(\mathcal{C}, \leq)$-Candidate Tree Decomposition of $\mathbf{S}$

```
    ⋮
5   repeat
6     foreach (B, C) ∈ blocks do
7       foreach X ∈ S ∖ {B} do
8         if X is a basis of (B, C) then
9           D_new ← Decomp(B, C, X)
10          if D_new ⊨ C and (basis(B, C) = ⊥ or D_new < Decomp(B, C, basis(B, C))) then
              basis(B, C) ← X
11  until no blocks changed
    ⋮
```

(2) For every $p \in \mathit{Partitions}$, the set of nodes u with $f(u) = p$ induces a subtree of T that is disjoint from the respective induced subtrees of all other partitions.

Introducing such constraints can, of course, increase the width. For example, for the 5-cycle C_5, it is easy to verify that ConCov-$ghw(C_5)$ = ConCov-$shw(C_5)$ = ConCov-$hw(C_5)$ = 3 even though $hw(C_5) = 2$. However, in practice, adherence to such constraints can still be beneficial. In the C_5 example, the decomposition of width 2 forces a Cartesian product in the evaluation and is likely to be infeasible even on moderately sized data. Yet using a ConCov decomposition of width 3 might even outperform a typical query plan of two-way joins executed by a standard relational DBMS.

6.1 Tractable Constrained Decompositions

Here, we are interested in the question of when it is tractable to find decompositions satisfying certain constraints. The bottom-up process of Algorithm 1 iteratively builds tree-decompositions for some induced subhypergraph of H (recall that a satisfied block (B, C) corresponds to a TD for $H[B \cup C]$). We will refer to such tree decompositions as *partial tree decompositions* of H.

A *subtree constraint* $\mathcal{C}$ (or, simply, a constraint) is a Boolean property of partial tree decompositions of a given hypergraph H. We write $\mathcal{T} \models \mathcal{C}$ to say that the constraint holds true on the partial tree decomposition $\mathcal{T}$. A tree decomposition (T, B) of H satisfies $\mathcal{C}$ if $T_u \models \mathcal{C}$ for every node the partial tree decomposition induced by T_u.

However, deciding the existence of a tree decomposition satisfying $\mathcal{C}$ might require the prohibitively expensive enumeration of all possible decompositions for a given set of candidate bags. To establish tractability for a large number of constraints, we additionally consider *total quasiorderings of partial tree decompositions (toptds)*[2] $\leq$. A tree decomposition (T, B) is *globally minimal* w.r.t. $\leq$, if for every node u, there is no partial tree decomposition (T'_u, B') of $H[B(T_u)]$ with $(T'_u, B') < (T_u, B)$. We note that our notion of minimality is closely related to the notion of split-monotonicity introduced by Ravid et al. [27] to ensure that the cost of a TD cannot increase if a subtree T' of this TD is replaced by a subtree T'' of lower cost. Our goal here is to reduce the task of finding decompositions that satisfy certain constraints to the task of finding globally minimal decompositions for an appropriate toptd. To formalise when this is possible, we introduce the following property.

Definition 7. *We say that a toptd $\leq$ is* preference complete *for a subtree constraint $\mathcal{C}$ if the following holds. For every hypergraph H, if there exists a CTD of H for which $\mathcal{C}$ holds, then $\mathcal{C}$ holds for all globally minimal (w.r.t. $\leq$) CTDs of H.*

Example 4. *Consider the constraint* ShallowCyc$_d$ *described above. We observe that the property is in a sense monotone over subtrees of the decomposition, i.e., the shallow cyclicity at node u cannot be*

[2]Recall that a total quasiordering of X is a reflexive, transitive, and total relation on X^2.

lower than the maximum shallow cyclicity of the partial tree decompositions rooted at the children of u. Hence, we obtain a decomposition with the least shallow cyclicity if we cover the components below with their respective shallowest partial decompositions.

If we thus consider the toptd $\leq_{\mathsf{ShallowCyc}_d}$ that simply orders partial tree decompositions according to their shallow cyclicity, a globally minimum CTD for $\leq_{\mathsf{ShallowCyc}_d}$ will be a CTD that achieves the least shallow cyclicity. In particular, we get that all the globally minimum CTDs will, by definition, have the same shallow cyclicity. Hence, if any of them satisfies $\mathsf{ShallowCyc}_d$, then all of them do. Therefore this toptd is preference complete for $\mathsf{ShallowCyc}_d$.

Subtree constraints – even with the additional condition of preference completeness – still include a wide range of constraints. For example, all three example constraints mentioned above are preference complete. Informally, for shallow cyclicity, those partial TDs that become acyclic from lower depth are preferred. For partition clustering, we prefer the root node of the partial TD to be in the same partition as one of the children over introducing a new partition.

However, efficient search for constrained decompositions is not the only use of toptds. Using the same algorithmic framework, one can see them as a way to order decompositions by preference as long as the respective toptd $\leq$ is *strongly monotone*: that is, a partial tree decomposition (T, B) is globally minimal only if for each child c of the root, (T_c, B) is globally minimal. A typical example for strongly monotone toptd are cost functions for the estimated cost to evaluate a database query corresponding to the respective subtree. The cost of solving a query with a tree decomposition is made up of the costs for the individual subqueries of the child subtrees, plus some estimated cost of combining them, strong monotonicity is a natural simplifying assumption for this setting. For instance, consider a quasiordering of partial subtrees $\leq_{cost}$ that orders partial decompositions by such a cost estimate. In combination with the constraint ConCov, $(\mathsf{ConCov}, \leq_{cost})$ forms a preference complete subtree constraint, such that a satisfying TD is a globally minimal cost tree decomposition where every bag has a connected edge cover of size at most k. Thus, optimisation of tree decompositions for strongly monotone toptds is simply a special case of preference complete subtree constraints.

Our main goal in introducing the above concepts is the general study of the complexity of incorporating constraints into the computation of CTDs. We define the $(\mathcal{C}, \leq)$-CandidateTD problem as the problem of deciding for given hypergraph H and set S of candidate bags whether there exists a CompNF CTD that satisfies $\mathcal{C}$ and is minimal for $\leq$. We want to identify tractable fragments of $(\mathcal{C}, \leq_{\mathcal{C}})$-CandidateTD. Let us call a constraint and toptd *tractable* if one can decide in polynomial time w.r.t. the size of the original hypergraph H and the set S of candidate bags both, whether $\mathcal{C}$ holds for a partial tree decomposition of H, and whether $(T', B') <_{\mathcal{C}} (T, B)$ holds for partial tree decompositions of H.

