FPT Approximation of Generalised Hypertree Width for Bounded Intersection Hypergraphs

Matthias Lanzinger
TU Wien, Austria
University of Oxford, UK

Igor Razgon
Birkbeck, University of London, UK

Abstract

Generalised hypertree width (*ghw*) is a hypergraph parameter that is central to the tractability of many prominent problems with natural hypergraph structure. Computing *ghw* of a hypergraph is notoriously hard. The decision version of the problem, checking whether $ghw(H) \leq k$, is paraNP-hard when parameterised by k. Furthermore, approximation of *ghw* is at least as hard as approximation of Set-Cover, which is known to not admit any FPT approximation algorithms.

Research in the computation of *ghw* so far has focused on identifying structural restrictions to hypergraphs – such as bounds on the size of edge intersections – that permit XP algorithms for *ghw*. Yet, even under these restrictions that problem has so far evaded any kind of FPT algorithm. In this paper we make the first step towards FPT algorithms for *ghw* by showing that the parameter can be approximated in FPT time for graphs of bounded edge intersection size. In concrete terms we show that there exists an FPT algorithm, parameterised by k and d, that for input hypergraph H with maximal cardinality of edge intersections d and integer k either outputs a tree decomposition with $ghw(H) \leq 4k(k+d+1)(2k-1)$, or rejects, in which case it is guaranteed that $ghw(H) > k$. Thus, in the special case of hypergraphs of bounded edge intersection, we obtain an FPT $O(k^3)$-approximation algorithm for *ghw*.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Theory of computation → Approximation algorithms analysis; Mathematics of computing → Hypergraphs

Keywords and phrases generalized hypertree width, hypergraphs, parameterized algorithms, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.48

Related Version *Full Version*: `https://arxiv.org/abs/2309.17049` [17]

Funding *Matthias Lanzinger*: Matthias Lanzinger acknowledges support by the Royal Society "RAISON DATA" project (Reference No. RP\R1\201074). Lanzinger has been supported by the Vienna Science and Technology Fund (WWTF) [10.47379/ICT2201].

1 Introduction

A tree decomposition of a hypergraph H is a pair $(T, \mathbf{B})$ where T is a tree and $\mathbf{B} : V(T) \to 2^{V(H)}$ assigns a *bag* to each node, that satisfies certain properties. The treewidth of a decomposition is $\max_{u \in V(T)} |\mathbf{B}(u)| - 1$ and the treewidth of H is the least treewidth taken over all decompositions. In many graph algorithms, treewidth is the key parameter that determines the complexity of the problem. However, for many problems whose underlying structure is naturally expressed in terms of hypergraphs the situation is different. The treewidth of a hypergraph is always at least as large as its *rank* (-1), i.e., the maximal size of an edge. Yet, many standard hypergraph problems can be tractable even with unbounded rank. To counteract this problem, *generalised hypertree width* (*ghw*) often takes the place of treewidth in these cases. The definition of *ghw* is also based on tree decompositions, with the

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).
Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;
Article No. 48; pp. 48:1–48:17


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

only difference being that the *ghw* of a decomposition is $\max_{u \in V(T)} \rho(\mathbf{B}(u))$, where $\rho(U)$ is the least number of edges required to cover $U \subseteq V(H)$. Parallel to treewidth in graphs, low *ghw* is a key criterion for tractability in the hypergraph setting. Prominent examples include the evaluation of conjunctive queries, database factorisation [19], winner determination in combinatorial auctions [7], and determining Nash Equilibria in strategic games [8].

Computation of *ghw* is computationally challenging. The problem is known to be paraNP-hard [13, 6] and W[2]-hard [9] in the parameterised setting[1] (see the discussion of related work below for details). This is shown by reduction from Set-Cover, which together with recent breakthrough results on the approximability of Set-Cover [16], also implies that there can be no FPT approximation algorithms for *ghw* under standard assumptions. In this paper, we introduce the first FPT algorithm for approximation of *ghw* in unbounded rank hypergraphs. Notably, this comes over 20 years after the parameter was first introduced [12]. As there can be no such algorithm in the general case, we instead consider the restriction of the cardinality of the intersections of any two edges. Formally, a $(2, d)$-hypergraph is a hypergraph where the intersection of any pair of edges has cardinality at most d. We will study the following problem f-ApproxGHW.

f-ApproxGHW

Input	$(2, d)$-hypergraph H, positive integer k
Parameters	k and d
Output	A tree decomposition of H with *ghw* at most $f(k, d)$, or **Reject**, in which case $ghw(H) > k$.

Let $\alpha(k, d)$ refer to the term $k(3k + d + 1)(2k - 1)$. Our main result is the following.

▶ **Theorem 1.** *$4\alpha(k, d)$-ApproxGHW is fixed-parameter tractable.*

Our algorithm follows classic ideas for FPT algorithms for treewidth but requires significant new developments. Most critically, we propose an FPT algorithm for computing approximate (A, B)-separators in $(2, d)$-hypergraph, i.e., set of vertices S such that sets of vertices A and B are not connected in H without S.

Related Work. Despite the close relationship between *ghw* and treewidth, there is a stark difference in the complexity of recognising the respective widths. While it is famously possible to decide $tw(\cdot) \leq k$ in fixed-parameter linear time [2], deciding $ghw(\cdot) \leq k$ is significantly harder. Intuitively, this is because techniques for efficiently deciding treewidth fundamentally rely on bounding the number of vertices in the bag, yet even α-acyclic hypergraphs (those with *ghw* 1) can require decompositions with arbitrarily large bags to achieve minimal *ghw*. In concrete terms, deciding $ghw(\cdot) \leq k$ has been shown to be NP-hard for all fixed $k > 1$ (or paraNP-hard in terms of parameterised complexity) [13, 6]. Additionally, deciding $ghw(\cdot) \leq k$ is known to be W[2]-hard by a reduction from Set Cover [9].

In response, significant effort has been invested in identifying conditions under which deciding $ghw(\cdot) \leq k$ is tractable [13, 11] for fixed k (i.e., the problem is in XP). Of particular note here is the observation that the problem is in XP if we restrict the problem to so-called (c, d)-hypergraphs, i.e., hypergraphs where any intersection of at least c edges has cardinality at most d. Notably, this coincides with the most general condition known to allow kernelization for Set-Cover [20].

[1] When discussing the parameterised complexity of deciding whether a width parameter is at most k, we always refer to the parameterisation by k if not specified otherwise.

As mentioned above, negative results for FPT approximation of Set-Cover apply also to approximating *ghw*. Nonetheless, some approximation results are known when looking beyond FPT. Importantly, *ghw* is 3-approximable in XP via the notion of (not generalised) *hypertree width* (*hw*) [12, 1]. Despite this improvement in the case of fixed k, deciding hypertree width is also W[2]-hard [9] by the same reduction from Set-Cover as *ghw*. *Fractional hypertree width fhw* generalises *ghw* in the sense that the width is determined by the *fractional cover number* of the bags [15]. This width notion is strictly more general than *ghw* and allows for cubic approximation on XP [18]. Recently, Razgon [21] proposed an FPT algorithm for constant factor approximation of *ghw* for hypergraphs of bounded rank. While we consider this to be an important conceptual step towards our result, it should be noted that for any hypergraph H it holds that $ghw(H) \leq tw(H) \cdot rank(H)$, i.e., assuming bounded rank tightly couples the problem to deciding treewidth.

Structure of the paper. Our main result combines on multiple novel combinatorial observations and FPT algorithms and the main body of this paper is therefore focused on presenting the main ideas and how these parts interact. Full proof details are provided in the appendix of the full version [17]. After basic technical preliminaries in Section 2, we give a high-level overview of the individual parts that lead to the main result in Section 3. After the initial overview, Section 4 presents the key algorithms formally, together with sketches of their correctness and complexity. Similarly, the main combinatorial ideas are discussed in Section 5. We discuss potential avenues for future research in Section 6.

2 Preliminaries

We will frequently write $[n]$ for the set $\{1, 2, \ldots, n\}$. We say that (possibly empty) pairwise disjoint sets $X_1, X_2, \ldots, X_n$ are a *weak partition* of set X if their union equals X. We assume familiarity with standard concepts of parameterised algorithms and refer to [3] for details. We use $poly(\cdot)$ to represent some polynomial function in the representation size of the argument.

