Near-linear time subhypergraph counting in bounded degeneracy hypergraphs

Daniel Paul-Pena*
University of California, Santa Cruz
dpaulpen@ucsc.edu

C. Seshadhri*
University of California, Santa Cruz
sesh@ucsc.edu

Abstract

Counting small patterns in a large dataset is a fundamental algorithmic task. The most common version of this task is subgraph/homomorphism counting, wherein we count the number of occurrences of a small pattern graph H in an input graph G. The study of this problem is a field in and of itself. Recently, both in theory and practice, there has been an interest in *hypergraph* algorithms, where $G = (V, E)$ is a hypergraph. One can view G as a set system where hyperedges are subsets of the universe V.

Counting patterns H in hypergraphs is less studied, although there are many applications in network science and database algorithms. Inspired by advances in the graph literature, we study when linear time algorithms are possible.

We focus on input hypergraphs G that have bounded *degeneracy*, a well-studied concept for graph algorithms. We give a spectrum of definitions for hypergraph degeneracy that cover all existing notions. For each such definition, we give a precise characterization of the patterns H that can be counted in (near) linear time. Specifically, we discover a set of "obstruction patterns". If H does not contain an obstruction, then the number of H-subhypergraphs can be counted exactly in $O(n \log n)$ time (where n is the number of vertices in G). If H contains an obstruction, then (assuming hypergraph variants of fine-grained complexity conjectures), there is a constant $\gamma > 0$, such that there is no $o(n^{1+\gamma})$ time algorithm for counting H-subhypergraphs. These sets of obstructions can be defined for all notions of hypergraph degeneracy.

*Authors supported by NSF CCF-1740850, DMS-2023495, and CCF-1839317.

1 Introduction

Subgraph/homomorphism counting in graphs is a widely studied problem in theory and practice [Lov67, DST02, FG04, DJ04, Lov12, ANRD15, CDM17, PSV17, ST19, CN25, DMW25]. The design of algorithms for homomorphism counting in graphs has gone on for decades, and remains an active area of research [CM77, BW99, DR10, BCL+06, PSV17, DRW19, PS20, Bre19, Bre21, BR22, PPS24, PPS25a, PPS25b]. Let n denote the number of vertices in G and k be the vertices in H. There is a trivial brute force $O(n^k)$ algorithm, and so the primary focus of theoretical research is to beat this bound. Given the practical importance of this problem, there have been advances in designing (near) linear time algorithms [BPS20, BPS21, BGL+22, PPS25a].

Recent work in network science and the theory of algorithms has focused on *hypergraphs* [VBK20, VBK22, ACP+23, cDF+23, LBERS25]. A hypergraph $G = (V(G), E(G))$ is a set system, where each $e \in E(G)$ is a subset of $V(G)$ of size at least two. Many natural real-world graphs are derived from hypergraphs. For example, coauthor/coactor networks are obtained from the hypergraph where authors are vertices, and papers are hyperedges. Hyperedges are replaced with cliques to get a coauthor network. Hence, hypergraphs are considered to be "original" data source, and practitioners suggest that they should be analyzed directly.

The problem of pattern counting in hypergraphs is less understood theoretically, even though it captures many important algorithmic tasks [HTKK08, YTW12, BAS+18, HLZM10, AVB20]. For example, the entire field of Conjunctive Query evaluation in database theory can be viewed as pattern counting in hypergraphs [DK21, KNOZ23]. Any database can be represented as a (colored) hypergraph, and the query is a pattern hypergraph H. We note that recent work by Bressan-Brinkmann-Dell-Roth-Wellnitz has studied FPT algorithms for subhypergraph counting [BBD+25].

Formally, given a pattern hypergraph $H = (V(H), E(H))$, an H-homomorphism is a map $f : V(H) \to V(G)$ that preserves hyperedges. Abusing notation, given $e = \{v_1, v_2, \ldots\}$, let $f(e) = \{f(v_1), f(v_2), \ldots\}$. So, $\forall e \in E(H)$, $f(e) \in E(G)$. If f is an injection (so distinct vertices of H are mapped to distinct vertices of G), this map corresponds to some subhypergraph [1]. We use $\mathrm{Hom}_H(G)$ (resp. $\mathrm{Sub}_H(G)$) to denote the count of the distinct H-homomorphisms (resp. H-subhypergraphs). The focus of this work is:

Are there characterizations of H and classes of G where $\mathrm{Hom}_H(G)$ can be computed in near-linear time?

1.1 Main result

A natural starting point for near-linear time algorithms for hypergraph homomorphism counting is to focus on *bounded degeneracy hypergraphs*[2]. There is a recent theory of linear time algorithms for bounded degeneracy *graphs* [Bre19, BPS20, BPS21, BR22, BGL+22, PPS24, PPS25a]. This is a rich class containing all proper minor-closed families, bounded expansion families, and preferential attachment graphs [Ses23]. Practical subgraph counting algorithms typically use algorithmic techniques for bounded degeneracy graphs [ANRD15, JSP15, PSV17, OB17, JS17, PS20].

Recent works in database theory have focused in studying databases with "degree constraints" [KNS16], and this theory has lead to cutting edge algorithms for query evaluation [KNS25]. A

[1] Such a map is an embedding. Each subhypergraph corresponds to $|Aut(H)|$ different embeddings, where $Aut(H)$ is the number of automorphisms of H (see [BBD+25]).

[2] Technically, we mean hypergraphs belonging to classes with bounded degeneracy.

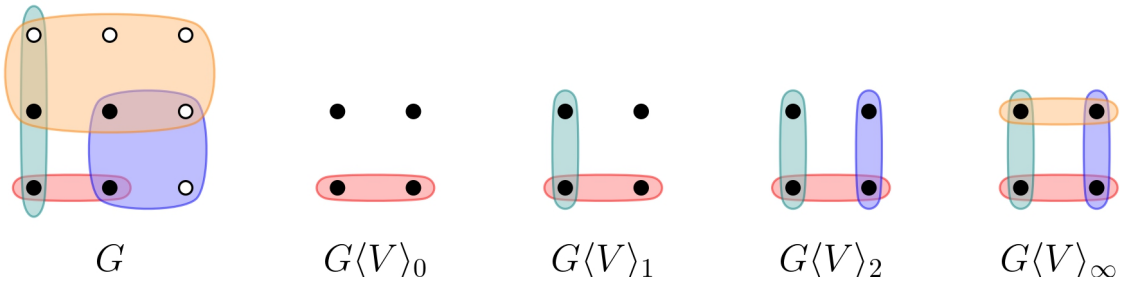


Figure 1: An example of a graph G. Let V be the set of black vertices. $G\langle V\rangle_l$ represents the l-trimmed subhypergraph of G induced by the vertices in V. A hyperedge is only included in the subhypergraph when it has at most l vertices outside V.

database with degree constraints can be represented as a bounded degeneracy hypergraph. A recent result by Deeds and Merkl [DM25] introduced partition constraints for databases, which are essentially equivalent to breaking databases/tables into hypergraphs with bounded degeneracy.

There are numerous possible definitions for the degeneracy of hypergraphs. Our results provide dichotomy theorems for near-linear homomorphism and subhypergraph counting algorithms for all these definitions.

As usual, the input hypergraph is $G = (V(G), E(G))$ with n vertices and m hyperedges. We consider H to be constant sized and independent of the input, and fold dependencies on H into $O(\cdot)$ or $poly(\cdot)$. The *arity* of a hyperedge is its size. We allow G to have hyperedges of different arity. The *rank* of a hypergraph is the maximum arity among all its hyperedges. We assume that the rank of G is bounded, and again suppress dependencies on the rank in $O(\cdot)/\text{poly}(\cdot)$ notation.[3]

Hypergraphs introduce complications when defining *induced* subhypergraphs, which this definition lays out.