Theorem 10. *Let $\mathcal{C}, \leq$ be a tractable constraint and toptd such that $\leq$ is preference complete for $\mathcal{C}$. Then $(\mathcal{C}, \leq)$-CandidateTD $\in$ PTIME.*

Proof Sketch. We obtain a polynomial time algorithm for $(\mathcal{C}, \leq)$-CandidateTD by modifying the main loop of Algorithm 1 as shown in Algorithm 2 (everything outside of the repeat loop remains unchanged). For a block (S, C) with basis X in Algorithm 2, the *basis* property of every block induces a (unique) tree decomposition for $S \cup C$, denoted as $Decomp(S, C, X)$. Where Algorithm 1 used dynamic programming to simply check for a possible way to satisfy the root block $(\varnothing, V(H))$. We instead use dynamic programming to find the preferred way, that is a basis that induces a globally minimal partial tree decomposition, to satisfy blocks. It is straightforward to verify the correctness of Algorithm 2. The polynomial time upper bound follows directly from the bound for Algorithm 1 and our tractability assumptions on $(\mathcal{C}, \leq_{\mathcal{C}})$. □

Our analysis here brings us back to one of the initially raised benefits of soft hypertree width: algorithmic flexibility. Our analysis, and Theorem 10 in particular, applies to any setting where a width measure can be effectively expressed in terms of CompNF candidate tree decompositions.

Corollary 11. *Let $\mathcal{C}$ be a subtree constraint, let $\leq$ be a preference complete toptd for $\mathcal{C}$, and suppose that $\mathcal{C}, \leq$ are tractable. Let k, i be non-negative integers. Then deciding $\mathcal{C}$-$shw^i(H) \leq k$ is feasible in polynomial time.*

Using CTDs, Gottlob et al. [15, 16] identified large fragments for which checking $ghw \leq k$ or $fhw \leq k$ is tractable. Hence, results analogous to Corollary 11 follow immediately also for tractable fragments of generalised and fractional hypertree decompositions.

6.2 Constraints in Other Approaches for Computing Decompositions

Bag-level constraints like ConCov are easy to enforce using existing combinatorial algorithms such as `det-k-decomp` [20] and `new-det-k-decomp` [12], which construct an HD top-down by combining up to k edges into λ-labels. Enforcing the ConCov is trivial in this case. However, enforcing more global constraints like PartClust while maintaining tractability remains unclear, as does integrating a cost function. These algorithms critically rely on caching for pairs of a bag B and vertex set C, whether there is an HD of $H[C]$ rooted at B. With constraints that apply to a larger portion of the decomposition, the caching mechanism no longer works. Other algorithms are also unsuitable: `log-k-decomp` [14], the fastest HD algorithm in practice, applies a divide-and-conquer strategy, splitting the hypergraph until a base case is reached, but never analyzes larger sections of the decomposition. The HtdSMT solver by Schidler and Szeider [29, 30] minimises HD width via SMT solving. It is unclear how to state constraints over these encodings. Furthermore, due to the reliance on SAT/SMT solvers, the method is not well suited for a theoretical analysis of tractable constrained decomposition methods.

The algorithm that comes closest to our algorithm for constructing a constrained decomposition is the `opt-k-decomp` algorithm from [28], which constructs a "weighted HD" of minimal cost up to a given width. The cost of an HD is defined by a function that assigns a cost to each node and each edge in the HD. The natural cost function assigns the cost of the join computation of the relations in λ_u to each node u and the cost of the (semi-)join between the bags at a node and its parent node to the corresponding edge. We also consider this type of cost function (and the goal to minimise the cost) as an important special case of a preference relation in our framework. However, our CTD-based construction of constrained decomposition allows for a greater variety of preferences and constraints, and the simplicity of Algorithm 2 facilitates a straightforward complexity analysis. Moreover, `opt-k-decomp` has been specifically designed for HDs. This is in contrast to our framework, which guarantees tractability for any combination of decompositions with tractable, preference complete subtree constraints as long as the set of candidate sets is polynomially bounded. As mentioned above, this is, for instance, the case for the tractable fragments of generalised and fractional HDs identified in [15, 16].

7 Experiments

While the focus of this paper is primarily on the theory of tractable decompositions of hypergraphs, the ultimate motivation of this work is drawn from applications to database query evaluation. We therefore include a focused experimental analysis of the practical effect of constraints and optimisation on candidate tree decompositions. The aim of our experiments is to gain insights into the effects of constraints and costs on candidate tree decompositions in practical settings. We perform experiments with cyclic queries over standard benchmarks (TPC-DS [26], LSQB [25] as well as a queries over the Hetionet Biomedical Knowledge Graph [23] from a recent cardinality

estimation benchmark [4]. In all experiments, we first compute candidate tree decompositions as in Algorithm 2 for the candidate bags as in Definition 3, i.e., we compute constrained soft hypertree decompositions. We then build on a recent line of research on rewriting the execution of Yannakakis' algorithm in standard SQL [5, 13], to execute Yannakakis' algorithm for these decompositions on standard relational DBMSs. For full details of our prototype implementation and experimental setup refer to the full version of this paper [24]. Our implementation is freely available at https://github.com/cem-okulmus/softhw-pods25. We note that a related analysis specifically for hypertree width was performed in the past by Scarcello et al. [28].

Results. We study the effect of optimising candidate tree decompositions that adhere to the ConCov constraint over two cost functions. The first cost function is based on estimated costs of joins and semi-joins by the DBMS (PostgreSQL) itself, the second is derived from the actual cardinalities of relations and joins (see [24] for details). The results of these experiments are summarised in Figure 4. These findings indicate that while certain decompositions can cut the execution time by more than half, others can be nearly ten times slower. This stark variation underscores the critical need for informed selection strategies when deploying decompositions for real-world database workloads. A second dimension of interest is the efficacy of the cost function. A priori, we expect the costs derived directly from the DBMS cost estimates to provide the stronger correlation between cost and time. However, we observe that the cost estimates of the DBMS are sometimes very unreliable, especially when it comes to cyclic queries[3]. Clearly,this makes it even harder to find good decompositions, as a cost function would ideally be based on good estimates. The comparison of cost functions in Figure 4 highlights this issue. There, we use as cost function the actual cardinalities as a proxy for good DBMS estimates. We see that the cost thus assigned to decompositions indeed neatly corresponds to query performance, whereas the cost using DBMS estimates inversely correlate to query performance.