A *hypergraph* H is a pair of sets $(V(H), E(H))$ where we call $V(H)$ the *vertices* of H and $E(H) \subseteq 2^{V(H)}$ the *(hyper)edges* of H. We assume throughout that H has no isolated vertices, i.e., vertices that are not in any edge. For $v \in V(H)$, define $I(v) := \{e \in E(H) \mid v \in e\}$, i.e., the set of edges incident to v. We say that H is a (c, d)-hypergraph if for any set $\{e_1, \ldots, e_c\} \subseteq E(H)$ of c edges, it holds that $|\bigcap_i^c e_i| \leq d$. We will refer to the set of (maximal) connected components of a hypergraph H as $CComp(H)$. The *induced subhypergraph* of H induced by $U \subseteq V(H)$ is the hypergraph H' with $V(H') = U$ and $E(H') = \{e \cap U \mid e \in E(H)\} \setminus \emptyset$. We use the notation $H[U]$ to mean the induced subhypergraph of H induced by U. For tree T and $v \in V(T)$ we sometimes write $T - v$ to mean the subgraph of T obtained by deleting v and its incident edges. Let $A, B \subseteq V(H)$. An (A, B)-separator is a set $S \subseteq V(H)$ such that there is no path from an $a \in A \setminus S$ to a $b \in B \setminus S$ in $H[V(H) \setminus S]$. For $e \in E(H)$, $U_1, \ldots, U_n \subseteq V(H)$ we say that e *touches* $U_1, \ldots, U_n$ if $e \cap U_i \neq \emptyset$ for all $i \in [n]$.

An *edge cover* μ for $U \subseteq V(H)$ is a subset of $E(H)$ such that $U \subseteq \bigcup \mu$. We sometimes refer to the cardinality of an edge cover as its *weight*. The *edge cover number* $\rho(U)$ for set $U \subseteq V(H)$ is the minimal weight over all edge covers for U. We sometimes say that μ is an edge cover of H to mean an edge cover of $V(H)$. Similarly, we use $\rho(H)$ instead of $\rho(V(H))$. A set of edges $E' \subseteq E(H)$ is *ρ-stable* if E' is a minimal weight cover for $\bigcup E'$.

A set of sets $S_1, \ldots$ can naturally be interpreted as a hypergraph, by considering each set S_i as an edge. In that light, it is clear that deciding $\rho(H) \leq k$ is precisely the same as deciding whether a set system admits a set cover of size k.

▶ **Proposition 2** ([4]). *There is an FPT algorithm parameterised by $k+c+d$ that decides for a given (c,d)-hypergraphs H and $k \geq 1$ whether $\rho(H) \leq k$.*

A *tree decomposition* (TD) of hypergraph H is a pair $(T, \mathbf{B})$ where T is a tree and $\mathbf{B} : V(T) \to 2^{V(H)}$ labels each node of T with its so-called *bag*, such that the following hold:

(i) for each $e \in E(H)$, there is a $u \in V(T)$ such that $e \subseteq \mathbf{B}(u)$, and

(ii) for each $v \in V(H)$, the set $\{u \in V(U) \mid v \in \mathbf{B}(u)\}$ induces a non-empty subtree of T.

We refer to the first property as the *containment property* and to the second as the *connectedness condition*. For a set $U \subseteq V(T)$ we use $\mathbf{B}(U)$ as a shorthand for $\bigcup_{u \in U} \mathbf{B}(u)$, i.e., all the vertices that occur in bags of nodes in U. Similarly, for subtree T' of T, we sometimes use $\mathbf{B}(T')$ instead of $\mathbf{B}(V(T'))$. The *generalised hypertree width* (*ghw*) of a tree decomposition is $\max_{u \in V(T)} \rho(\mathbf{B}(u))$, and the generalised hypertree width of H (we write $ghw(H)$) is the minimal *ghw* over all tree decompositions for H. It will be important to remember at various points that *ghw* is monotone under taking induced subhypergraphs, i.e., $ghw(H[U]) \leq ghw(H)$ for all $U \subseteq V(H)$.

3 High-level Overview of Algorithm

The algorithm for $4\alpha(k,d)$-ApproxGHW is based on a standard approach for treewidth computation [22]. However, this approach is in fact applied for a tree decomposition *compression* rather than approximation from scratch. In other words, the actual problem being solved is the following one.

Compress

Input	A $(2,d)$-hypergraph H, integer k, a TD $(T, \mathbf{B})$ of H with *ghw* $4\alpha(k,d)+1$, $W \subseteq V(H)$ with $\rho(W) \leq 3\alpha(k,d)$
Parameters	k and d
Output	A tree decomposition $(T^*, \mathbf{B}^*)$ of H with *ghw* at most $4\alpha(k,d)$ such that $W \subseteq \mathbf{B}^*(u)$ for some $u \in V(T^*)$, or **Reject**, in which case $ghw(H) > k$.

▶ **Theorem 3.** *Compress is fixed-parameter tractable.*

One natural question is how the Compress being FPT implies $4\alpha(k,d)$-ApproxGHW. This is done through the use of *iterative compression*, a well known methodology for the design of FPT algorithms. The resulting algorithm for Theorem 1 is presented in Algorithm 1. In particular, we let $V(H) = \{v_1, \ldots, v_n\}$, set $V_i = \{v_1, \ldots, v_i\}$ and $H_i = H[V_i]$ and solve the $4\alpha(k,d)$-ApproxGHW for graphs $H_1, \ldots, H_n$. If some intermediate H_i is rejected, the whole H can be rejected. Otherwise, the application to H_i results in a tree decomposition $(T_i, \mathbf{B}_i)$ of *ghw* at most $4\alpha(k,d)$. Add v_{i+1} to each bag of $(T_i, \mathbf{B}_i)$. If the *ghw* of the resulting decomposition is still at most $4\alpha(k,d)$ simply move on to the next iteration. Otherwise, apply the algorithm for Compress rejecting if the algorithm rejects and moving to the next iteration if a compressed tree decomposition is returned.

Let us turn our attention to the algorithm for Compress. A central ingredient of the algorithm [22] considers a set S of size $O(k)$ goes through all partitions of S into two balanced subsets and for each such a partition checks existence of a small separator. However, in our setting the set S can contain arbitrarily many vertices, as long as it can be covered by a bounded number of hyperedges. The following statement provides us with an appropriate variant of the classic result for treewidth that is applicable to our setting.

Algorithm 1 An FPT algorithm for $4\alpha(k,d)$-ApproxGHW.

Input : $(2,d)$-hypergraph H, positive integer k
$\{v_1,\dots,v_n\} \leftarrow V(H)$
Let T_1 be the tree with a single node r
Let $\mathbf{B}_1$ be the function $r \mapsto \{v_1\}$
for $1 < i \leq n$ **do**
  $V_i \leftarrow \{v_1,\dots,v_i\}$
  $H_i \leftarrow H[V_i]$
  $T_i \leftarrow T_{i-1}$
  $\mathbf{B}_i \leftarrow \{t \mapsto \mathbf{B}_{i-1}(t) \cup \{v_i\} \mid t \in T_i\}$
  if $ghw((T_i,\mathbf{B}_i)) > 4\alpha(k,d)$ **then**
    $X \leftarrow Compress(H_i,k,(T_i,\mathbf{B}_i),\emptyset)$
    if X *is* ***Reject*** **then**
      return ***Reject***
    $(T_i,B_i) \leftarrow X$
return $(T_n,\mathbf{B}_n)$

▶ **Theorem 4.** *Let H be a hypergraph with $ghw(H) \leq k$ and let $E' \subseteq E(H)$. Then there exists a weak partition of E' into three sets E'_0, E'_1, E'_2 such that*
1. *there is a $(\bigcup E'_1, \bigcup E'_2)$-separator S such that $\bigcup E'_0 \subseteq S$ and $\rho(S) \leq k$,*
2. *$|E'_1| \leq \frac{2}{3}|E'|$, and $|E'_2| \leq \frac{2}{3}|E'|$.*

Theorem 4 allows us to consider all partitions of a small set of hyperedges covering the given potentially large set of vertices thus guaranteeing an FPT upper bound for the number of such partitions.

The other obstacle in upgrading the result [22] is that a balanced separator is no longer required to be small but rather to have a small edge cover number. In order to compute such a separator we will need a witnessing tree decomposition of H of a small *ghw*. This is also the reason why we employ iterative compression rather than providing a direct algorithm for approximation. However, even in presence of the tree decomposition, we were still unable to design a 'neat' algorithm that would either produce an (approximately) small separator or reject, implying that a small separator does not exist. Instead, we propose an algorithm for the following problem with a *nuanced* reject that is still suitable for our purposes.