Definition 1.1. *Let $l \in \mathbb{Z}^+ \cup \{\infty\}$. For any subset $S \subseteq V(G)$, the induced l-trimmed subhypergraph on S is the set of hyperedges $\{e \cap S \mid e \in E(G), |e \setminus S| \leq l\} \setminus \{\emptyset\}$.*

Conventionally, if l is not specified, then it is set to ∞. (Depending on context, arity 1 hyperedges might be discarded.)

We note that the term "trimming" comes from [BBD+25], though they only define the $l = \infty$ version. See Fig. 1 for an example of different induced l-trimmed subhypergraphs. When $l = 0$, this is the simplest (standard) notion of induced subhypergraph. Only hyperedges contained in S are considered. In some settings, hypergraphs are thought of as simplicial complexes wherein all subsets of a hyperedge are also part of G. This corresponds to $l = \infty$, where an induced subhypergraph is formed by taking the intersection of every hyperedge with S (technically, we only need to set l to be the rank of G). For intermediate values of l, we only include the intersection $e \cap S$ if at most l vertices of e are outside S. This concept is related to nuanced notions of hypergraph cuts [VBK20, VBK22].

We now define the hypergraph degeneracy. The degree of a vertex is the number of hyperedges in which it participates.

[3]Note that hyperedges of unbounded arity can not be mapped by any hyperedge of H, and therefore do not affect the homomorphism/subhypergraph counts. If given a hypergraph of unbounded rank, one can filter out the high arity hyperedges in $O(m)$ time to get a bounded rank hypergraph.

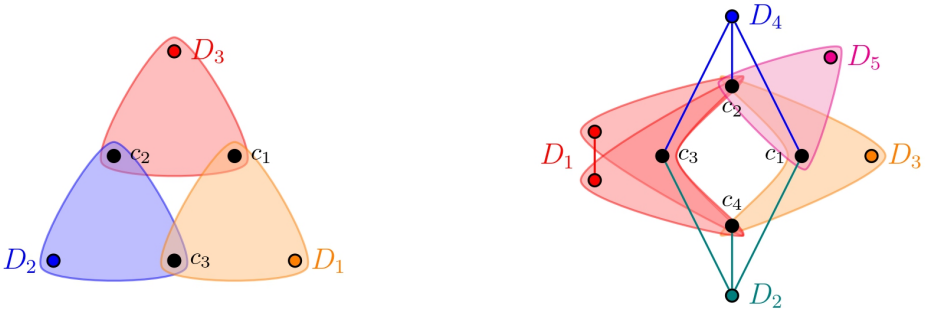


Figure 2: Two examples of obstructions in $\mathcal{H}_0$. The example on the left has a core with $k = 3$. Removing the vertices from the core disconnects the hypergraph into 3 connected components (single vertices $D_1 - D_3$). The sets $E_1 - E_3$ correspond to the hyperedges containing each D_i. The example on the right has a core with $k = 4$. Removing the core we end up with 5 connected components. Note that we have an extra component D_5.

Definition 1.2. *The l-degeneracy of G, denoted $\kappa_l(G)$, is the minimum value κ such that every induced l-trimmed subhypergraph of G has minimum degree of at most κ.*

Note that for $l < l'$, $\kappa_l(G) \leq \kappa_{l'}(G)$. So, having bounded 0-degeneracy is a weaker condition than having bounded l-degeneracy (for $l > 0$). The 0-degeneracy corresponds to the standard definition of degeneracy in hypergraphs [KR06], while the ∞-degeneracy is polynomially equivalent to the degeneracy of the clique completion of G (assuming bounded rank), also called skeletal degeneracy [FSS+23].

Our aim is to characterize H where $\mathrm{Hom}_H(G)$ and $\mathrm{Sub}_H(G)$ can be counted in near-linear time for bounded l-degeneracy inputs G. Towards this characterization, we define a series of obstruction hypergraphs.

Definition 1.3. *A hypergraph H is an* l-obstruction *if the following hold. For some $k \geq 3$ there is a* core *$C \subseteq V(H)$ of size $|C| = k$. Let the connected components of the induced ∞-trimmed subhypergraph on $V(H) \setminus C$ be (the vertex sets) $D_1, D_2, \ldots$. For each D_i, let E_i be the set of hyperedges of $E(H)$ that are completely contained in $C \cup D_i$, that is $E_i = \{e \in E(H) : e \subseteq C \cup D_i, |e| > 1\}$.*

The core C and $E_1, E_2, \ldots$ satisfy the following conditions:

1. *The l-trimmed subhypergraph induced by the core C is empty*[4].
2. *There is no E_j that contains all of C. $\forall E_j, C \not\subseteq \bigcup_{e \in E_j} e$.*
3. *For each $c \in C$, E_i contains*[5] *the vertices $C \setminus \{c_i\}$ and* does not *contain c_i.*

The set of l-obstructions is denoted by $\mathcal{H}_l$. A hypergraph H' is called $\mathcal{H}_l$ ITS (induced trimmed subhypergraph) free if it does not contain any member of $\mathcal{H}_l$ as an induced ∞-trimmed subhypergraph.

Remark 1.4. *As mentioned, the sets $D_1, D_2, \ldots$ are the connected components of the induced ∞-trimmed subhypergraph on $V(H) \setminus C$. From condition 1 we have that the core is empty, that is, every hyperedge of arity more than 1 must include some vertex outside the core. Note that from the*

[4] It is allowed to include hyperedges of arity 1, as they do not affect the connectivity of the core.

[5] Technically, we number the vertex sets $D_1, D_2, \ldots$ so that this condition holds.

definition, no edge can contain vertices in two different sets D_i, D_j. Therefore every hyperedge will belong to an unique set E_i.

Condition 2 specifies that no set E_i contains all the vertices in the core, while condition 3 says that for each vertex c_i in the core, there is at least a set that contains $C \setminus \{c_i\}$. We reorder the sets D_i and E_i such that each E_i corresponds to c_i. Note that there might be more sets E_i than elements in the core.

We give more intuition on these conditions in §1.2. This is a technical definition, but we can prove that Definition 1.3 are the only obstructions for near-linear time algorithms.

For the hardness, we state a generalization of the Triangle Detection Conjecture [AW14], which is the fundamental assumption used for linear time hardness for subgraph counting [BPS21, BGL+22, PPS24, PPS25a]. This conjecture states that there is no near-linear time algorithm for finding triangles in an arbitrary graph. The best known algorithm takes $O(m^{1.41\cdots})$ time [AYZ97]. A k-simplex is a hypergraph with $k+1$ vertices and $k+1$ hyperedges of arity k, where each hyperedge includes a different subset of k vertices. The triangle is the 2-simplex, while the tetrahedron is the 3-simplex. Detecting simplices is the hypergraph generalization of triangle detection. No algorithm is known for detecting simplices of any arity in linear time.