To expand our analysis, we also tested two graph queries over Hetionet. In Figure 5. we report on the 10 cheapest decompositions of width 2. On these queries, both cost functions perform very similarly; we report costs based on DBMS estimates here. We still see a noticeable difference between decompositions, but more importantly, all of them are multiple times faster than the standard execution of the query in PostgreSQL. It turns out that connected covers alone are critical. In the right-most chart of Figure 5, we show the average time of executing the queries for 10 randomly chosen decompositions of width 2 with and without the ConCov constraint enforced. We see that the constraint alone is already sufficient to achieve significant improvements over standard execution in relational DBMSs.

[3]This is not particularly surprising and remains a widely studied topic of database research (see e.g., [11]).

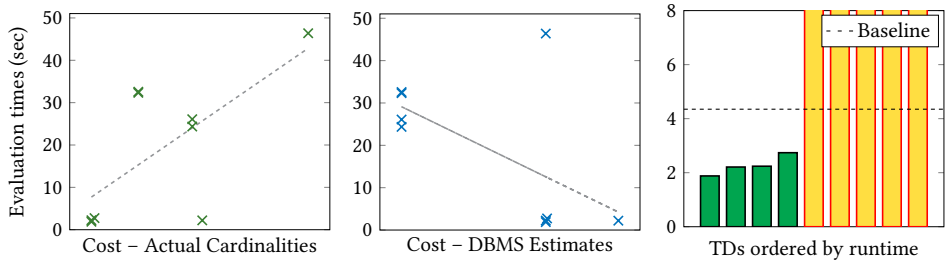


Fig. 4. Performance over a ConCov-*shw* 2 TPC-DS query using PostgreSQL as a backend.

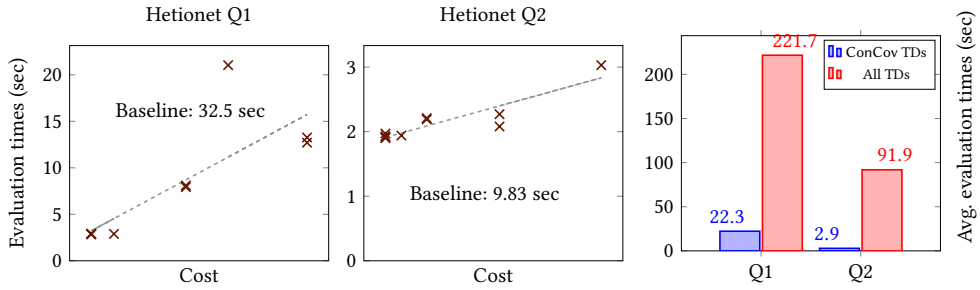


Fig. 5. Performance over two ConCov-*shw* 2 Hetionet queries using PostgreSQL as a backend.

Our implementation computes candidate tree decompositions closely following Algorithm 2. We find that the set of candidate bags in real world queries is very small, especially compared to the theoretical bounds on the set $\text{Soft}_{H,k}$. The TPC-DS query from Figure 4 has only 9 elements in $\text{Soft}_{H,2}$ (one of which does not satisfy ConCov). The two queries in Figure 5 both have 25 candidate bags, 16 of which satisfy ConCov. Accordingly, it takes only a few milliseconds to enumerate all decompositions ranked by cost in these examples.

8 Conclusion

In this work, we have introduced the concept of soft hypertree decompositions (soft HDs) and the associated measure of soft hypertree width (*shw*). Despite avoiding the special condition of HDs, we retained the tractability of deciding whether a given CQ has width at most k. At the same time, *shw* is never greater than *hw* and it may allow for strictly smaller widths. Most importantly, it provides more algorithmic flexibility. Building on the framework of candidate tree decompositions (CTDs), we have demonstrated how to incorporate diverse constraints and preferences into the decomposition process, enabling more specialised decompositions that take application-specific concerns beyond width into account.

On a purely theoretical level, we additionally introduce a hierarchy of refinements of *shw* that still yields tractable width measures for every fixed step on the hierarchy. In particular, we show that *ghw* emerges as the limit of this hierarchy. A yet unexplored facet of this hierarchy is that Lemma 6 can possibly be strengthened to bound α more tightly with respect to structural properties of hypergraphs. Moreover, by Appendix B.1 and previous results [2] we have that

$$ghw(H) = shw^{\infty}(H) \leq \cdots \leq shw^{i+1}(H) \leq shw^{i}(H) \leq \cdots \leq shw^{0}(H) \leq hw(H) \leq 3 \cdot ghw(H) + 1.$$

The natural question then is whether these bounds can be tightened for steps in the hierarchy.

Acknowledgments

This work was supported by the Vienna Science and Technology Fund (WWTF) [10.47379/VRG18013, 10.47379/ICT2201]. Georg Gottlob acknowledges support from the PNRR FAIR project Future AI Research (PE00000013), Spoke 9 - Green-aware AI, under the NRRP MUR program funded by the NextGenerationEU initiative. The work of Cem Okulmus is supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

[1] Adler, I. Marshals, monotone marshals, and hypertree-width. *J. Graph Theory 47*, 4 (2004), 275–296.

[2] Adler, I., Gottlob, G., and Grohe, M. Hypertree width and related hypergraph invariants. *Eur. J. Comb. 28*, 8 (2007), 2167–2181.

[3] Arnborg, S., Corneil, D. G., and Proskurowski, A. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods 8*, 2 (1987), 277–284.

[4] Birler, A., Kemper, A., and Neumann, T. Robust join processing with diamond hardened joins. *Proc. VLDB Endow. 17*, 11 (2024), 3215–3228.

[5] Böhm, D. To rewrite or not to rewrite: Decision making in query optimization of sql queries. Master's thesis, Technische Universität Wien, 2024.

[6] Bouchitté, V., and Todinca, I. Minimal triangulations for graphs with "few" minimal separators. In *Proc. ESA* (1998), vol. 1461 of *Lecture Notes in Computer Science*, Springer, pp. 344–355.

[7] Bouchitté, V., and Todinca, I. Treewidth and minimum fill-in of weakly triangulated graphs. In *Proc. STACS* (1999), vol. 1563 of *Lecture Notes in Computer Science*, Springer, pp. 197–206.

[8] Bouchitté, V., and Todinca, I. Listing all potential maximal cliques of a graph. In *Proc. STACS* (2000), vol. 1770 of *Lecture Notes in Computer Science*, Springer, pp. 503–515.

[9] Bouchitté, V., and Todinca, I. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM J. Comput. 31*, 1 (2001), 212–232.

[10] Bouchitté, V., and Todinca, I. Listing all potential maximal cliques of a graph. *Theor. Comput. Sci. 276*, 1-2 (2002), 17–32.

[11] Chen, J., Huang, Y., Wang, M., Salihoglu, S., and Salem, K. Accurate summary-based cardinality estimation through the lens of cardinality estimation graphs. *SIGMOD Rec. 52*, 1 (2023), 94–102.