ApproxSep

Input	A $(2,d)$-hypergraph H, sets $A_1, A_2 \subseteq V(H)$, TD $(T,\mathbf{B})$ of H with *ghw* p, integers $0 \leq k_0 \leq k \leq p$
Parameters	p and d
Output	An (A_1,A_2)-separator with edge cover number at most $(3k+d+1)(2k-1)k_0$, or **Reject**, in which case there either exists no (A_1,A_2)-separator with edge cover number at most k_0 or $ghw(H) > k$.

▶ **Theorem 5.** *ApproxSep is fixed-parameter tractable.*

We postpone to the next section a more detailed consideration of the algorithm for ApproxSep. In the rest of this section we discuss the criterion for large GHW used by the algorithm and the context in which the critetion is checked. For this purpose, we will require

some technical definitions. For hypergraph H, let us call $U \subseteq V(H)$ a *subedge* (of H) if there is $e \in E(H)$ such that $U \subseteq e$. We say that two subedges U_1, U_2 are *incompatible* if their union $U_1 \cup U_2$ is not a subedge.

▶ **Definition 6.** *An* (a, b) subedge hypergrid *(or* (a, b)*-shyg) consists of pairwise incompatible subedges* $U_1, \ldots, U_a, S_1, \ldots, S_b$ *such that:*

1. $U_1, \ldots, U_a$ *are pairwise disjoint, and*
2. *for each* $j \in [b]$, S_j *touches* $U_1, \ldots, U_a$.

Throughout this paper we will be interested in a specific dimension of subedge hypergrid. Namely for deciding width k in $(2, d)$-hypergraphs, we will be interested in the existence of $(3k + d + 1, \xi(k, d))$ subedge hypergrids, where ξ is a function in $O((kd)^d)$ (refer to Section 5 for details). We will denote the set of all subedge grids of H of this dimension by $\mathbf{S}_{k,d}(H)$. The reason we care about these subedge hypergrids in particular is that their existence is a sufficient condition for high *ghw*.

▶ **Theorem 7.** *Let* H *be a* $(2, d)$*-hypergraph such that* $\mathbf{S}_{k,d}(H) \neq \emptyset$. *Then* $ghw(H) > k$.

In fact, the algorithm for the ApproxSep problem either constructs a required separator or discovers that $\mathbf{S}_{k,d}(H) \neq \emptyset$. More precisely, the identification of an element of $\mathbf{S}_{k,d}(H)$ takes place within the procedure described in Theorem 9 below. The central part the algorithm for ApproxSep is to pick a vertex $t \in T$ (recall that $(T, \mathbf{B})$ is the input tree decomposition for H) and then guess a set $W \subseteq \mathbf{B}(t)$ that shall be part of the separator being constructed. Technically, the guessing means a loop exploring a family of subsets of $\mathbf{B}(t)$. This family must be of an FPT size and the edge cover of each element of the family must not be too large compared to k_0 This idea is formalised in the notion of *gap cover approximator* formally defined below.

▶ **Definition 8.** *For hypergraph* H *and set* $U \subseteq V(H)$ *a* (β, γ)-gap cover approximator *(for* U*) is a set* $\mathbf{X} \subseteq 2^{V(H)}$ *such that*

- **(i)** *For each* $X \in \mathbf{X}$, $\rho(X) \leq \beta$.
- **(ii)** *For each* $U' \subseteq U$ *with* $\rho(U') \leq \gamma$, *there is a* $X \in \mathbf{X}$ *such that* $U' \subseteq X$.

In order to produce the desired gap cover approximator, the algorithm solving the ApproxSep problem runs a function *GapCoverApprox*. The function computes either an gap cover approximator or an element of $\mathbf{S}_{k,d}(H)$ and, in the latter case, rejects. In particular, when the algorithm rejects, we know (implicitly) that $\mathbf{S}_{k,d}(H) \neq \emptyset$, which in turn guarantees that *ghw* is greater than k in this case and the rejection can be propagated to the top-level. A formal description of the behaviour of *GapCoverApprox* is provided below.

▶ **Theorem 9.** *There is an algorithm* $GapCoverApprox(H, U, p, k, k_0)$ *whose input is a* $(2, d)$*-hypergraph* H, $U \subseteq V(H)$ *with* $\rho(U) \leq p$, *and integers* $k_0 \leq k \leq p$. *The algorithm returns a* $((3k + d + 1)k_0, k_0)$*-gap cover approximator of* U *or* ***Reject****, in which case it is guaranteed that* $ghw(H) > k$. *The algorithm is in* FPT *when parameterised in* p *and* d.

4 Algorithmic Details

In this section we sketch proofs of Theorems 3 and 5. In particular, we provide pseudocodes of the corresponding algorithms and intuitive justification of their correctness and FPT membership. Algorithm 2 uses as a subroutine the algorithm *AppSep*, which is discussed afterwards in Section 4.2.

4.1 An FPT Algorithm for the Compression Step (Theorem 3)

To prove Theorem 3 we define Algorithm 2, prove that it is correct, and that the algorithm works FPT time. In general principle the algorithm follows similar ideas to previous algorithms for checking *ghw* (e.g., [5]) in that, at each stage, we separate the problem into subproblems for each connected component, and recurse. The set W provides an interface to how the subproblem connects to the rest of the decomposition. By guaranteeing that W is covered in the root of the decomposition for the subproblem (line 8), we guarantee that the decompositions for all the subproblems can be assembled into a decomposition for the parent call in lines 8 to 22. (see also [10] where a similar idea is formalised in terms of *extended hypergraphs* in the context of checking plain hypertree width).

We move on to giving an overview of the argument for the runtime and correctness of the algorithm. For the overall time complexity of the algorithm, we first observe that a single recursive application of the algorithm runs in FPT time. The search for E_W in line 1 is FPT by using Proposition 2 to find a cover for W (the procedure from the proposition is constructive). If the produced cover is smaller than $3\alpha(k,d)+1$ we can incrementally increase the size of the cover by searching for covers for $W' \supseteq W$ created by adding a vertex outside of the cover to W. For line 2 we can naively iterate through all possible partitions of E_W intro three sets and call *AppSep*, which itself is in FPT by Theorem 5. For line 7 we note again that testing ρ is FPT by Proposition 2. From line 8 onward, except for the recursion, the algorithm performs straightforward manipulations of sets and hypergraphs that are of no deeper interest to our time bound.

Next, we observe that the number of recursive applications is, in fact, polynomial in H. We naturally organise recursive applications into a successors (recursion) tree and upper bound the number of nodes of the tree by the product of the height of the tree and the number of leaves. We observe that the height of the tree is at most $|V(H)|$. For this we prove two auxiliary statements. The first is that for X, as computed in line 6, $H \setminus X$ has at least two connected components. The second, immediately following from the first one is that for each H_i, created in line 10, $|V(H_i)| < |V(H)|$. Thus it follows that the number of vertices of the input hypergraph decreases as we go down the recursion tree thus implying the upper bound on the height of the tree.

Additionally, we prove that the number of leaves is no larger than $\rho(H)^2$. The main part of this proof is an induction for the case where the number of sets $U_1, \dots, U_q$ obtained at line 7 is at least 2. In particular, we notice that since $\rho(U_i) \geq 3\rho(X)$ for each $i \in [q]$ and since $\rho(\bigcup_{i=1}^{q} U_i) = \sum_{i=1}^{q} \rho(U_i)$, it holds that

$$\rho(U_i \cup X)^2 \leq (\rho(U_i) + \rho(X))^2 \leq (\sum_{i=1}^{q} \rho(U_i))^2 = \rho(\bigcup_{i=1}^{q} U_i)^2 \leq \rho(H)^2.$$

To prove correctness of the **Reject** output, we observe that return of **Reject** by the whole algorithm is triggered by return of **Reject** on line 4, failure to find an appropriate weak partition, or by rejection in one of its recursive applications. By Theorem 4 and the ρ-stability of E', the **Reject** on line 4 implies that either H, or one of its induced subgraphs have *ghw* greater than k. In the latter case, of course also $ghw(H) > k$.