Conjecture 1.5 (Simplex detection conjecture)**.** *For every $k \geq 2$, there exists a $\gamma > 0$ such that any (even randomized) algorithm to decide whether a hypergraph with n vertices and m hyperedges contains a k-simplex requires $\Omega(m^{1+\gamma})$ time in expectation.* [6]

Theorem 1.6 (Main Theorem)**.** *Let $l \in \mathbb{Z}^+ \cup \{\infty\}$ and H be a hypergraph.*

- *Suppose H is $\mathcal{H}_l$ ITS free. Then, there is an algorithm that computes $\mathrm{Hom}_H(G)$ in time $\mathrm{poly}(\kappa_l) n \log n$.* [7]
- *Otherwise, assuming the Simplex Detection Conjecture, there exists an absolute constant $\gamma > 0$, such that: for all functions $f : \mathbb{N} \to \mathbb{N}$, any algorithm that computes $\mathrm{Hom}_H(G)$ requires $f(\kappa_l) n^{1+\gamma}$ time.*

Thus, for bounded κ_l for any value of l, we have a precise characterization of when $\mathrm{Hom}_H(G)$ can be computed in near-linear time.

We can extend the hardness characterization to subhypergraph counting. The quotient set of H is the set of hypergraphs obtained by merging sets of vertices that form a partition of H (see Definition 3.1 for a formal definition). Like in the case of graphs, we can use the homomorphism basis to lift the hardness from counting homomorphisms of the quotient set of H to counting subhypergraphs of H.

Theorem 1.7 (Characterization for subhypergraphs)**.** *Let $l \in \mathbb{Z}^+ \cup \{\infty\}$ and H be a hypergraph.*

- *If every hypergraph in the quotient set of H is $\mathcal{H}_l$ ITS free. Then, there is an algorithm that computes $\mathrm{Sub}_H(G)$ in time poly$(\kappa_l) n \log n$.*

[6] The $(k+1, k)$-Hyperclique Hypothesis presented in [LWW18] conjectures that any algorithm for finding a k-simplex in a k-regular hypergraph requires $n^{k+1-o(1)}$ time. In the dense case, it is a stronger assumption than our conjecture.

[7] Technically, the result of this theorem is fixed-parameter near-linear time. However, this implies a near-linear time algorithm for hypergraph classes with bounded degeneracy.

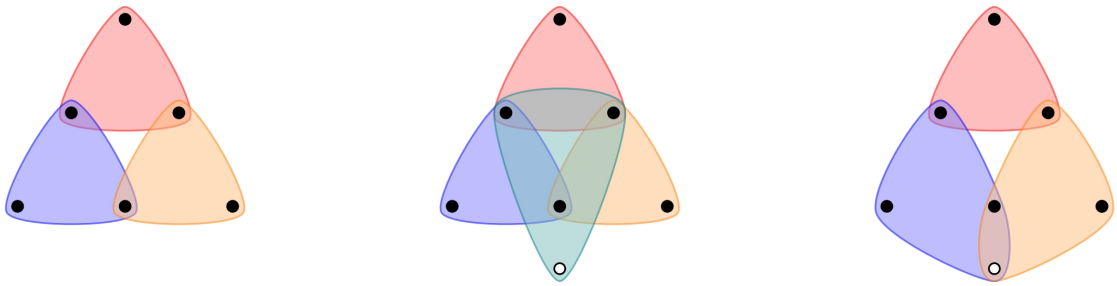


Figure 3: Left: An obstruction in $\mathcal{H}_0$. Center: A pattern that contains the obstruction as a (0-trimmed) subhypergraph when induced by the black vertices, but it does not contain any obstruction as a ∞-trimmed subhypergraph. Therefore, it can be counted efficiently. Right: A pattern that contains the obstruction as the ∞-trimmed hypergraph induced by the black vertices. Note that the induced 0-trimmed subhypergraph only contains the red hyperedge. This pattern cannot be counted efficiently.

- *Otherwise, assuming the simplex detection conjecture, there exists an absolute constant $\gamma > 0$, such that: for all functions $f : \mathbb{N} \to \mathbb{N}$, any algorithm that computes $\mathrm{Sub}_H(G)$ requires $f(\kappa_l)n^{1+\gamma}$ time.*

1.2 Perspectives on Definition 1.3 and Theorem 1.6

We explain why all the technicalities of Definition 1.3 are necessary. This provides useful intuition and also demonstrates why subhypergraph counting is technically challenging. We consider Definition 1.3 as our main discovery, which illustrates why subhypergraph counting is not a simple generalization of subgraph counting.

We give examples showing that removing any aspect of Definition 1.3 leads to patterns that can be counted in linear time. Let us focus on $l = 0$ for ease of presentation. Fig. 2 has examples of patterns in $\mathcal{H}_0$. The edge sets E_i in Definition 1.3 are called "connectors". For convenience, we use "hardness" for near-linear time hardness. When we say "counting", we refer to homomorphism counting.

For context, we recall the characterization of hard pattern graphs for graph homomorphism counting [BPS21, BGL+22]. If the longest induced cycle length (LICL) of H is at least 6, it is hard; otherwise, there is a near-linear time algorithm to count $\mathrm{Hom}_H(G)$. The difficulty of discovering Definition 1.3 and the many technical conditions for hypergraph counting should be contrasted with the easy categorization of the graph case.

Containing $\mathcal{H}_0$ as induced ∞-trimmed subhypergraphs: Theorem 1.6 states that the hard patterns contain a member of $\mathcal{H}_0$ as induced ∞-trimmed subhypergraphs. We cannot replace that with induced (say) 0-trimmed subhypergraphs, which is the typical definition of induced hypergraphs. In fact, we can find both patterns that contain an obstruction in $\mathcal{H}_0$ as an induced (0-trimmed) subhypergraph that are easy to count, and patterns that do not contain an obstruction in $\mathcal{H}_0$ as an induced subhypergraph that are hard to count. For example, consider Fig. 3. On the left, we have a pattern H in $\mathcal{H}_0$. In the center, we modify H and obtaining a pattern H' that contains H as an induced 0-trimmed (standard) subhypergraph but not as an induced ∞-trimmed subhypergraph. Our algorithm can count $\mathrm{Hom}_{H'}(G)$ in near-linear time. Finally, on the right, we modify it again, obtaining H'', which contains H as an induced ∞-trimmed subhypergraph but not

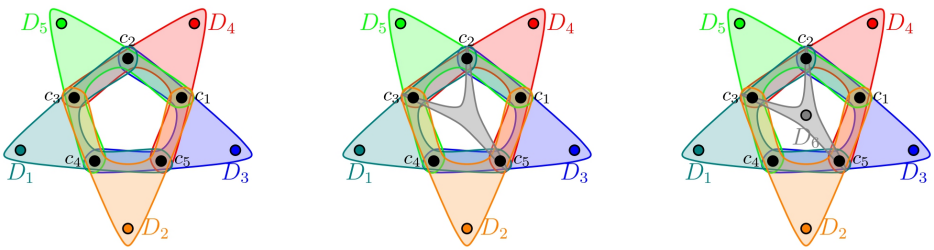


Figure 4: An example of an obstruction with a core of size 5. Adding the gray hyperedge no longer results in an obstruction. Adding an external vertex to the gray hyperedge creates a new obstruction.

as an induced 0-trimmed subhypergraph. We can prove hardness for counting this pattern.

The empty core: The first condition of Definition 1.3 requires that the core C induces no l-trimmed hyperedges. As an example, we give an obstruction $H \in \mathcal{H}_0$ with $k = 5$ (Fig. 4). Note that each hyperedge contains five vertices; four in the core, and one vertex outside. If we add any hyperedge to the core, the resulting H' can be counted in near-linear time. Our algorithm can exploit this extra connectivity to efficiently count H'. Adding an external vertex to that hyperedge results in a pattern that is again hard to count, as the hyperedge just forms a new connector, and the induced 0-trimmed subhypergraph of the core is again empty.