[12] Fischl, W., Gottlob, G., Longo, D. M., and Pichler, R. Hyperbench: A benchmark and tool for hypergraphs and empirical findings. *ACM J. Exp. Algorithmics 26* (2021), 1.6:1–1.6:40.

[13] Gottlob, G., Lanzinger, M., Longo, D. M., Okulmus, C., Pichler, R., and Selzer, A. Structure-guided query evaluation: Towards bridging the gap from theory to practice. *CoRR abs/2303.02723* (2023).

[14] Gottlob, G., Lanzinger, M., Okulmus, C., and Pichler, R. Fast parallel hypertree decompositions in logarithmic recursion depth. *ACM Trans. Database Syst. 49*, 1 (2024), 1:1–1:43.

[15] Gottlob, G., Lanzinger, M., Pichler, R., and Razgon, I. Fractional covers of hypergraphs with bounded multi-intersection. In *Proc. MFCS* (2020), vol. 170 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 41:1–41:14.

[16] Gottlob, G., Lanzinger, M., Pichler, R., and Razgon, I. Complexity analysis of generalized and fractional hypertree decompositions. *J. ACM 68*, 5 (2021), 38:1–38:50.

[17] Gottlob, G., Leone, N., and Scarcello, F. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci. 64*, 3 (2002), 579–627.

[18] Gottlob, G., Miklós, Z., and Schwentick, T. Generalized hypertree decompositions: NP-hardness and tractable variants. *J. ACM 56*, 6 (2009), 30:1–30:32.

[19] Gottlob, G., Okulmus, C., and Pichler, R. Fast and parallel decomposition of constraint satisfaction problems. *Constraints An Int. J. 27*, 3 (2022), 284–326.

[20] Gottlob, G., and Samer, M. A backtracking-based algorithm for hypertree decomposition. *ACM J. Exp. Algorithmics 13* (2008).

[21] Gottlob, G., Scarcello, F., and Sideri, M. Fixed-parameter complexity in AI and nonmonotonic reasoning. *Artif. Intell. 138*, 1-2 (2002), 55–86.

[22] Grohe, M., and Marx, D. Constraint solving via fractional edge covers. *ACM Trans. Algorithms 11*, 1 (2014), 4:1–4:20.

[23] Himmelstein, D. S., Lizee, A., Hessler, C., Brueggeman, L., Chen, S. L., Hadley, D., Green, A., Khankhanian, P., and Baranzini, S. E. Systematic integration of biomedical knowledge prioritizes drugs for repurposing. *eLife 6* (sep 2017), e26726.

[24] Lanzinger, M., Okulmus, C., Pichler, R., Selzer, A., and Gottlob, G. Soft and constrained hypertree width. *CoRR abs/2412.11669* (2024).

[25] Mhedhbi, A., Lissandrini, M., Kuiper, L., Waudby, J., and Szárnyas, G. LSQB: a large-scale subgraph query benchmark. In *Proc. GRADES* (2021), ACM, pp. 8:1–8:11.

[26] Pöss, M., Smith, B., Kollár, L., and Larson, P. Tpc-ds, taking decision support benchmarking to the next level. In *Proc. SIGMOD* (2002), ACM, pp. 582–587.

[27] Ravid, N., Medini, D., and Kimelfeld, B. Ranked enumeration of minimal triangulations. In *Proc. PODS* (2019), ACM, pp. 74–88.

[28] Scarcello, F., Greco, G., and Leone, N. Weighted hypertree decompositions and optimal query plans. *J. Comput. Syst. Sci. 73*, 3 (2007), 475–506.

[29] Schidler, A., and Szeider, S. Computing optimal hypertree decompositions. In *Proc. ALENEX* (2020), SIAM, pp. 1–11.

[30] Schidler, A., and Szeider, S. Computing optimal hypertree decompositions with SAT. *Artif. Intell. 325* (2023), 104015.

[31] Yannakakis, M. Algorithms for acyclic database schemes. In *Proc. VLDB* (1981), IEEE Computer Society, pp. 82–94.

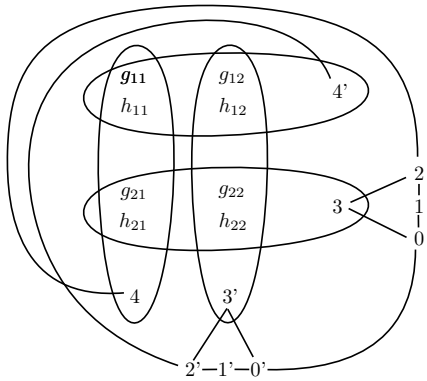


Fig. 6. The hypergraph H_3 (some edges omitted) with $ghw(H_3) = shw(H_3) = 3$ and $hw(H_3) = 4$ (adapted from [1]).

A Further Details on Section 4

A.1 SHW vs. HW

We now present an example of a hypergraph with $shw = 3$ and $hw = 4$. To this end, We consider the hypergraph H_3 adapted from [1], defined as follows. Let $G = \{g_{11}, g_{12}, g_{21}, g_{22}\}$, $H = \{h_{11}, h_{12}, h_{21}, h_{22}\}$, and $V = \{0, 1, 2, 3, 4, 0', 1', 2', 3', 4'\}$. The vertices of H_3 are the set $H \cup G \cup V$. The edges of H_3 are

$$\begin{aligned} E(H_3) = &\{\{w, v\} \mid w \in G \cup H, v \in V\} \cup \{\{2, 4\}, \{2', 4'\}\} \cup \\ &\{0, 0'\} \cup \{\{0, 1\}, \{1, 2\}, \{0, 3\}, \{2, 3\}\} \cup \\ &\{\{0', 1'\}, \{1', 2'\}, \{0', 3'\}, \{2', 3'\}\} \cup \\ &\{\{g_{11}, g_{12}, h_{11}, h_{12}, 4'\}, \{g_{21}, g_{22}, h_{21}, h_{22}, 3\}, \\ &\{g_{11}, g_{21}, h_{11}, h_{21}, 4\}, \{g_{12}, g_{22}, h_{12}, h_{22}, 3'\}\} \end{aligned}$$

The hypergraph is shown in Figure 6 with the edges $\{\{w, v\} \mid w \in G \cup H, v \in V\}$ omitted. We have $ghw(H_3) = 3$ and $hw(H_3) = 4$. We now show that indeed also $shw(H_3) = 3$. A witnessing decomposition is given in Figure 7. Note that, strictly speaking, only $mw(H_3) = 3$ is shown in [1] and, in general, mw is only a lower bound on ghw. However, it is easy to verify that $ghw(H_3) = 3$ holds by inspecting the winning strategy for 3 marshals in [1]. Moreover, the ghw-result is, of course, implicit when we show $shw(H_3) = 3$ next. Actually, the bags in the decomposition given in Figure 7 are precisely the χ-labels one would choose in a GHD of width 3. And the λ-labels are obtained from the (non-monotone) winning strategy for 3 marshals in [1].