Finally, the two main aspects of correctness of the non-rejection output are the upper bound on the *ghw* of the resulting tree decomposition and that the properties of the tree decomposition are not lost by the 'gluing' procedure as specified in lines 8-22 of the algorithm. The requirement that E_W must cover W is essential for ensuring that the properties of the tree decomposition are not destroyed by the gluing. Intuitively, the parameter W_i in the recursion on Line 13 represents the connection of the component H_i with the rest of the decomposition.

Algorithm 2 The algorithm $Compress(H, k, (T, \mathbf{B}), W)$.

Input : $(2,d)$-hypergraph H, a positive integer k, a TD $(T, \mathbf{B})$ of H with ghw $4\alpha(k,d)+1$, $W \subseteq V(H)$ wih $\rho(W) \leq 3\alpha(k,d)$

$E_W \leftarrow$ any ρ-stable subset of $E(H)$ covering W of cardinality $3\alpha(k,d)+1$
Find a weak partition E_0, E_1, E_2 of E_W s.t. $\rho(E_0) \leq k$, $\rho(E_1), \rho(E_2) \leq 2\alpha(k,d)$, and $AppSep(H, \bigcup E_1, \bigcup E_2, (T, \mathbf{B}), k, k, p, V(T))$ does not return **Reject**
if *no such weak partition exists* **then**
    return ***Reject***
else
    $X \leftarrow AppSep(H, \bigcup E_1, \bigcup E_2, (T, \mathbf{B}), k, k, p, V(T))$
$U_1, \ldots, U_q \leftarrow \{C \in CComps(H \setminus X) \mid \rho(C \cup X) > 4\alpha(k,d)\}$
Let T^* be a tree with a new node r and $B^*(r) = W \cup X$
for $i \in [q]$ **do**
    $H_i \leftarrow H[U_i \cup X]$
    $W_i \leftarrow (\bigcup E_W \cap U_i) \cup X$
    $\mathbf{B}_i \leftarrow \{t \mapsto \mathbf{B}(t) \cap (U_i \cup X) \mid t \in V(T)\}$
    $O_i \leftarrow Compress(H_i, k, (T, \mathbf{B}_i), W_i)$
    if O_i *is* ***Reject*** **then**
        return ***Reject***
    else
        $(T'_i, \mathbf{B'}_i) \leftarrow O_i$
        $t_i \leftarrow$ a node u of T'_i s.t. $W_i \subseteq \mathbf{B'}_i(u)$
        Add T'_i to T^* by making t_i a neighbour of r and let $\mathbf{B}^*(u) = \mathbf{B'}_i(u)$ for all $u \in T'_i$.
for $U \in CComps(H \setminus X)$ *where* $\rho(U \cup X) \leq 4\alpha(k,d)$ **do**
    Add new node u as a neighbour of r to T^*.
    Set $\mathbf{B}^*(u) = U \cup X$.
return $(T^*, \mathbf{B}^*)$

4.2 Finding Approximate Separators in FPT (Theorem 5)

The proof of Theorem 5 requires significant extension of notation. First, for a tree decomposition $(T, \mathbf{B})$ of a hypergraph and $X \subseteq V(T)$, we denote by $ghw(T, \mathbf{B}, X)$ the maximum of $\rho(\mathbf{B}(t))$ among $t \in X$. We are looking for a separator subject to several constraints. Repeating these constraints every time we refer to a separator is somewhat distracting and we therefore define the set of separators that we need to consider for this overview. Define $sep(H, A, B, k_0, (T, \mathbf{B}), X)$ as the set of all (A,B)-separators W of H with $\rho(W) \leq k_0$ and $W \subseteq \mathbf{B}(X)$ where $(T, \mathbf{B})$ is a tree decomposition of H and $X \subseteq V(T)$.

The following theorem is a generalisation of Theorem 5.

▶ **Theorem 10.** *There is an algorithm $AppSep(H, A, B, (T, \mathbf{B}), k_0, k, p, X)$ whose input is a $(2,d)$-hypergraph H, $A, B \subseteq V(H)$, three positive integers $k_0 \leq k \leq p$, a tree decomposition $(T, \mathbf{B})$ of H and $X \subseteq V(T)$ such that all the elements of $V(T) \setminus X$ are leaves and $ghw(T, \mathbf{B}, X) \leq p$. The algorithm either returns an element of $sep(H, A, B, (3k+d+1)(2k-1)k_0, (T, \mathbf{B}), X)$ or* ***Reject****. In the latter case, it is guaranteed that either $ghw(H) > k$ or $sep(H, A, B, k_0, (T, \mathbf{B}), X) = \emptyset$*

Clearly, Theorem 10 implies Theorem 5 by setting $X = V(T)$. The reason we need this extra parameter is that in the recursive applications of *AppSep* some bags may have the edge cover number larger than p, so we keep track of the set of nodes whose bags are 'small'.

To present the pseudocode, we define a specific choice of a subtree of the given tree. Let T be a tree, $t \in V(T)$, $Y \subset V(T)$. Then $T_{t,Y}$ is the subtree of T which is the union of all paths starting from t whose second vertex belongs to Y. The notion of $T_{t,Y}$ naturally extends to subsets of $V(T)$ and to tree decompositions where T is the underlying tree. In particular, for $X \subseteq V(T)$, we denote $X \cap V(T_{t,Y})$ by $X_{t,Y}$. Next, if $(T, \mathbf{B})$ is a tree decomposition of H then by denote by $\mathbf{B}_{t,Y}$ the restriction to $\mathbf{B}$ to $V(T_{t,Y})$ and by $H_{t,Y}$ the graph $H[\mathbf{B}(V(T_{t,Y}))]$.

We also introduce a variant $T^+_{t,Y}$ of $T_{t,Y}$ which will be needed for recursive applications of *AppSep*. The tree $T^+_{t,Y}$ is obtained from $T_{t,Y}$ by introducing a new node r and making it adjacent to t. The function $\mathbf{B}^+_{t,Y}$ is obtained from $\mathbf{B}_{t,Y}$ by setting $\mathbf{B}^+_{t,Y}(r) = \bigcup_{t' \in V(T) \setminus V(T_{t,Y})} \mathbf{B}(t')$. One final notational convention concerns adjusting a tree decomposition $(T, \mathbf{B})$ of H in case a set $W \subseteq V(H)$ is removed from H. In this case we set $\mathbf{B}^{-W}(t') = \mathbf{B}(t') \setminus W$ for each $t' \in V(T)$.

The following statement is important for verifying that these recursive applications are well-formed.

▶ **Theorem 11.** *Let H be a hypergraph, $(T, \mathbf{B})$ a TD of H, $t \in V(T)$, $Y \subseteq N_T(t)$. Then $(T_{t,Y}, \mathbf{B}_{t,Y})$ is a TD of $H_{t,Y}$, $(T^+_{t,Y}, \mathbf{B}^+_{t,Y})$ is a TD of H and $(T, \mathbf{B}^{-W})$ is a TD of $H \setminus W$. Moreover, let $X \subseteq V(T)$ such that all the vertices of $V(T) \setminus X$ are leaves of T. Then all the vertices of $V(T_{t,Y}) \setminus X_{t,Y}$ are leaves of $T_{t,Y}$.*

The algorithm (roughly speaking) chooses a vertex $t \in V(T)$, partitions $N(t)$ into Y_1 and Y_2 and applies recursively to H_{t,Y_1} and H_{t,Y_2}. However, the triple (t, Y_1, Y_2) is chosen not arbitrarily but in a way that both X_{t,Y_1} and X_{t,Y_2} are significantly smaller than X. The possibility of such a choice is guaranteed by the following theorem.

▶ **Theorem 12.** *Let T be a tree, $X \subseteq V(T)$ such that all $|X| \geq 3$ and all the vertices of $V(T) \setminus X$ are leaves. Then there is $t \in X$ with $deg_{T[X]}(t) \geq 2$ and a partition Y_1, Y_2 on $N_T(t)$ so that for each $i \in \{1, 2\}$, $|X_{t,Y_i}| \leq 3/4|X|$. Moreover, the triple (t, Y_1, Y_2) can be computed in a polynomial time.*

Theorem 12 can be seen as a variant of a classical statement that a rooted tree has a descendant rooting a subtree with the number of leaves between one third to two third of the total number of leaves. The proof is based on a similar argument of picking a root and gradually descending towards a 'large' subtree until the desired triple is found.