This core condition is the only place where l occurs. Refer to Fig. 5a. We can construct a pattern $H \in \mathcal{H}_1$, so it is hard to count for both 0-degeneracy and 1-degeneracy bounded input graphs. We modify it so that the 1-trimmed hypergraph induced by the core is non-empty, but the corresponding 0-trimmed subhypergraph is empty. So, it remains hard for bounded 0-degeneracy inputs, but can be counted in near-linear time for bounded 1-degeneracy inputs.

No E_j contains the core: We give an obstruction $H \in \mathcal{H}_0$ in Fig. 5b. We add one hyperedge to get H' so that, according to Definition 1.3, there is a connector of hyperedges E_j that contains the three vertices in the core. We can also see that H' does not contain any obstruction as an induced trimmed subhypergraph. So H' can be counted by our algorithms in near-linear time.

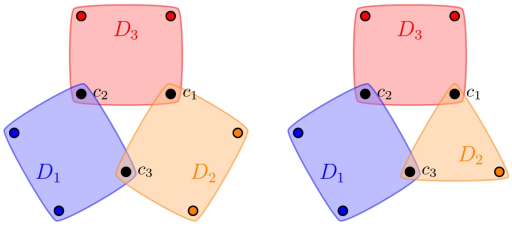
E_i contains $C \setminus \{c_i\}$: Let us explain the last condition of Definition 1.3.

We give two examples. First, consider an example of an obstruction $H \in \mathcal{H}_0$ with $k = 3$. The core has vertices c_1, c_2, c_3. As shown in Fig. 6a, there is a connector E_3 that contains $\{c_1, c_2\}$. We disconnect D_3, so no connector spans $\{c_1, c_2\}$. The resulting pattern is no longer an obstruction.

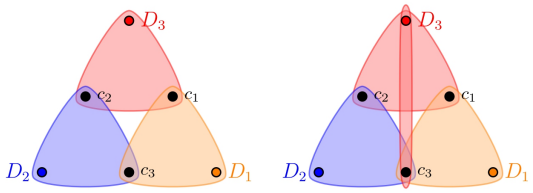
Second, consider now an obstruction in $\mathcal{H}_0$ with core c_1, c_2, c_3, c_4. As shown in Fig. 6b, there is a connector E_4 that contains $\{c_1, c_2, c_3\}$. We split it into two connectors spanning $\{c_1, c_2\}$ and $\{c_2, c_3\}$. Intuitively, all portions of the core are connected, but H' is no longer an obstruction.

However, in this case, we can see that the ∞-trimmed subhypergraph induced by all the vertices except c_4 will actually be an obstruction in $\mathcal{H}_0$ with $\{c_1, c_2, c_3\}$ as the core. Note that this was not the case before.

The role of k: It is easiest to understand $k = 3$. In this case, we can think of the connectors as (hyper)paths that connect pairs in the core. The obstruction looks like an induced hypercycle with at least 6 vertices. This is exactly the pattern characterization in the graph setting, of having an induced cycle of length at least 6. For the graph setting, we can prove that obstructions for larger k contain an induced cycle with at least 6 vertices. Even for the hypergraph case, if all hyperedges include at most one vertex of the core, we can show a similar statement (Claim 1.8).

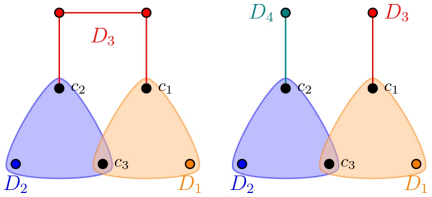


(a) The pattern on the left is an obstruction in both $\mathcal{H}_0$ and $\mathcal{H}_1$. By contracting the two orange vertices, the 1-trimmed subhypergraph induced by the core will no longer be empty. It will not be an obstruction in $\mathcal{H}_1$.

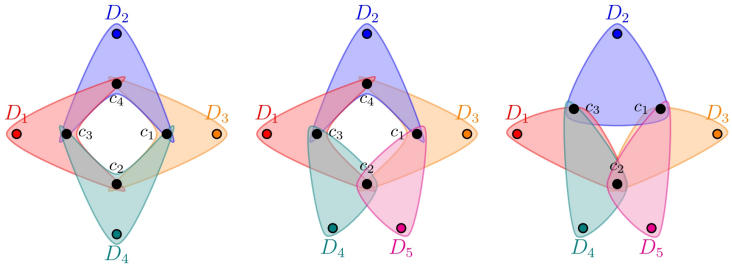


(b) An example of an obstruction in $\mathcal{H}_0$. Adding a hyperedge makes E_3 contain all the vertices in the core, resulting in a pattern not in $\mathcal{H}_0$.

Figure 5: More examples of obstructions.



(a) An example of an obstruction in $\mathcal{H}_0$. Disconnecting D_3 makes the pattern no longer an obstruction, as no connector covers $\{c_1, c_2\}$.



(b) An example of an obstruction H in $\mathcal{H}_0$. We obtain H' by splitting the teal hyperedge into two different connectors, C will no longer be a valid core. However, the induced ∞-trimmed subhypergraph by $V(H') \setminus \{c_4\}$ (right) actually forms an obstruction.

Figure 6: Examples showcasing the connectivity requirements of the connectors.

It becomes complicated when hyperedges of higher arity contain multiple vertices of the core. Consider the obstructions in Fig. 2 with $k = 4$ and in Fig. 4 with $k = 5$. They do not contain an obstruction with a smaller k, and therefore Definition 1.3 must consider all values of k.

The role of l: As mentioned earlier, l only appears in the first condition of Definition 1.3. We find it curious that all other notions of induced subhypergraphs either used the 0-trimmed or ∞-trimmed variants, regardless of the l value. Moreover, note that for $l < l'$, $\mathcal{H}_l \supseteq \mathcal{H}_{l'}$.

When $l = \infty$, the obstruction becomes simpler and defaults to the graph case. Let the clique completion of a hypergraph be the graph obtained by replacing each hyperedge with a clique. We can prove the following.

Claim 1.8. *H is $\mathcal{H}_\infty$ ITS free iff LICL$(H^c) < 6$ where H^c is the clique completion of H.*

Where LICL is the length of the longest induced cycle in H^c. This is not too difficult to prove, since no hyperedge can contain more than one vertex of the core. As we discussed earlier, the connectors can be broken down into paths, and it defaults to the $k = 3$ setting.

Why hypergraphs are challenging: The examples shown highlight the difficulties in characterizing hard patterns for near-linear time algorithms. In the graph case, the basic obstruction is a subdivision of a triangle, and the hardness of general triangle counting neatly matches up with the

algorithmic benefits of bounded degeneracy inputs. For hypergraphs, while the same philosophy holds, the obstructions are much more complicated. At a high level, it is some embedding of a simplex, but the specific connectivity imposed by Definition 1.3 is quite tricky. Obstructions can be quite brittle, in that adding a strategic hyperedge allows for efficient algorithms. Overall, we believe that this shows the need for more theory of subhypergraph counting, and the potential for precise dichotomy theorems.

1.3 Related work

Subgraph and homomorphism counting are vast topics, and we only give a brief list of papers in this area. We focus on results that are directly relevant for our work.

Tree decompositions have played a central role in subgraph/homomorphism counting. Tree decomposition and treewidth were first discovered by Bertele-Brioschi [BB73] and Halin [Hal76]. They were also developed in the context of graph minor theory by Robertson and Seymour [RS83, RS84, RS86]. A classic result of Díaz et al [DST02] gives a $O(2^k n^{tw(H)+1})$ algorithm for determining the H-homomorphism count, where $tw(H)$ is the treewidth of H. Assuming $\#W[1]$-hardness, Dalmau and Jonsson [DJ04] proved that $\mathrm{Hom}_H(G)$ is polynomial time solvable iff H has bounded treewidth.