To prove $shw(H_3) = 3$, it remains to verify that all the bags in Figure 7 are contained in $\text{Soft}_{H_3,3}$. We see that $G \cup H$ are in all the bags and we, therefore, focus on the remaining part of the bags in our discussion. For the root, this is $\{3, 0', 0\}$. Here, the natural cover λ_1 consists of the two large horizontal edges in Figure 6 together with the edge $\{0, 0'\}$. The union $\bigcup \lambda_1$ contains the additional vertex $4'$. As λ_2 of Equation (1) we use the same two large edges plus $\{4', 2'\}$. As above, there is only one $[\lambda_2]$-edge component, which contains all vertices but $4'$. Note that, in contrast to the previous example, $4'$ in fact has high degree, but all of the edges that touch $4'$ are inside the separator.

Except for the bag $G \cup H \cup \{2, 4\}$, the bags always miss either vertex 4 or $4'$ from the "natural" covers, and the arguments are analogous to the ones above for the root bag. For the remaining bag, we consider the cover λ_1 consisting of the two vertical large edges plus the additional edge

$\{2, 4\}$. Thus, we have the problematic vertex $3'$, which is contained in $\bigcup \lambda_1$ but not in the bag. Here, we observe a more complex scenario than before. As λ_2 take the two large horizontal edges plus $\{0', 1'\}$. This splits H_3 into two $[\lambda_2]$-components: one with vertices $G \cup H \cup \{0', 0, 1, 2, 3, 4\}$ and another with vertices $G \cup H \cup \{1', 2', 3', 4'\}$. The intersection of $\bigcup \lambda_1$ with the first component will produce the desired bag.

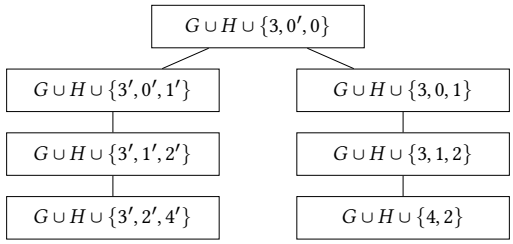


Fig. 7. A soft hypertree decomposition of H_3 with width 3.

B Further Details for Section 5

Proof of Lemma 3. We first prove $E^{(i)} \subseteq \mathrm{Soft}^i_{H,k}$. Indeed, let e' be an arbitrary edge in $E^{(i)}$. Now pick $\lambda_1 = \{e'\}$ and $\lambda_2 = \emptyset$. Hence, in particular, $C = E(H)$ itself is the only $[\lambda_2]$-component in $E(H)$. We thus get $e' = B$ with $B = (\bigcup \lambda_1) \cap (\bigcup C)$ and, therefore, $e' \in \mathrm{Soft}^i_{H,k}$.

From this, the inclusion $E^{(i)} \subseteq E^{(i+1)}$ follows easily. Indeed, by Definition 6, we have $E^{(i+1)} = E^{(i)} \bowtie \mathrm{Soft}^i_{H,k}$. Now let e' be an arbitrary edge in $E^{(i)}$. By $E^{(i)} \subseteq \mathrm{Soft}^i_{H,k}$, we conclude that $e' \in \mathrm{Soft}^i_{H,k}$ and therefore, also $e' = e' \cap e' \in E^{(i)} \bowtie \mathrm{Soft}^i_{H,k} = E^{(i+1)}$ holds.

Finally, consider an arbitrary element $B \in \mathrm{Soft}^i_{H,k}$, i.e., B is of the form $B = (\bigcup \lambda_1) \cap (\bigcup C)$, where C is a $[\lambda_2]$-component of H, λ_1 is a set of at most k elements of $E^{(i)}$, and λ_2 is a set of at most k elements of $E(H)$. From $E^{(i)} \subseteq E^{(i+1)}$, it follows that λ_1 can also be considered as a set of at most k elements of $E^{(i+1)}$. Hence, B is also contained in $\mathrm{Soft}^{i+1}_{H,k}$. □

Proof of Lemma 4. By Lemma 3, $E^{(i)} \subseteq \mathrm{Soft}^i_{H,k}$ holds and, by Definition 6, we have $E^{(i+1)} = E^{(i)} \bowtie \mathrm{Soft}^i_{H,k}$. Hence, $E^{(i+1)} = E^{(i)} \bowtie \mathrm{Soft}^i_{H,k} \subseteq \mathrm{Soft}^i_{H,k} \bowtie \mathrm{Soft}^i_{H,k}$ and, therefore, $|E^{(i+1)}| \leq |\mathrm{Soft}^i_{H,k}|^2$. According to Definition 6, $\mathrm{Soft}^{i+1}_{H,k}$ contains all edges of the form $B = (\bigcup \lambda_1) \cap (\bigcup C)$, where C is a $[\lambda_2]$-component of H, λ_1 is a set of at most k elements of $E^{(i+1)}$, and λ_2 is a set of at most k elements of $E(H)$. Hence, there are at most $|E^{(i+1)}|^{k+1}$ possible choices for λ_1, at most $|E(H)|^{k+1}$ many possible choices for λ_2, and at most $|E(H)|$ possible choices of a $[\lambda_2]$-component for given λ_2. In total, using the monotonicity of $E^{(i)}$ and, hence, the relationship $E(H) \subseteq E^{(i+1)}$, we thus get $|\mathrm{Soft}^{i+1}_{H,k}| \leq |E^{(i+1)}|^{2k+3}$. Together with $|E^{(i+1)}| \leq |\mathrm{Soft}^i_{H,k}|^2$, we get the desired inequality $|\mathrm{Soft}^{i+1}_{H,k}| \leq (|\mathrm{Soft}^i_{H,k}|)^{4k+6}$. □

Proof of Lemma 6. To avoid expressions of the form $\bigcap\bigcup$, we introduce the following notation: for a set C of edges, we write $V(C)$ to denote the set of all vertices contained in C, i.e., $V(C) = \bigcup C$.

The proof proceeds in three steps.