For the validity of *AppSep*, it is important to note that each X_{t,Y_i} preserves for T_{t,Y_i} the invariant that all the $V(T_{t,Y_i}) \setminus X_{t,Y_i}$ are leaves of T_{t,Y_i}. We are almost ready to consider the pseudocode, it only remains to identify auxiliary functions. In particular *GetBalVert*(T, X) is a polynomial time algorithm as specified in Theorem 12. Also recall that *GapCoverApprox* is an FPT algorithm constructing a $((3k + d + 1)k_0, k_0)$-gap cover approximator in the way specified by Theorem 9.

The pseudocode of *AppSep* is presented in Algorithm 1. For the sake of readability, we make two notational conventions. First, since parameters p and k do not change when passed through recursive calls, we consider them fixed and do not mention them as part of the input when recursing. Second, we move consideration of the case with $|X| \leq 2$ into a separate function *SmallSep* provided in Algorithm 3 and use it as an auxiliary function in Algorithm 4. Here the idea is straightforward, we naively test for all gap cover approximators whether they are (A, B)-separators.

Algorithm 3 The algorithm $SmallSep(H, A, B, p, k, k_0, (T, \mathbf{B}), X)$.

Input : $(2, d)$-hypergraph H, $A, B \subseteq V(H)$, positive integers $k_0 \leq k \leq p$, a TD $(T, \mathbf{B})$ of H, $X \subseteq V(T)$, $|X| \leq 2$ all the elements of $V(T) \setminus X$ are leaves

1 **if** $|X| = 1$ **then**
2 $\{t\} \leftarrow X$
3 $Sets \leftarrow GapCoverApprox(H, \mathbf{B}(t), p, k, k_0)$
4 **else**
5 $\{t_1, t_2\} \leftarrow X$
6 $Sets \leftarrow GapCoverApprox(H, \mathbf{B}(t_1) \cup \mathbf{B}(t_2), p, k, k_0)$
7 **if** *Sets is* ***Reject*** **then**
8 **return** ***Reject***
9 **for** $W \in Sets$ **do**
10 **if** W *is an* (A, B)*-separator* **then**
11 **return** U
12 **return** ***Reject***

The general case in *AppSep* is considerably more complex. Intuitively, the algorithm searches for separators in small local parts of the tree decomposition by searching through the output of *GapCoverApprox* called on the component of that local part of the decomposition (Lines 1 and 2 of the algorithm). In general this is of course not sufficient to find (A, B)-separators. What we do instead is to try and find partial separators that ultimately combine into a single (A, B)-separator. To that end we split the tree decomposition into two decompositions in a balanced fashion (Line 3). The two cases handled from Lines 4 to 9 cover the special case where the search is propagated to one part of the tree decomposition, while the body of the loop at Line 13 considers (roughly speaking) all possibilities of splitting up the separator over both parts.

The first step of proving Theorem 10 is to prove the FPT runtime of *AppSep*. We first observe that a single application of *AppSep* takes FPT time. The efficiency of *GetBalVert* and *GapCoverApprox* has been discussed above. *SmallSep* is effectively a loop over the output of *GapCoverApprox* with polynomial time spent per element. Note that since *GapCoverApprox* is computed in FPT time we also obtain a corresponding bound on the size of the $((3k + d + 1)k_0, k_0)$-gap cover approximator to iterate over. Finally, in the loop of Line 13, we only need to observe that the number of connected components of $\mathbf{B}(t) \setminus W$ is at most p as otherwise the edge cover number of $\mathbf{B}(t)$ is greater than p. Hence, the number of partitions C_1, C_2 considered in the loop is $O(2^p)$.

Next, we need to demonstrate that the number of recursive applications of *AppSep* is FPT. We present the number of applications as a recursive function $F(k_0, m)$ where $m = |X|$. If $m \leq 2$ then there is only a single application through running *SmallSep*. Otherwise, there is one recursive application at Line 4 and one at Line 7 where the first parameters remains the same and the second parameter is at most $3/4m$ (by selection of t, Y_1, Y_2). Additionally, there are also the recursive applications in Lines 14 and 15 where the first parameter is at most $k_0 - 1$ and the second parameter is at most $3/4m$. As a result, we obtain a recursive formula $F(k_0, m) \leq 2F(k_0, 3/4m) + g(p)F(k_0 - 1, 3/4m)$. We note that this function can be bounded above by a fixed-parameter cubic function (see Lemma 43 in the full version appendix for details) thus establishing the FPT runtime of *AppSep*.

Algorithm 4 The algorithm $AppSep(H, A, B, (T, \mathbf{B}), k_0, k, p, X)$.

Input : $(2,d)$-hypergraph H, $A, B \subseteq V(H)$, a TD (T, B) of H, positive integers $0 \leq k_0 \leq k \leq p$, $X \subseteq V(T)$ s.t. all the elements of $V(T) \setminus X$ are leaves

1 **if** $|X| \leq 2$ **then**
2 **return** $SmallSep(H, A, B, p, k, k_0, (T, \mathbf{B}), X)$
3 $(t, Y_1, Y_2) \leftarrow GetBalVert(T, X)$
4 $Out \leftarrow AppSep(H, A, B, (T^+_{t,Y_1}, \mathbf{B}^+_{t,Y_1}), k_0, X_{t,Y_1})$
5 **if** *Out is not* ***Reject*** **then**
6 **return** *Out*
7 $Out \leftarrow AppSep(H, A, B, (T^+_{t,Y_2}, \mathbf{B}^+_{t,Y_2}), k_0, X_{t,Y_2})$
8 **if** *Out is not* ***Reject*** **then**
9 **return** *Out*
10 $Sets \leftarrow GapCoverApprox(H, \mathbf{B}(t), p, k, k_0)$
11 **if** *Sets is* ***Reject*** **then**
12 **return** ***Reject***
13 **for** *each* $W \in Sets$, *each* $k_1, k_2 > 0$ *s.t.* $k_1 + k_2 \leq k_0$, *and each weak partition* C_1, C_2 *of* $\mathbf{B}(t) \setminus W$ *into unions of connected components of* $H[\mathbf{B}(t) \setminus W]$ **do**
14 $Out_1 \leftarrow AppSep(H_{t,Y_1} \setminus W, (A_{t,Y_1} \cup C_1) \setminus W, (B_{t,Y_1} \cup C_1) \setminus W, (T_{t,Y_1}, \mathbf{B}^{-W}_{t,Y_1}), k_1, X_{t,Y_1})$
15 $Out_2 \leftarrow AppSep(H_{t,Y_2} \setminus W, (A_{t,Y_2} \cup C_2) \setminus W, (B_{t,Y_2} \cup C_2) \setminus W, (T_{t,Y_2}, \mathbf{B}^{-W}_{t,Y_2}), k_2, X_{t,Y_2})$
16 **if** *neither of* Out_1, Out_2 *is* ***Reject*** **then**
17 **return** $Out_1 \cup Out_2 \cup W$
18 **return** ***Reject***

Next, we need to demonstrate correctness of the non-rejection output. It is straightforward to see by construction that the returned set S is a subset of $\mathbf{B}(X)$: ultimately, the set S is comprised of unions of outputs of $GapCoverApprox(H, U, \dots)$, either directly in Line 10, or indirectly via *SmallSep*. In the first case, the returned sets are a subset of $\mathbf{B}(t)$, where $t \in X$ by definition of *GetBalVert*. In the second case, $U \subseteq \mathbf{B}(X_{t,Y_i})$ for $i = 1$ or $i = 2$ by definition of *SmallSep*. By definition, both X_{t,Y_i} are subsets of X and thus $\mathbf{B}(X_{t,Y_i})$ is a subset of $\mathbf{B}(X)$. Since the recursion always restricts parameter X to either X_{t,Y_1} or X_{t,Y_2} the inductive application of this observation is immediate.

We need to show S is an (A, B)-separator and that its edge cover number is within a specified upper bound. Both claims are established by induction. The main part of proving that S is an (A, B)-separator is showing that if S as returned on Line 17 then it is an (A, B)-separator. This follows from the induction assumption applied to Out_1 and Out_2 and the following statement.