The notion of graph *degeneracy* has played an important role in subgraph counting. The first use was pioneered by Chiba-Nishizeki for clique counting [CN85]. Since then, it has been used in many theoretical and applied subgraph counting results [CN85, Epp94, ANRD15, JSP15, PSV17, OB17, JS17, PS20]. The short survey of Seshadhri discusses these connections [Ses23].

A seminal result of Bressan connected degeneracy to tree decompositions, through the concept of DAG treewidth [Bre19, Bre21]. This concept has been central in many near-linear time algorithms for bounded degeneracy inputs. Bera-Pashanasangi-Seshadhri initiated the theory of linear time homomorphism counting [BPS20], showing that all patterns with at most 5 vertices could be counted in linear time. Bera-Pashanasangi-Seshadhri and Bera-Gishboliner-Levanzov-Seshadhri then gave a precise characterization, showing that H-homomorphisms can be counted in near-linear time in bounded degeneracy graphs iff $LICL(H) < 6$ [BPS21, BGL+22]. (This assumes fine-grained complexity conjectures.)

Hypergraph cut functions are a relatively recent area of study, but there has been a surge of research on this problem in the network science community[VBK20, VBK22]. Subhypergraph counting is not as well studied. A recent work by Bressan-Brinkmann-Dell-Roth-Wellnitz studied FPT algorithms for subhypergraph counting [BBD+25], showing that the homomorphism basis also extends to the hypergraph setting. The problem of homomorphism counting in hypergraphs has also been studied in hypergraphs of both bounded [DJ04] and unbounded rank [GM14] from a parameterized complexity point of view.

The triangle detection conjecture was first stated by Abboud-Williams for getting fine-grained complexity results [AW14]. The best known algorithm for the triangle detection problem uses fast matrix multiplication and runs in time $O(\min\{n^\omega, m^{2\omega/(\omega+1)}\}) \approx O(m^{1.41\cdots})$ [AYZ97]. If $\omega = 2$, this yields a running time of $m^{4/3}$, which many believe to be the true complexity. Any improvement even on this bound would be considered a huge breakthrough in algorithms, and a near-linear time algorithm would completely upend our understanding of triangle counting.

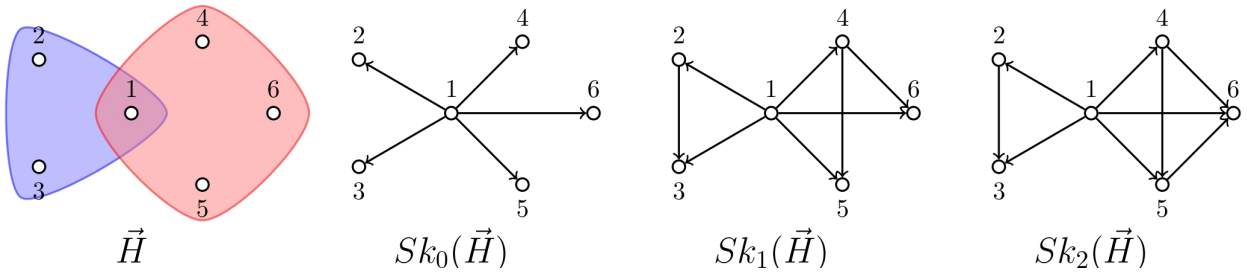


Figure 7: An example of a directed acyclic hypergraph following the ordering of the vertex numbering with all its different l-skeletons.

2 Technical overview

In this section, we give a brief exposition of the main technical aspects of the paper along with the main results of each of the sections.

2.1 Directed acyclic hypergraphs

Most of the work in homomorphism counting of degenerate graphs requires exploiting orientations of the input graph with low outdegree and reducing the problem to homomorphism counting of directed acyclic graphs. This approach was first started by Chiba-Nishizeki [CN85].

If we wish to do the same in the hypergraph setting, we need to start by defining directed hypergraphs. In the most common definition of a directed hypergraph, each edge is defined by two sets of vertices: the tail and the head [AL17]; this directly generalizes directed graphs where, for every directed edge, one vertex is the tail and another the head.

However, an alternative way of generalizing directed acyclic graphs is by using an ordering of the vertices. We will use the term directed acyclic hypergraph (DAH) to refer to this variant of direct hypergraphs.

Definition 2.1 (Directed acyclic hypergraph)**.** *A directed acyclic hypergraph $\vec{G}$ is a hypergraph G with an ordering function $\pi : V(G) \to [|V(G)|]$.*

In a DAH, every hyperedge will be given by a sorted list of vertices that matches the order of the hypergraph.

2.2 l-skeletons

In relation to DAH, we define different notions of "skeletons". The l-skeleton of a DAH $\vec{H}$ is the directed acyclic graph obtained by connecting each of the first $(l+1)$ vertices on each hyperedge of $\vec{G}$ to any vertex that appears after them in the internal ordering of the hyperedge. If $l = 0$, this is equivalent to replacing every hyperedge by an "out-star" (a star where the direct edges go from the center to the external nodes). If $l = \infty$ (or just an integer at least the rank of H minus 2), this becomes the orientation of $\vec{H}^c$ that matches the ordering of $\vec{H}$. See Fig. 7 for an example.

2.3 The degeneracy orientations

A property of degenerate graphs is that one can obtain an ordering of the vertices such that orienting the graph using such ordering yields a directed acyclic graph with maximum out-degree equal to

the degeneracy. Moreover, Matula and Beck showed that such an orientation can be constructed in linear time[MB83].

In §4 we show that this classical result can be extended to l-degenerate hypergraphs. First, we define the l-outdegree of a vertex in a DAH $\vec{G}$ as its outdegree in the l-skeleton of $\vec{G}$. We show that we can construct an orientation of any bounded l-degeneracy hypergraph G that has bounded l-outdegree.

Lemma 2.2. *Let $l \in \mathbb{Z}^+ \cup \{\infty\}$ and let G be a hypergraph with l-degeneracy $\kappa_l(G)$ and rank $r(G)$. There exists an ordering of the vertices of $V(G)$ such that the directed acyclic hypergraph $\vec{G}^{(l)}$ obtained by applying this ordering to G has a maximum l-outdegree $\Delta_l^+(\vec{G}^{(l)}) \leq \kappa_l(G) \cdot r(G)$. Moreover, given G, one can construct $\vec{G}^{(l)}$ in $O(n + m(\log m \cdot r(G) + r(G)^2))$ time.*

2.3.1 DAG-tree decomposition for hypergraphs

Bressan introduced the concepts of DAG-tree decomposition and DAG-treewidth [Bre19, Bre21] in the context of counting homomorphisms of degenerate graphs. The DAG-tree decomposition gives a way of partitioning the pattern graph in order to compute the homomorphism counts efficiently from the counts of each partition, while the DAG-treewidth is the maximum size of the bags of sources of the tree decomposition, and gives an upper bound on the complexity of counting the pattern. It was proved that we can only find linear time algorithms for counting patterns with a DAG-treewidth of 1.

We extend this notion to l-degenerate hypergraphs. We define the l-DAG-treewidth (τ_l) of a directed acyclic hypergraph $\vec{H}$ as the DAG-treewidth of its l-skeleton (see Definition 3.9 for more details).

In §5 we show how to use these decompositions to compute the homomorphisms of directed acyclic hypergraphs, extending the homomorphism counting result of Bressan [Bre19] to hypergraph homomorphism counting, for all notions of l-degeneracy and l-DAG-treewidth.