Claim A. For every $i \geq 0$, the elements in $E^{(i)}$ can be represented in the form

$$e \cap \Big(\Big[\bigcap_{e_1 \in E_1} e_1 \cap \bigcap_{C_1 \in \mathcal{C}_1} V(C_1) \Big] \cup \cdots \cup \Big[\bigcap_{e_m \in E_m} e_m \cap \bigcap_{C_m \in \mathcal{C}_m} V(C_m) \Big] \Big)$$

for some $m \geq 0$, where $e \in E(H)$, $E_1, \ldots, E_m \subseteq E(H)$, and $\mathcal{C}$ is a set of sets of edges, such that every element $C \in \mathcal{C}$ is a $[\lambda]$-component, where λ is a set of at most k edges from $E(H)$.

The claim is easily proved by induction on i. For the induction begin, we have $E^{(0)} = E(H)$ and every element $e \in E(H)$ clearly admits such a representation, namely by setting $m = 0$. For the induction step, we use the induction hypothesis for the elements in $E^{(i)}$, expand the definition of the elements in $\text{Soft}^i_{H,k}$ and $E^{(i+1)}$ according to Definition 6, and then simply apply distributivity of $\cup$ and $\cap$.

The above claim implicitly confirms that all elements contained in $E^{(i)}$ are either edges in $E(H)$ or subedges thereof. Let us now ignore for a while the sets $\mathcal{C}$ of sets of components in the above representation of elements in $E^{(i)}$. The next claims essentially state that after n iterations, for the sets E_j in the above representation, we may use any non-empty subset $E(H)$ and, after another n iterations, no new elements (assuming all sets $\mathcal{C}_j$ to be empty) can be produced.

Claim B. Let $i \geq 0$. Then $E^{(i)}$ contains *all* possible vertex sets S that can be represented as $S = \bigcap_{e' \in E'} e'$ for some $E' \subseteq E(H)$ with $|E'| \leq i$.

Again, the claim is easily proved by induction on i. In the definition of $\text{Soft}^i_{H,k}$, we always choose λ_1 as a singleton subset of $E(H)$ and $\lambda_2 = \varnothing$. Hence, in particular, after n iterations, we have all possible sets of the form $S = \bigcap_{e' \in E'} e'$ for all possible subsets $E' \subseteq E(H)$ at our disposal. This fact will be made use of in the proof of the next claim.

Claim C. Suppose that in the definition of $\text{Soft}^i_{H,k}$ in Definition 6, we only allow $\lambda_2 = \varnothing$, i.e., the only component we can thus construct is $C = E(H)$ and, therefore, $V(C) = V(H)$ holds. Then, for every $i \geq 2n$, we have $E^{(i)} = E^{(i+1)}$.

To prove the claim, we inspect the above representation of elements in $E^{(i)}$, but assuming $V(C) = V(H)$ for every C. That is, all elements are of the form

$$e \cap \Big(\Big[\bigcap_{e_1 \in E_1} e_1 \Big] \cup \cdots \cup \Big[\bigcap_{e_m \in E_m} e_m \Big] \Big),$$

for some $m \geq 0$. By Claim B, we know that from $i \geq n$ on, we may choose as set E_j any subset of $E(H)$. It is easy to show, by induction on i, that we can produce any element of the form $e \cap \Big(\Big[\bigcap_{e_1 \in E_1} e_1 \Big] \cup \cdots \cup \Big[\bigcap_{e_i \in E_i} e_i \Big] \Big)$ in $E^{(n+i)}$. Then, to prove Claim C, it suffices to show that choosing i bigger than n does not allow us to produce new subedgdes of e. To the contrary, assume that there exists $n' > n$ and a set $e' = e \cap \Big(\Big[\bigcap_{e'_1 \in E'_1} e'_1 \Big] \cup \cdots \cup \Big[\bigcap_{e'_{n'} \in E'_{n'}} e'_{n'} \Big] \Big)$ that cannot be represented as $e' = e \cap \Big(\Big[\bigcap_{e_1 \in E_1} e_1 \Big] \cup \cdots \cup \Big[\bigcap_{e_n \in E_n} e_n \Big] \Big)$. Moreover, assume that n' is minimal with this property. Then we derive a contradiction by inspecting the sequence $s_1 = e \cap \Big(\Big[\bigcap_{e'_1 \in E'_1} e'_1 \Big] \Big)$, $s_2 = e \cap \Big(\Big[\bigcap_{e'_1 \in E'_1} e'_1 \Big] \cup \Big[\bigcap_{e'_2 \in E'_2} e'_2 \Big] \Big)$, …, $s_{n'} = e \cap \Big(\Big[\bigcap_{e'_1 \in E'_1} e'_1 \Big] \cup \cdots \cup \Big[\bigcap_{e'_{n'} \in E'_{n'}} e'_{n'} \Big] \Big)$. Clearly, this sequence is monotonically increasing and all elements s_j are subedges of e. By $n' > n \geq |V(H)|$, the sequence cannot be strictly increasing. Hence, there exists j, with $s_j = s_{j-1}$. But then we can leave $\Big[\bigcap_{e'_j \in E'_j} e'_j \Big]$ out from the representation $e' = e \cap \Big(\Big[\bigcap_{e'_1 \in E'_1} e'_1 \Big] \cup \cdots \cup \Big[\bigcap_{e'_n \in E'_n} e'_n \Big] \Big)$ and still get e'. This contradicts the minimality of n'.

Completion of the proof of the lemma. From Claim C it follows that after $2n$ iterations, any new elements can only be obtained via the intersections with sets of the form $\bigcap_{C \in \mathcal{C}} V(C)$, where $\mathcal{C}$ is a set of components. Recall that, according to Definition 6, components are only built w.r.t. sets λ_2 of sets of edges from $E(H)$. Hence, all possible components C are available already in the first iteration. Hence, analogously to Claim B, we can show that after n successive iterations, we can

choose any sets $\mathcal{C}_j$ with $|\mathcal{C}_j| \leq i$ for constructing elements of the form

$$e \cap \Big(\Big[\bigcap_{e_1 \in E_1} e_1 \cap \bigcap_{C_1 \in \mathcal{C}_1} V(C_1) \Big] \cup \cdots \cup \Big[\bigcap_{e_m \in E_m} e_m \cap \bigcap_{C_m \in \mathcal{C}_m} V(C_m) \Big] \Big),$$

in $E^{(2n+i)}$. In particular, in $E^{(3n)}$, we get any element of the above form for any possible choice of $\mathcal{C}_j$ with $|\mathcal{C}_j| \leq n$. Finally, analogously to Claim C, we can show that taking sets $\mathcal{C}_j$ with $|\mathcal{C}_j| > n$ cannot contribute any new element. Again, we make use of the fact that $n \geq |V(H)|$ and it only makes sense to add an element C to $\mathcal{C}_j$ if the intersection with $V(C)$ allows us to delete another vertex from e. But we cannot delete more than n vertices. □

B.1 Proof of Theorem 7

Proof. Let $\mathcal{G} = (T, \lambda, \chi)$ be a GHD of width $ghw(H)$ of hypergraph H. For a node u in T, we write λ_u and χ_u to denote the λ-label and χ-label at u. It suffices to show that, for some $m \geq 0$, the following property holds: for every node u in T and every $e \in E(H)$ with $e \in \lambda_u$, the subedge $e \cap \chi_u$ is contained in $E^{(m)}$. Clearly, if this is the case, then every bag χ_u is contained in $\text{Soft}^m_{H,k}$, since we can define λ_1 by replacing every e in λ by $e \cap \chi_u$ and setting $\lambda_2 = \varnothing$. Then $\mathcal{G}' = (T, \chi)$ is a candidate TD of $\text{Soft}^m_{H,k}$, and, therefore, we get $ghw(H) = shw^m(H)$.