▶ **Lemma 13.** *Let H be a hypergraph, $V_1, V_2 \subseteq V(H)$ be such that $V_1 \cup V_2 = V(H)$ and $Y = V_1 \cap V_2$ is a (V_1, V_2)-separator. Let $W \subseteq Y$ and let C_1, C_2 be a weak partition of $Y \setminus W$. Let $H_1 = H[V_1 \setminus W]$ and $H_2 = H[V_2 \setminus W]$. Let $A, B \subseteq V(H)$. For each $i \in \{1, 2\}$ let $A_i = (A \cap V(H_i)) \cup C_i$, let $B_i = (B \cap V(H_i)) \cup C_2$, and let W_i be an (A_i, B_i)-separator of H_i. Then $W_1 \cup W_2 \cup W$ is an (A, B)-separator of H.*

To prove that the size of the output S of *AppSep* matches the required upper bound, we observe that the output is the union of several sets $S_1, \ldots, S_q$ each of which is in a family returned by an application of *GapCoverApprox*. This guarantees that $\rho(S_i) \leq (3k+d+1)k_0$. It only remains to show that $q \leq 2k_0 - 1$ (q is the number of sets $S_1, \ldots, S_q$ whose union make up S). To this end we observe that the recursive applications invoking *GapCoverApprox* can be naturally organised into a recursion tree where each node has two children, accounting for the recursive applications in Lines 14 and 15 (the applications on Lines 4 and 7 are not relevant since there is no invocation of *GapCoverApprox* associated with them). Then q is simply the number of nodes of the tree. By a simple induction we observe that if k_1 and k_2 are the numbers as obtained in Line 13 then $k_1 + k_2 \leq k_0$ and the number of nodes rooted by children of the tree is at most $2k_1 - 1$ and $2k_2 - 1$, respectively. Hence, the total number of nodes, accounting for the root, is at most $(2k_1-1)+(2k_2-1)+1 \leq 2(k_1+k_2)-1 \leq 2k_0-1$ as required. For the correctness of the **Reject** output we first recursively define the **Reject** *triggered* by *GapCoverApprox*. This happens when *AppSep* runs *SmallSep* and the latter returns **Reject** in Lines 5 or 10 of Algorithm 3, or **Reject** is returned in Line 12 of Algorithm 4, or when **Reject** is returned in Line 18 of Algorithm 4 and one of recursive applications leading to this output returns **Reject** triggered by *GapCoverApprox*. By inductive application of Theorem 9 we observe that **Reject** triggered by *GapCoverApprox* implies that the *ghw* of some induced subgraph of H (and hence of H itself) is greater than k.

Finally, we demonstrate that if the **Reject** output is not triggered by *GapCoverApprox* then $sep(H, A, B, k_0, (T, \mathbf{B}), X) = \emptyset$. First, we assume that $|X| \leq 2$ and demonstrate this for Algorithm 3. It follows from the description that, in the considered case, no element of *Sets* is an (A, B) separator. As each $W' \subseteq \mathbf{B}(X)$ with $\rho(W') \leq k_0$ is a subset of some element of *Sets*, it follows that no such W' is an (A, B)-separator of H. In the case where $|X| \geq 3$, a **Reject** not inherited from *GapCoverApprox* can only be returned in Line 18 of Algorithm 1. This, in particular, requires **Reject** to be returned by recursive applications in Lines 4 or 7. By the induction assumption, $sep(H, A, B, k_0, (T^+_{t,Y_i}, \mathbf{B}^+_{t,Y_i}), X_{t,Y_i}) = \emptyset$ for each $i \in \{1, 2\}$. This means that if there is $W^* \in sep(H, A, B, k_0, (T, \mathbf{B}), X)$ then W^* is *not* a subset of $\mathbf{B}(X_{t,Y_1})$ nor of $\mathbf{B}(X_{t,Y_2})$. We conclude that by the induction assumption, such a W^* would cause a non-rejection output in one of iterations of the loop in Line 13. Since the algorithm passes through to Line 18, such an iteration does not happen so we conclude that such a W^* does not exist.

5 Combinatorial Statements

In this section we prove Theorem 4 and sketch the proofs for Theorems 7 and 9. While Theorem 9 also refers to the existence of an algorithm, we consider the nature of the theorem to be purely combinatorial. The resulting algorithm is simply a naive enumeration of all possibilities of combining certain sets.

5.1 Theorem 4

For Theorem 4 we can make use of a result from the literature and prove the statement in full here. We first recall key terminology from Adler et al. [1], who proved the result that we will use. For a set $E' \subseteq E(H)$, and $C \subseteq V(H)$ define $ext(C, E') := \{e \in E' \mid e \cap C \neq \emptyset\}$. We say that C is E'-*big* if $|ext(C, E')| > \frac{|E'|}{2}$. A set $E' \subseteq E(H)$ is k-*hyperlinked* if for every set $S \subseteq E(H)$ with $|S| < k$, $H \setminus \bigcup S$ has an E'-big connected component. The hyperlinkedness $hlink(H)$ of H is the maximal k such that H contains a k-hyperlinked set. From another perspective, if $hlink(H) \leq k$, then for any set $E' \subseteq E(H)$, there is an $S \subseteq E(H)$ with $|S| \leq k$ such that no connected component of $H \setminus \bigcup S$ is E'-big. Adler et al. [1] showed the following.

▶ **Proposition 14** ([1]). *For every hypergraph H, $hlink(H) \leq ghw(H)$.*

Proof of Theorem 4. By assumption and Proposition 14, we have $hlink(H) \leq k$. Then for E', there is a set of k edges $S \subseteq E(H)$ such that no connected component of $H \setminus S$ is E'-big. Let $C_1, \ldots, C_\ell$ be the connected components of $H \setminus \bigcup S$. First, observe that $ext(C_i, E') \cap ext(C_j, E') = \emptyset$ for any distinct $i, j \in [\ell]$. Suppose, w.l.o.g., that $|ext(C_i, E')| \geq |ext(C_{i+1}, E')|$ for $i \in [\ell - 1]$. Let m be the highest integer such that $\sum_{i=1}^{m} |ext(C_i, E')| < \frac{2}{3}|E'|$. Such an $m \geq 1$ always exists because no component is E'-big. We claim that $E'_0 = S \cap E'$, $E'_1 = \bigcup_{i=1}^{m} ext(C_i, E')$, and $E'_2 = \bigcup_{i=m+1}^{\ell} ext(C_i, E')$ are as desired by the statement.

It is clear that E'_0 satisfies the condition of the lemma (for separator $\bigcup S$). Furthermore, $\bigcup S$ is an $(\bigcup E'_1, \bigcup E'_2)$-separator as the two sets touch unions of different $H \setminus \bigcup S$ components. The size bound $|E'_1| \leq \frac{2}{3}|E'|$ holds by construction.

What is left to show is that the size bound also holds for E'_2. To that end we first observe that $|E'_1| \geq \frac{1}{3}|E'|$. Indeed, by the ordering of components by the size of ext, we have that $ext(C_{m+1}, E') \leq |E'_1|$. Thus, $|E'_1| < \frac{1}{3}|E'|$ would contradict the choice of m. Since E'_1 and E'_2 are disjoint, this leaves at most $|E'| - |E'_1| \leq \frac{2}{3}|E'|$ edges for E'_2. ◀

Proposition 14 already plays an important role in the state of the art of *ghw* computation. The implication of a so-called *balanced separator* of size k has been a key ingredient for various practical implementations for computing *ghw* and related parameters [6, 14, 10]. Our application of this idea is somewhat different from this prior work. There, balanced separators are used to reduce the search space of separators that need to be checked, and to split up the problem into small subproblems. Our applications of Theorem 4 is different: in Algorithm 2 for Compress we use it to find a way to separate the interface to the parent node in the decomposition. Notably, this search requires the split into only a constant number of sets (rather than the possibly linear number of connected components) which is part of why we require our variation of the previous hyperlinkedness result.

5.2 Large Subedge Hypergrids (Theorem 7)

Recall from Definition 6 that an (a, b) *subedge hypergrid* (or (a, b)-shyg) consists of pairwise incompatible subedges $U_1, \ldots, U_a, S_1, \ldots, S_b$ such that: $U_1, \ldots, U_a$ are pairwise disjoint, and for each $j \in [b]$, S_j touches $U_1, \ldots, U_a$. Our proof of Theorem 7 first relates (a, b)-shygs to more restricted structures that we call *strong* (a, b)-shygs.