Theorem 2.3. *Let $l \in \mathbb{Z}^+ \cup \{\infty\}$ and H, G be two bounded rank hypergraphs. There is an algorithm that computes $\mathrm{Hom}_H(G)$ in time $\mathrm{poly}(\kappa_l) n^{\tau_l(H)} \log m$.*

2.3.2 Characterizing the hypergraphs with l-DAG-treewidth equal to 1

To find near linear time countable instances of H, we analyze which instances of H will have $\tau_l = 1$ in all its acyclical orientations. In §6 we prove that being $\mathcal{H}_l$ ITS free is the exact condition for having l-DAG-treewidth equal to 1. This, together with Theorem 2.3 gives an upper bound in the complexity of counting homomorphisms of $\mathcal{H}_l$ ITS free hypergraphs, giving the upper bound of Theorem 1.6.

Theorem 2.4. *Let $l \in \mathbb{Z}^+ \cup \{\infty\}$ and H a hypergraph. $\tau_l(H) = 1$ if and only if H is $\mathcal{H}_l$ ITS free.*

2.3.3 The hardness

In §7, we prove that our characterization is tight. The proof combines the color-coding techniques introduced in [AYZ94], with a construction that allows us to relate colorful homomorphism counting of patterns in $\mathcal{H}_l$ with finding colorful simplices. This theorem, together with Conjecture 1.5 gives the hardness result of Theorem 1.6.

Theorem 2.5. *Let $l \in \mathbb{Z}^+ \cup \infty$ and let H be a hypergraph that contains $H' \in \mathcal{H}_l$. If there exists an algorithm that computes $\mathrm{Hom}_H(G)$ in time $f(\kappa_l)O(n^\gamma)$ for all G and some $\gamma > 1$, then there is an algorithm that, for some $k \geq 2$, detects if a regular arity $k+1$ hypergraph G contains a k-simplex in time $\tilde{O}(m^\gamma)$.*

2.3.4 Computing subhypergraph counts

Finally, in §8 we extend the homomorphism result to subhypergraphs. The number of subhypergraphs of a pattern can be written as a linear combination of the homomorphism counts of each hypergraph in its quotient set (see Lemma 3.2). This directly gives the upper bound of Theorem 1.7.

In a recent paper, Bressan et al. showed that one can extend the hardness of hypergraph homomorphism counting to subhypergraph counting using the *homomorphism basis* [BBD+25], analogous to the graph case [CDM17]. We show that one can also use the homomorphism basis to extend more fine-grained bounds in hypergraphs.

3 Preliminaries

3.1 Hypergraphs and homomorphisms

A **hypergraph** G is a generalization of a graph defined by a vertex set $V(G)$ and a hyperedge set $E(G)$ where every hyperedge $e \in E(G)$ is a non-empty subset of the vertices in $V(G)$ [8]. We assume that the hypergraphs are simple, that is, they do not contain duplicated hyperedges.

The **arity** of a hyperedge e is the number of vertices that is incident to, $|e|$. The rank $r(H)$ of a hypergraph H is the maximum arity of its hyperedges. Graphs corresponds to the special case of hypergraphs of regular arity 2 (where all hyperedges have arity exactly equal to 2). The **degree** $d_H(v)$ of a vertex v in a hypergraph H is the number of hyperedges that contain the vertex. If H is clear from the context, we might write $d(v)$ instead.

A **homomorphism** from a hypergraph H to a hypergraph G is a map $\phi : V(H) \to V(G)$ such that for each $e \in E(H)$ we have $\{\phi(v) : v \in e\} \in E(G)$, that is, for every hyperedge in H the image of its vertices in G corresponds exactly with one of its hyperedges[9]. Note that a hyperedge in H can be mapped to an edge in G of equal or lesser arity, but never to an edge of greater arity. Hence, we can ignore all edges of G with arity greater than the rank of H. We also define **arity-preserving homomorphisms** as a map $\phi' : V(H) \to V(G)$ such that $\forall e \in E(H), \exists e' \in E(G)$ s.t. $\{\phi(v) : v \in e\} = e'$ and $|e| = |e'|$.

We use $\Phi(H, G)$ to denote the collection of homomorphisms from H to G. We use $\mathrm{Hom}_H(G)$ for the number of homomorphisms from H to G, $|\Phi(H, G)|$. Note that in the case of hypergraphs with arity 2 in all their hyperedges, this problem corresponds to the standard homomorphism counting problem in graphs. We use $\mathrm{rHom}_H(G)$ for the number of arity-preserving homomorphisms from H to G.

We use H^c to denote the **clique completion** of H, that is, the graph obtained by replacing every hyperedge e of H by a clique of size $|e|$. $E(H^c) = \{e' \subseteq e | e \in E(H), |e'| = 2\}$.

Additionally, we will use the following notation regarding homomorphisms:

- We use $V(\phi)$ to represent the **domain** of a homomorphism, that is, the vertices being mapped by it.
- We say that two homomorphisms ϕ, ϕ' **agree** if for every vertex v in $V(\phi) \cap V(\phi')$, $\phi(v) = \phi'(v)$.
- The **restriction** of ϕ to a set $V' \subseteq V(\phi)$, is the homomorphism ϕ' with $V(\phi') = V'$ and $\phi'(v) = \phi(v)$ for all $v \in V'$.
- Let H' be a subhypergraph of H. The **extension** of a homomorphism $\phi : V(H') \to V(G)$ to H, $ext(H, G; \phi)$, is the number of homomorphisms $\phi' : V(H) \to V(G)$ that agree with ϕ.

3.2 Subhypergraphs

In the standard notion of subhypergraphs, every hyperedge of the subhypergraph must be completely contained in the set of vertices of the subhypergraph. Formally, a hypergraph $H' = (V(H'), E(H'))$ is a **subhypergraph** of $H = (V(H), E(H))$ if $V(H') \subseteq V(H)$ and $E(H') \subseteq \{e \in$

[8] An alternative way of defining hypergraphs uses an incidence function f_G which maps every edge $e \in E(G)$ to a subset of $V(G)$. In our context, both definitions are equivalent.

[9] An alternative way to define homomorphisms in hypergraphs uses two maps, one for vertices and one for edges, however, because we assume our input graph to be simple, this is not necessary.

$E(H)|e \subseteq V(H')\}$. Given two hypergraphs, we will use $\text{Sub}_H(G)$ for the number of subhypergraphs of G that are isomorphic to H.

The quotient set of a hypergraph is the set of all possible hypergraphs that can be obtained by merging partitions of its vertex set.

Definition 3.1 (**Quotient set** ($\mathcal{Q}(H)$) [BBD+25])**.** *Let H be a hypergraph and $\tau = \{V_1, ..., V_l\}$ a partition of its vertices. For a set of vertices $X \subseteq V$, the quotient of X under τ is $X/\tau = \{V_i \in \tau : |V_i \cap X| > 0\}$. For a hypergraph H, the quotient of H under τ is the hypergraph $H/\tau = (V(H)/\tau, \{e/\tau : e \in E(H)\})$. $\mathcal{Q}(H)$ is the set of all the quotients of H.*

Note that the image of every homomorphism of H in a graph G corresponds to a subhypergraph of some pattern in the quotient set of H. [BBD+25] showed that like in the general graph case, this can be used to express the number of subhypergraphs of H as a linear combination of the number of homomorphisms of all the patterns in its quotient set.