To obtain such an m with $(e \cap \chi_u) \in E^{(m)}$ for every node u in T and edge $e \in \lambda_u$, we recall the following result from [16], Lemma 5.12: Let $e \in \lambda_u$ with $e \notin \chi_u$ for some node u and let u' be a node with $e \subseteq \chi_{u'}$ (such a node u' must exist in a GHD). Moreover, let $u = u_0, \ldots, u_\ell = u'$ be the path from u to u' in T. Then the following property holds: $e \cap \chi_u = e \cap \big(\bigcap_{j=1}^{\ell} \bigcup \lambda_{u_j} \big)$.
Clearly, for every j, the bag $\bigcup \lambda_{u_j}$ is in $\text{Soft}_{H,k}$ and, therefore, the subedge $s_j = e \cap \bigcup \lambda_{u_j}$ of e is in $E^{(1)}$. We claim that, for every $i \geq 1$, it holds that $\big(\bigcap_{j=1}^{i} s_j \big)$ is contained in $E^{(i)}$.

Proof of the claim. We proceed by induction on i. For $i = 1$, the claim reduces to the property that $s_1 = e \cap \bigcup \lambda_{u_1}$ is in $E^{(1)}$, which we have just established. Now suppose that the claim holds for arbitrary $i \geq 1$. We have to show that $\big(\bigcap_{j=1}^{i+1} s_j \big)$ is contained in $E^{(i+1)}$. On the one hand, we have that every $s_j = e \cap \bigcup \lambda_{u_j}$ is in $E^{(1)}$ and, therefore, also in $E^{(i)+1}$ and in $\text{Soft}^{i+1}_{H,k}$. On the other hand, by the induction hypothesis, $\big(\bigcap_{j=1}^{i} s_j \big)$ is contained in $E^{(i)}$ and, therefore, also in $\text{Soft}^{i}_{H,k}$. Hence, by Definition 6, $\big(\bigcap_{j=1}^{i+1} s_j \big)$ is contained in $E^{(i+1)}$.

We conclude that, for every node u in T and every edge $e \in \lambda_u$, the subedge $e \cap \chi_u = e \cap \big(\bigcap_{j=1}^{\ell} \bigcup \lambda_{u_j} \big)$ of e is contained in $\text{Soft}^p_{H,k}$, where p is the maximal length of paths in T. That is, we have $\chi_u = \bigcup \lambda'_u$, where λ'_u is obtained by replacing every edge $e \in \lambda_u$ by the subedge $e \cap \chi_u$. Hence, (T, χ) is a candidate TD of $\text{Soft}^p_{H,k}$ and, therefore, $shw^p(H) = ghw(H)$ □

B.2 Proof of Theorem 9

Proof. Let (H, α, β) be the switch graph from Example 2 of [1]. For convenience we recall the definition here. Let $n \geq 2$. Let N_1 be a punctured hypergraph with $\text{mw}(N_1) = \text{mon-mw}(N_1) > n$. Let N_2 be a disjoint copy of N_1. Let (H, α, β) be the switch graph with

$$V(H) := V(N_1) \uplus V(N_2) \uplus \{m'\} \uplus \{e_{1i} \mid i = 1, \ldots, n+1\} \uplus \{e_{2i} \mid i = 1, \ldots, n+1\}.$$

and

$$E(H) := E(N_1) \uplus E(N_2) \uplus \{\{e_{1i}\} \mid i = 1, \ldots, n+1\} \uplus \{\{e_{2i}\} \mid i = 1, \ldots, n+1\} \uplus \{\{m\}\} \quad (2)$$

$$\uplus \{\{e_{1i}, n_1\} \mid i = 1, \ldots, n+1, n_1 \in V(N_1)\} \uplus \{\{e_{2i}, n_2\} \mid i = 1, \ldots, n+1 mn_2 \in V(N_2)\} \quad (3)$$

$$\uplus \{\{m', n\} \mid n \in V(N_1) \cup V(N_2)\}. \quad (4)$$

and

$$\alpha = \{\{e_{11}\}, \ldots, \{e_{1,n+1}\}, \{m'\}\} \cup V(N_2),$$

$$\beta = \{\{e_{21}\}, \ldots, \{e_{2,n+1}\}, \{m'\}\} \cup V(N_1).$$

Let $s := |V(N_1)| + (n+1) + 1 = |\alpha| = |\beta|$.

Let H_{BOG} be the hypergraphs with n-machinists associated to (H, α, β) as in the proof of Theorem 4.2 in [1]. Let $E(N_1)$ and $E(N_2)$ be the sets of edges that are fully contained within $V(N_1)$ and $V(N_2)$, respectively. Furthermore, let $E^+(N_1) = E(N_1) \cup \{\{m', n\} \mid n \in N_1\}$, and analogously for $E^+(N_2)$. It is known that $ghw(H_{BOG}) = s + 1$ and $hw(H_{BOG}) = s + n + 1$ [1].

We make a critical modification to the construction H_{BOG}. Add the new vertex $\star$, and the edges $\{\star, g\}$ for every $g \in B$. That is, $\star$ is adjacent exactly to the balloon vertices. We denote the result of this construction as $H^\star_{BOG}$. We will argue the following.

Theorem 12.

$$shw^1(H^\star_{BOG}) = ghw(H_{BOG}) \leq hw(H_{BOG}) - n = hw(H^\star_{BOG})$$

The inequality is an implicit consequence of the construction for Theorem 4.2 in [1] (and can be validated also through our argument). Furthermore H_{BOG} is an induced subhypergraph of $H^\star_{BOG}$ (induced on all vertices but $\star$). It is easy to see that hypertreewidth is monotone over induced subhypergraphs, thus $hw(H_{BOG}) \leq hw(H^\star_{BOG})$. To show the equality we explicitly give a (CompNF) tree decomposition for $H^\star_{BOG}$ using only bags from $\text{Soft}^1_{H^\star_{BOG}, s+1}$ in Figure 8. We verify this through a number of independent claims.