▶ **Definition 15.** *An (a, b)-shyg $U_1, \ldots, U_a, S_1, \ldots, S_b$ is* strong *if $S_j \cap S_{j'} \cap \bigcup_{i \in [a]} U_i = \emptyset$ for each $j \neq j' \in [b]$.*

▶ **Theorem 16.** *Let H be a $(2, d)$ hypergraph having a strong $(3k + 1, (3k + 1)d + 1)$-shyg. Then $ghw(H) > k$.*

To prove Theorem 7, we first prove Theorem 16 and then demonstrate that non-emptiness of $\mathbf{S}_{k,d}$ implies existence of a strong $(3k + 1, (3k + 1)d + 1)$-shyg. We continue with an overview of our proof of Theorem 16.

An important observation for subedges in $(2, d)$-hypergraphs is that if a subedge U is large enough, and in particular if $|U| > d$, then this will uniquely determine the edge e such that $U \subseteq e$. In this section we will refer to this uniquely determined e as $e(U)$. Using this we state the following auxiliary lemma that gives us a lower bound for separating two subedges that are part of a shyg.

▶ **Lemma 17.** *Let $U_1, U_2, S_1, \ldots, S_b$ be a strong $(2,b)$-shyg of a $(2,d)$-hypergraph H. Let $\{e_1, \ldots, e_q\} \subseteq E(H)$ be such that $\{e(U_1), e(U_2)\} \cap \{e_1, \ldots, e_q\} = \emptyset$ and $W = \bigcup_{i\in[q]} e_i$ is a U_1, U_2-separator. Then $q \geq b/2d$.*

Back to the proof of Theorem 16, we observe that $e(U_1), \ldots, e(U_{3k+1})$ is ρ-stable. This is because each $e(U_i)$, to 'incorporate' all S_j must be of size at least $(3k+1)d+1$ and, in a $(2,d)$-hypergraph, this is too large to be covered by $(3k+1)$ other hyperedges. We will then use Theorem 4 to prove the desired lower bound on $ghw(H)$, by showing that there is the set $\{e(U_1), \ldots, e(U_{3k+1}\}$ cannot be separated in the way specified by Theorem 4. Towards a contradiction, we assume existence of a weak partition E'_0, E'_1, E'_2 of $\{e(U_1), \ldots, e(U_{3k+1}\}$, $|E'_i| \leq 2k$ for each $i \in \{1,2\}$ and $W \subseteq V(H)$ such that $\rho(W) \leq k$, $\bigcup E'_0 \subseteq W$ and W is a $\bigcup E'_1, \bigcup E'_2$- separator. W.l.o.g. we assume existence of $\{e_1, \ldots, e_r\} \subseteq E(H)$, $r \leq k$ such that $W = \bigcup_{i\in[q]} e_i$. Let E^*_0 be the set of all elements of $e(U_1), \ldots, e(U_{3k+1})$ that are subsets of W. We note that $E'_0 \subseteq E^*_0$ and, since no $e(U_i)$ can be covered by k other hyperedges, $E^*_0 \subseteq \{e_1, \ldots, e_r\}$. We also note that both $E'_1 \setminus E^*_0$ and $E'_2 \setminus E^*_0$ are nonempty. Indeed, if say $E'_2 \setminus E^*_0 = \emptyset$ then $3k+1 = |\{e(U_1), \ldots, e(U_{3k+1})\}| = |E'_1 \cup E^*_0| \leq 2k+k$, or in other words, in such a situation E'_0, E'_1, E'_2 could not form a weak partition of $3k+1$ edges. Let $e(U_{i_1}) \in E'_1 \setminus E^*_0$ and $e(U_{i_2})) \in E'_2 \setminus E^*_0$. Then W is a (U_{i_1}, U_{i_2})-separator. By Lemma 17, $r \geq (3k+1)d/2d > k$, and we arrive at a contradiction. This completes the sketch of the proof for Theorem 16.

As a next step, we show that a large enough shyg will imply the existence of a strong $(3k+1, (3k+1)d+1)$-shyg. We will show this inductively via a graded version of strong shygs that we will call *c-strong shygs*. The only difference from Definition 15 is that $|S_j \cap S_{j'} \cap \bigcup_{i\in[a]} U_i| \leq c$ for each $j \neq j' \in [b]$. Thus a strong shyg is 0-strong one and any ordinary shyg is d-strong by definition of a $(2,d)$-hypergraph.

Let us recursively define a function $g(c) = g_{k,d}(c)$ as follows. Let $g(0) = (3k+1)d+1$. For $c > 0$, assuming that $g(c-1)$ has been defined, we let $g(c) = g(0)^2(g(c-1)-2))+1$. Further on, we let $\xi(k,d) = g_{k,d}(d)$ and let $\mathbf{S}_{k,d}(H)$ to be the set of all $(3k+d+1, \xi(k,d))$-shygs of H. The second part of the proof of Theorem 7 is the following statement.

▶ **Theorem 18.** *Let $c \geq 0$ and let H be a $(2,d)$-hypergraph that has a c-strong $(3k+1+c, g(c))$-shyg. Then H has a strong $(3k+1, (3k+1)d+1)$-shyg.*

Theorem 7 is immediate from the combination of Theorem 18 with $c = d$ and Theorem 16. So, let us discuss the proof of Theorem 18.

Proof Sketch. The proof is by induction on c. The case $c = 0$ is immediate as the considered shyg is exactly the desired strong shyg. For $c > 0$, we demonstrate we can 'extract' from the considered shyg either a $c-1$-strong $(3k+c, g(c-1))$ shyg (implying the theorem by the induction assumption) a strong $(3k+1, g(0))$-shyg exactly as required by the theorem.

So, let $U_1, \ldots, U_{3k+c+1}, S_1, \ldots, S_{g(c)}$ be the considered c-strong shyg. Assume first that there is $u \in \bigcup_{i\in 3k+c+1} U_i$ that touches $g(c-1)$ sets S_j. We assume w.l.o.g that $u \in U_{3k+c+1}$ and that the sets S_j touching u are precisely $S_1, \ldots, S_{g(c)-1}$. Since one intersection point between these sets is spent on U_{3k+c+1} for any $j \neq j' \in [g(c)-1]$, $|S_j \cap S_{j'} \cap \bigcup_{i\in[3k+c]} U_i| \leq c-1$. In other words, $U_1, \ldots, U_{3k+c}, S_1, \ldots, S_{g(c)-1}$ is a $(c-1)$-strong shyg implying the theorem by the induction assumption. It remains to assume that each $u \in \bigcup_{i\in 3k+c+1} U_i$ touches at most $g(c-1)-1$ sets S_j. We are going to identify $I \subseteq [g(c)]$ of size $g(0)$ so that for each $j \neq j' \in I$, $S_i \cap S_j \cap \bigcup_{i\in[3k+1]} U_i = \emptyset$. This means that $U_1, \ldots, U_{3k+1}$ along with S_j for each $j \in I$ will form a stron $(3k+1, g(0))$-shyg as required by the theorem.

We use the same elementary argument as if we wanted to show that a graph with many vertices and a small max-degree contains a large independent set. The initial set CI of candidate indices is $[g(c)]$ and initially $I = \emptyset$. We choose $j \in CI$ into I and remove from CI the j and all the j' such that $S_j \cap S_{j'} \cap \bigcup_{i\in[3k+1]} U_i \neq \emptyset$ By definition of $g(c)$, it is enough to show that, apart from j itself, we remove at most $(g(0)-1)(g(c-1)-2)$ other elements. Indeed, the size of $S_j^* = S_j \cap \bigcup_{i\in[3k+1]} U_i$ is at most $(3k+1)d = g(0)-1$ and each point of S_i^*, apart from S_j touches at most $g(c-1)-2$ elements simply by assumption. ◀

5.3 Constructing Gap Cover Approximators (Theorem 9)

Our overall plan for the proof of Theorem 9 is to show that we can produce the elements that make up the desired gap cover approximator, as a combination of four parts, each of which we can bound appropriately. The resulting algorithm is then primarily a matter of enumerating all combinations of elements from these parts. In the following we discuss the construction of these parts and why this yields an FPT algorithm.

The first part is what we will refer to as the set $\mathsf{BE}_p(U)$ of p-big edges w.r.t. U, which are those edges $e \in E(H[U])$ for which $|e| > pd$. The intuition for the importance of big edges is simple, in a $(2,d)$-hypergraph, they are necessary to obtain low weight covers. In particular, any edge cover of U with weight at most p must contain all edges of $\mathsf{BE}_p(U)$: if $|e| > pd$ then the vertices in e cannot be covered by less than $p+1$ other edges, since any other $e' \neq e$ will only intersect e in at most d vertices (see Appendix B in the full version for in-depth discussion of $\mathsf{BE}_p(U)$ sets). For the rest of this section we will simply say that edges are big to mean p-big. Similarly, we will refer to all edges that are not p-big as small.