Lemma 3.2. *[BBD+25] For all hypergraphs H there is a function $\gamma : \mathcal{Q}(H) \to \mathbb{Q}$ such that, for all hypergraph G we have:*

$$\text{Sub}_H(G) = \sum_{H' \in \mathcal{Q}(H)} \gamma(H')\text{Hom}_{H'}(G)$$

Moreover, for all $H' \in \mathcal{Q}(H)$, $\gamma(H') \neq 0$.

3.3 Induced Subhypergraphs and Induced Trimmed Subhypergraphs

Definition 3.3 (Induced subhypergraph)**.** *For any subset $S \subseteq V(G)$, the induced subhypergraph of G on S is the hypergraph $G(S) = (S, E(S))$ with $E(S) = \{e \in E(H) \mid e \subseteq S\}$.*

We can obtain a more *relaxed* definition of an induced subhypergraph by including hyperedges that are only partially included in the vertex set of the subhypergraph. This was called a shrinkage in [SS21] and a trimmed subhypergraph in [BBD+25]. We will refer to it as the latter in this paper.

Definition 3.4 (Induced Trimmed Subhypergraph)**.** *For any subset $S \subseteq V(G)$, the induced trimmed subhypergraph of G on S is the hypergraph $G\langle V\rangle = (S, E\langle S\rangle)$ with $E\langle S\rangle = \{e \cap S \mid e \in E(H),\ e \cap S \neq \emptyset\}$.*

Note that, if H is a graph, then all induced trimmed subhypergraphs are also induced subhypergraphs. Let $\mathcal{H}$ be a set of hypergraphs, we say that the hypergraph H is **$\mathcal{H}$-ITS free** (Induced trimmed subhypergraph free) if no induced trimmed subhypergraph of H is isomorphic to H'.

We can also define intermediary notions of induced subhypergraphs where we allow edges that contain only a limited number of vertices that are not part of the vertex set of the subhypergraph. We call this an induced l-trimmed subhypergraph.

Definition 3.5 (Induced l-trimmed subhypergraph)**.** *Let $l \in \mathbb{Z}^+ \cup \{\infty\}$. For any subset $S \subseteq V(G)$, the induced l-trimmed subhypergraph of G on S is the hypergraph $G\langle S\rangle_l = (S, E\langle S\rangle_l)$ with $E\langle S\rangle_l = \{e \cap S \mid e \in E(G), |e \setminus S| \leq l, e \cap S \neq \emptyset\}$.*

Note that an induced subhypergraph is equivalent to an induced 0-trimmed subhypergraph, and a trimmed subhypergraph corresponds to a ∞-trimmed subhypergraph. Additionally, note that for a fixed subset $S \subseteq V$, $E\langle S\rangle_l \subseteq E\langle S\rangle_{(l+1)}$.

3.4 Directed Acyclic Hypergraphs

Like in the standard graph homomorphism counting problem, we will be exploiting specific orientations of the input graph to compute the number of homomorphisms. Hence, we need to define directed hypergraphs. In the most common definition of a **directed hypergraph** [AL17], each hyperedge is defined by two sets of vertices: the tail and the head; this directly generalizes directed graphs where, for every directed edge, one vertex is the tail and another the head.

For our purposes, we will think of directed hypergraphs as given by an ordering of its vertices. Each directed hyperedge will hence be given by an ordered list of vertices, which matches the ordering of the hypergraph. Directed hypergraphs defined this way are equivalent to directed acyclic graphs, and we will call them Directed Acyclic Hypergraphs. A **Directed Acyclic Hypergraph (DAH)** $\vec{G}$, is a hypergraph H with an ordering of its vertices $\pi : V(G) \to [V(G)]$, every hyperedge $e \in E(G)$ becomes an ordered list in $E(\vec{G})$, where the order of each hyperedge follows the ordering of G.

We use $\Sigma(H)$ to denote the set of acyclical orientations of H, up to isomorphism.

For a directed acyclic hypergraph, a **source** is a vertex that is first in the ordering of all hyperedges to which it belongs. We use $S(\vec{H})$ to denote the set of sources of the directed acyclic hypergraph $\vec{H}$.

For a directed acyclic hypergraph $\vec{H}$ we define its **skeleton** $Sk_0(\vec{H})$ as the directed acyclic graph obtained by replacing every directed hyperedge by an out-star centered at the first vertex of the edge, that is, a set of directed edges connecting the first vertex in the ordering of the hyperedge to every other vertex in it.

Similarly, we can define the **l-skeleton** $Sk_l(\vec{H})$ of $\vec{H}$ as the graph obtained by replacing every directed hyperedge e by a set of directed edges connecting each of the first $l+1$ vertices in e to all vertices that appear after it in the internal ordering of e. See Fig. 7 for an example.

Note that the ∞-skeleton of a hypergraph $\vec{H}$ corresponds to the acyclic orientation of the clique completion of $\vec{H}$ that matches the vertex ordering of $\vec{H}$. Note that for all l, the set of sources of a hypergraph and its l-skeletons are the same. $S(\vec{H}) = S(Sk_l(\vec{H}))$.

Given two directed acyclic hypergraphs $\vec{H}, \vec{G}$, a **DAH homomorphism** is a mapping $\phi : V(\vec{H}) \to V(\vec{G})$, such that for every directed hyperedge $e \in E(\vec{H})$ we have a $\phi(e) \in E(\vec{G})$. Note that because the directed hyperedges are ordered lists, they will necessarily preserve arity. Moreover, note that for $e = (v_1, v_2, ..., v_{|e|})$, we will have $\phi(e) = (\phi(v_1), \phi(v_2), ..., \phi(v_{|e|})$, and the internal hyperedge ordering will also be preserved.

3.5 Reachability and outdegree

The concept of reachability in graphs is central to the homomorphism counting techniques based on DAG-treewidth (which we will define next). In a directed graph, we say that a vertex u **reaches** v if there is a direct path connecting u to v. However, in a directed hypergraph, we can think of reachability in different ways.

Using the standard definition of directed hypergraphs, reaching the tail set of a hyperedge allows one to reach the vertices of the head. The most naive way to think about reachability in a directed acyclic hypergraph $\vec{H}$ would be to have the first vertex in the internal ordering of an edge reach all the other vertices in the edge; this would match the reachability of the 0-skeleton of $\vec{H}$. It is also equivalent to the standard notion of reachability in hypergraphs when setting the first vertex in each hyperedge as tail and the rest as head.

Furthermore, we can consider alternative notions of reachability by instead allowing the first $l+1$ vertices in a hyperedge reach the vertices appearing after it. Note that this would be equivalent to the reachability of the l-skeleton of the directed acyclic hypergraph.

We say that a vertex u l-**reaches** v in a DAH $\vec{H}$, if u reaches v in the l-skeleton of $\vec{H}$.

In a directed acyclic graph $\vec{G}$, we will use $Reach_{\vec{G}}(s)$ for the set of vertices reachable by s in $\vec{G}$, and $Reach_{\vec{G}}(S)$ for the union of $Reach_{\vec{G}}(s)$ for all $s \in S$. Moreover, we will use $\vec{G}[s]$ and $\vec{G}[S]$ for the subgraph of G induced by $Reach_{\vec{G}}(s)$ and $Reach_{\vec{G}}(S)$ respectively.

Similarly for a DAH $\vec{H}$, we will $Reach_{\vec{H},l}(s)$ (or $Reach_{\vec{H},l}(s)$ if clear from the context) for the set of vertices l-reachable by s in $\vec{H}$. Note that $Reach_{\vec{H},l}(s) = Reach_{Sk_l(\vec{H})}(s)$. We use $Reach_{\vec{H},l}(S)$ for the union of $Reach_{\vec{H},l}(s)$ for all $s \in S$. And we will use $\vec{H}_l[s]$ and $\vec{H}_l[S]$ for the (0-trimmed) subhypergraphs of G induced by the vertices in $Reach_{\vec{H},l}(s)$ and $Reach_{\vec{H},l}(S)$ respectively.