Claim. The TD from Figure 8 is a valid tree decomposition.

It is straightforward to check that every edge of $H^\star_{BOG}$ is covered. To cover the eyelet edges some care is required to maintain connectedness. Recall that the eyelet edges are for every $g \in B, h \in E(H)$ and $\ell \in [m]$ (where m is an integer, unrelated to the vertex m' in the switch graph.) there are edges $\{g, p_\ell(g,h)\}$ and $h \cup \{p_\ell(g,h)\}$. Observe that the edges $E(H)$ can be partitioned into $E^+(N_1)$, $E^+(N_2)$ as well as the sets of edges connecting every e_{1i} to all vertices in N_1 and e_{2i} to all vertices in N_2. With this partition in mind one can then observe that the decomposition in Figure 8 satisfies the connectedness condition. ▲

Claim. $\{\star\} \cup B \in \text{Soft}^0_{H^\star_{BOG}, s+1}$.

Take as λ_2 in Equation (1) the edges $\{a_1, \ldots, a_s\}$. We have $\bigcup \lambda_2 \supset B$ (cf., [1]). Then, $\star$ is not $[\lambda_2]$-connected to any other vertex. Hence there is a $[\lambda_2]$-component C where $\bigcup C$ equals $\star$ plus all adjacent vertices to $\star$. By definition then $\bigcup C = \{star\} \cup B$. ▲

Claim. For every i, the subedges $a'_i = \{g_{i1}, \ldots, g_{is}\}$ and $b'_j = \{g_{1j}, \ldots, g_{sj}\}$ are in $E^{(1)}$.

We argue for a'_i, the case for b'_j is symmetric. Recall from the construction of H_{BOG} that there is an edge $a_i = \{g_{i1}, \ldots, g_{is}\} \cup \alpha_i$ for every $i \in [s]$. Thus $a_i \in E^{(0)}$, and from the previous claim $\{\star\} \cup B \in \text{Soft}^0_{H^\star_{BOG}, s+1}$. Hence $a'_i \in E^{(0)} \bowtie \text{Soft}^0_{H^\star_{BOG}, s+1} = E^{(1)}$ ▲

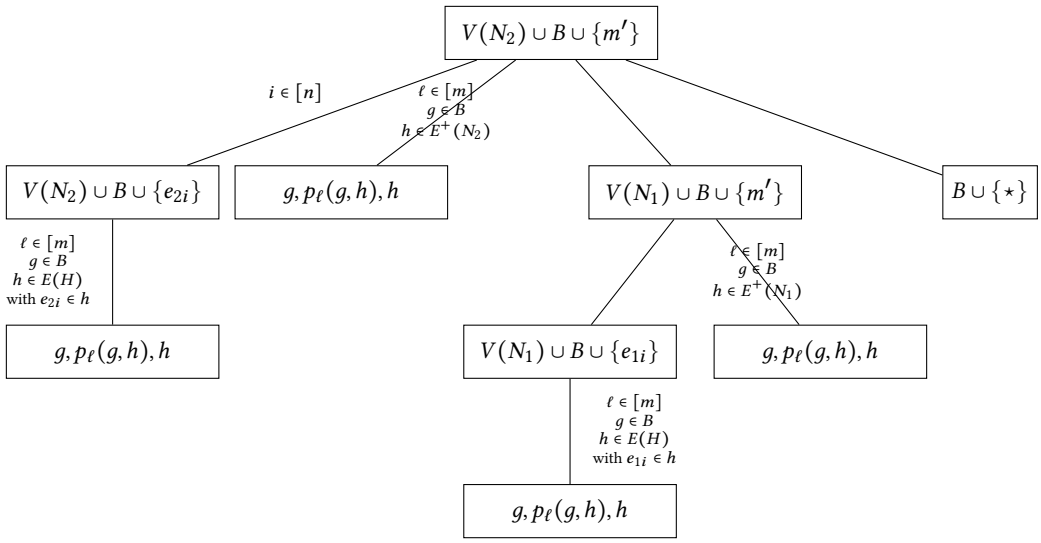


Fig. 8. A soft hypertree decomposition of width $s+1$ for $H^\star_{BOG}$. We note that the terms $\ell \in [m]$ refer to the number m of eyelet vertices from the construction of H_{BOG} in [1]. For consistency we minimised changes in the naming from the reference, despite the slightly confusing similarity to m' (also called m in [1]), a vertex in the switch graphs that the construction is applied to.

Claim. For every $i \in [n+1]$, $B \cup V(N_2)$ is the union of s edges in $E^{(1)}$. The same holds also for $B \cup V(N_1)$.

The respective edges are $X = \{a'_1, \ldots, a'_{n+2}, a_{n+3}, \ldots, a_s\}$. That is we take the subedges of a_i for the first $n+2$ indexes, i.e., those indexes where $\alpha_i = e_{1i}$ or $\alpha_i = m$. The α for the rest of the indexes make up exactly the set $V(N_1)$ by definition. In summary we have $\bigcup X = (\bigcup_{i=1}^{s} a_i) \setminus \{e_{11}, \ldots, e_{1,n+1}, m'\} = B \cup V(N_2)$. For the case with $V(N_1)$ make the same construction for the b_j edges. ▲

Claim. The TD from Figure 8 uses only bags from $\mathrm{Soft}_{H^\star_{BOG}, s+1}$.

There are 4 kinds of bags that are not simply unions of $s+1$ (even 2) edges. All are either of the form $X = V(N_2) \cup B \cup \{v\}$ or, $Y = V(N_1) \cup B \cup \{v\}$. From the previous claim it then follows immediately that these are in $\mathrm{Soft}_{H^\star_{BOG}, s+1}$. ▲

□

The corresponding results in [1] are stated in terms of games, in particular it is shown that the gap between marshal width *mw* and monotone marshal width *mon-mw* on H_{BOG} is at least n. It has been considered folklore that this implies also that the gap between *ghw* and *hw* is at least n on these hypergraphs (and thus arbitrarily high in general). But to the best of our knowledge, no proof of this is explicitly written down in the literature. The problem being, that while $\textit{mon-mw}(H) = \textit{hw}(H)$, the marshal width itself is not necessarily equal to *ghw* but only a lower bound. We note that Figure 8 is also a generalised hypertree decomposition of width $s+1$, and also serves to fill in this small gap in the literature.

Received December 2024; revised February 2025; accepted March 2025