The more challenging part is to determine the structure of those vertices that are not covered by the big hyperedges. To this end we first define the boundary BE_p^* of the big edges:

$$\mathsf{BE}_p^*(U) := \{e \setminus \bigcup(\mathsf{BE}_p(U) \setminus \{e\}) \mid e \in \mathsf{BE}_p(U)\}.$$

That is $\mathsf{BE}_p^*(U)$ contains those subedges of big edges that are unique to a single big edge.

With respect to this set we will be particularly interested in those members that together cover the intersection of a small edge with the vertices in the boundary. We formalise this via the *spanning set* $sp(e)$ for $e \in E(H[U]) \setminus \mathsf{BE}_p(U)$, which is the set $E' \subseteq \mathsf{BE}_p^*(U)$ such that $e \cap \bigcup E' = e \cap \bigcup \mathsf{BE}_p^*(U)$. We will refer to the cardinality of $sp(e)$ as the *span* of e. The final part we need is those vertices U_p^0 that are not part of any subedge in the boundary BE_p^*, formally $U_p^0 = U \setminus \bigcup \mathsf{BE}_p^*(U)$. In addition to the three parts described above, may need to add subsets of U_p^0 to construct the elements of the gap cover approximator. In particular, to add those vertices that are not part of any big edge. The key observation here is that, under the assumption that $\rho(U) \leq p$, there cannot be too many such vertices and specifically $|U_p^0| = O(p^2 d)$.

Ultimately, what we prove is that for any $U' \subseteq U$ with $\rho(U') \leq k_0$, there is a set X such that $X \supseteq U'$ and $\rho(X) \leq (3k+1+d)k_0$, where X is the union of big edges, short edges and $Y \subseteq U_p^0$. The short edges are actually split in two cases, depending on their span. The construction of X may require some number of short edges with span at most $3k+d$, as well some short edges with span greater than $3k+d$. The last set is the most challenging in terms of achieving an FPT algorithm. All other sets can be bounded in terms of p and d (a small edge with a small span is covered by the union of its span leading to the stated approximation factor). Such a bound seems to not be achievable for the set of small edges with large span. To get around this issue, we show that if there are many small edges with large span, then this implies the existence of a large subedge hypergrid. In more concrete

terms, if there is a set $E' \subseteq \mathsf{BE}^*_p(U)$ with $|E'| > 3k + d$ and $sp^{-1}(E') > \xi(k, d)$ (with ξ as in the definition of $\mathbf{S}_{k,d}(H)$), then $\mathbf{S}_{k,d}(H) \neq \emptyset$. In consequence, we can detect the case for which we could not achieve an FPT bound, and we know that in this case we can safely reject as $ghw(H)$ is guaranteed to be greater than k.

6 Conclusion

We have presented a fixed-parameter tractable algorithm for approximating the generalised hypertree width of hypergraphs with bounded intersection size. In particular, we give an algorithm that either decides in $f(k, d)poly(H)$ time whether a $(2, d)$-hypergraph H has $ghw(H) \leq 4k(k+d+1)(2k-1)$ or rejects, in the latter case it is guaranteed that $ghw(H) \geq k$.

Our main result represents a first step into the area of FPT algorithms for ghw. Our focus has been on developing the overarching framework, and we expect that with further refinement, better approximation factors are achievable and present a natural avenue for further research. The most immediate question is whether subcubic approximation can be achieved. Recall that the $(3k + d + 1)(2k - 1)k_0$ factor for ApproxSep comes from two sources, $(3k + d + 1)k_0$ is a result of using $(3k + d + 1)k_0, k_0$-gap cover approximators to find partial covers. The factor $(2k - 1)$ is a result of combining the partial separators. It is unclear whether either of these factors can be avoided.

For full proof details and an extensive discussion of possible future work we refer to reader to the full version of this paper [17].

References

1 Isolde Adler, Georg Gottlob, and Martin Grohe. Hypertree width and related hypergraph invariants. *Eur. J. Comb.*, 28(8):2167–2181, 2007. doi:10.1016/j.ejc.2007.04.013.

2 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

3 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

4 Grzegorz Fabianski, Michal Pilipczuk, Sebastian Siebertz, and Szymon Torunczyk. Progressive algorithms for domination and independence. In Rolf Niedermeier and Christophe Paul, editors, *STACS 2019*, volume 126 of *LIPIcs*, pages 27:1–27:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.STACS.2019.27.

5 Wolfgang Fischl, Georg Gottlob, Davide Mario Longo, and Reinhard Pichler. Hyperbench: A benchmark and tool for hypergraphs and empirical findings. *ACM J. Exp. Algorithmics*, 26:1.6:1–1.6:40, 2021. doi:10.1145/3440015.

6 Wolfgang Fischl, Georg Gottlob, and Reinhard Pichler. General and fractional hypertree decompositions: Hard and easy cases. In *Proceedings PODS*, pages 17–32. ACM, 2018. doi:10.1145/3196959.3196962.

7 Georg Gottlob and Gianluigi Greco. Decomposing combinatorial auctions and set packing problems. *J. ACM*, 60(4):24:1–24:39, 2013. doi:10.1145/2508028.2505987.

8 Georg Gottlob, Gianluigi Greco, and Francesco Scarcello. Pure nash equilibria: Hard and easy games. *J. Artif. Intell. Res.*, 24:357–406, 2005. doi:10.1613/jair.1683.

9 Georg Gottlob, Martin Grohe, Nysret Musliu, Marko Samer, and Francesco Scarcello. Hypertree decompositions: Structure, algorithms, and applications. In *WG*, volume 3787 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2005. doi:10.1007/11604686_1.

10 Georg Gottlob, Matthias Lanzinger, Cem Okulmus, and Reinhard Pichler. Fast parallel hypertree decompositions in logarithmic recursion depth. In *Proceedings PODS*, pages 325–336. ACM, 2022. doi:10.1145/3517804.3524153.

11 Georg Gottlob, Matthias Lanzinger, Reinhard Pichler, and Igor Razgon. Complexity analysis of generalized and fractional hypertree decompositions. *J. ACM*, 68(5):38:1–38:50, 2021. doi:10.1145/3457374.

12 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *J. Comput. Syst. Sci.*, 66(4):775–808, 2003. doi:10.1016/S0022-0000(03)00030-8.

13 Georg Gottlob, Zoltán Miklós, and Thomas Schwentick. Generalized hypertree decompositions: NP-hardness and tractable variants. *J. ACM*, 56(6):30:1–30:32, 2009. doi:10.1145/1568318.1568320.

14 Georg Gottlob, Cem Okulmus, and Reinhard Pichler. Fast and parallel decomposition of constraint satisfaction problems. *Constraints An Int. J.*, 27(3):284–326, 2022. doi:10.1007/s10601-022-09332-1.

15 Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Trans. Algorithms*, 11(1):4:1–4:20, 2014. doi:10.1145/2636918.

16 Karthik C. S., Bundit Laekhanukit, and Pasin Manurangsi. On the parameterized complexity of approximating dominating set. *J. ACM*, 66(5):33:1–33:38, 2019. doi:10.1145/3325116.

17 Matthias Lanzinger and Igor Razgon. FPT approximation of generalised hypertree width for bounded intersection hypergraphs. *CoRR*, abs/2309.17049, 2023. doi:10.48550/ARXIV.2309.17049.

18 Dániel Marx. Approximating fractional hypertree width. *ACM Trans. Algorithms*, 6(2):29:1–29:17, 2010. doi:10.1145/1721837.1721845.

19 Dan Olteanu and Maximilian Schleich. Factorized databases. *SIGMOD Rec.*, 45(2):5–16, 2016. doi:10.1145/3003665.3003667.

20 Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar. Solving dominating set in larger classes of graphs: FPT algorithms and polynomial kernels. In *Proceedings ESA*, volume 5757 of *Lecture Notes in Computer Science*, pages 694–705. Springer, 2009. doi:10.1007/978-3-642-04128-0_62.

21 Igor Razgon. FPT algoritms providing constant ratio approximation of hypertree width parameters for hypergraphs of bounded rank. *CoRR*, abs/2212.13423, 2022. doi:10.48550/arXiv.2212.13423.

22 Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.