Similarly to how we did for reachability, we define the **l-outdegree** of a vertex v in a DAH $\vec{H}$, $d^+_{\vec{H},l}(v)$ as the outdegree in the l-skeleton. We will use $\Delta^+(\vec{H})$ to denote the maximum l-outdegree of $\vec{H}$.

3.6 The DAG-tree decomposition

Bressan [Bre19, Bre21] introduced the concept of DAG-tree decomposition of a directed acyclic graph to find efficient algorithms for homomorphism counting in degenerate graphs.

Definition 3.6 (DAG-tree decomposition [Bre19])**.** *For a given DAG $\vec{H}$, a DAG-tree decomposition of $\vec{H}$ is a rooted tree $T = (\mathcal{B}, \mathcal{E})$ such that:*

- *Each node of the tree $B \in \mathcal{B}$ is a subset of the sources of $\vec{H}$, $B \subseteq S(\vec{H})$.*
- *Every source $s \in S(\vec{H})$ appears in at least one node $B \in \mathcal{B}$, $\bigcup_{B\in\mathcal{B}} B = S(\vec{H})$.*
- *$\forall B, B_1, B_2 \in \mathcal{B}$. If B is in the (unique) path between B_1 and B_2 in T, then $Reach_{\vec{H}}(B_1) \cap Reach_{\vec{H}}(B_2) \subseteq Reach_{\vec{H}}(B)$.*

The DAG-treewidth $\tau(T)$ of a DAG-tree decomposition T is the maximum size among all the nodes in $\mathcal{B}$. The DAG-treewidth $\tau(\vec{H}$ of a DAG $\vec{H}$ is the minimum $\tau(T)$ of all valid DAG-tree decomposition T of $\vec{H}$. There exists a relation between DAG-treewidth and α-acyclicity.

Definition 3.7 (α-acyclicity)**.** *A hypergraph H is α-acyclic if there exists a tree T whose vertices are the hyperedges of F, such that, for all $e_1, e_2, e_3 \in E(H)$ such that e_2 is in the unique path of T between e_1 and e_3, we have $e_1 \cap e_3 \subseteq e_2$.*

The **reachability hypergraph** of a dag $\vec{H}$, is the hypergraph H' on vertex set $V(H') = \vec{H}$ that contains, for every source $s \in S(\vec{H})$ a hyperedge $e_s = Reach_{\vec{H}}(s)$. A dag $\vec{H}$ having $\tau(\vec{H}) = 1$ is equivalent to its reachability hypergraph being α-acyclic.

Lemma 3.8. *A directed acyclic $\vec{H}$ graph has $\tau(\vec{H})$ iff the reachability hypergraph of $\vec{H}$ is α-acyclic.*

We can generalize the concept of DAG-tree decomposition to directed acyclic hypergraphs. Note that the definition of DAG-tree decomposition only depends on the reachability function of the graph. We could define multiple notions of DAG-tree decompositions depending on which l-reachability of the DAH $\vec{H}$ we are using, however this turns out to be equivalent to computing

DAG-tree decomposition(s) for each possible l-skeleton of $\vec{H}$. Therefore, we will instead define notions of DAG-treewidth using the DAG-tree decompositionof the l-skeletons.

Definition 3.9 (τ_l)**.** *Let $l \in \mathbb{Z}^+ \cup \{\infty\}$. The l-DAG-treewidth of a directed acyclic hypergraph $\vec{H}$ is the DAG-treewidth of its l-skeleton.*

$$\tau_l(\vec{H}) = \tau(Sk_l(\vec{H}))$$

The l-DAG-treewidth of an undirected hypergraph H is the maximum l-DAG-treewidth across all DAH $\vec{H} \in \Sigma(H)$.

The **down-closure** of a bag B in a DAG-tree decomposition $\mathcal{T}$, $\Gamma(B)$, is the union of all the bags in the sub-tree of $\mathcal{T}$ rooted at B. We will use $\vec{H}[\Gamma(B)]_l$ for the subhypergraph of H induced by all the vertices in $Reach_{\vec{H},l}(\Gamma(B))$.

3.7 Degeneracy

The degeneracy of a graph is a measurement of its sparsity. A graph G is κ-degenerate if all its subgraphs have minimum degree of at most κ, while the degeneracy of G is the maximum κ such that G is κ-degenerate. A similar notion exists for hypergraphs [KR06].

Definition 3.10 (Degeneracy)**.** *We say that a hypergraph H is κ-degenerate if every induced subhypergraph of H has a minimum degree of at most κ. The degeneracy $\kappa(H)$ of a hypergraph H is the minimum value of κ such that H is κ-degenerate.*

We can define additional notions of degeneracy by using induced l-subhypergraphs instead of induced subhypergraphs.

Definition 3.11 (l-degeneracy)**.** *We say that a hypergraph is κ-l-degenerate if every induced l-trimmed subhypergraph of G has a minimum degree of at most κ. The l-degeneracy $\kappa_l(G)$ of a hypergraph H is the minimum value of κ such that H is κ-degenerate.*

We have $\kappa(G) = \kappa_0(G)$. Also, if G has bounded rank, the $\kappa_l(G)$ will be bounded by $O(\kappa(G^c))$. Furthermore, note that the l-degeneracy increases with l, since the induced l-trimmed subhypergraphs are subhypergraphs of the induced $(l+1)$-trimmed subhypergraphs. Thus, for all l, $\kappa_l(G) \leq \kappa_l(G)$.

4 The degeneracy orientation of hypergraphs

A property of degenerate graphs is that one can obtain an ordering of the vertices such that orienting the graph using such ordering yields a directed acyclic graph with maximum outdegree equal to the degeneracy. Moreover, such an orientation can be constructed in linear time [MB83].

We can show that this classical result can be extended to all the notions of l-degeneracy that we have defined, giving orientations with maximum l-outdegree bounded by the l-degeneracy. We will use $\vec{G}^{(l)}$ to denote the l-degeneracy orientation of G.

Lemma 2.2. *Let $l \in \mathbb{Z}^+ \cup \{\infty\}$ and let G be a hypergraph with l-degeneracy $\kappa_l(G)$ and rank $r(G)$. There exists an ordering of the vertices of $V(G)$ such that the directed acyclic hypergraph $\vec{G}^{(l)}$ obtained by applying this ordering to G has a maximum l-outdegree $\Delta_l^+(\vec{G}^{(l)}) \leq \kappa_l(G) \cdot r(G)$. Moreover, given G, one can construct $\vec{G}^{(l)}$ in $O(n + m(\log m \cdot r(G) + r(G)^2))$ time.*

Algorithm 1 *Compute_ordering*(G, l)

1: Set $G' = G$, $V' = V(G)$.
2: **for** $i \in [n]$ **do**
3: Set v_i as a vertex of G' with minimum degree
4: Set $V' = V \setminus \{v_i\}$
5: Set $G' = G\langle V' \rangle_l$
6: Return $v_0, ..., v_{n-1}$

Proof. Fix l. We can show that algorithm *Compute_ordering* (Alg. 4), gives the desired ordering. Moreover, we show how to implement it efficiently. The algorithm follows a similar structure to the original ordering algorithm from [MB83].

- Step 3: We use the degree structure from [MB83]: for each possible degree, we create a doubly linked list of vertices of degree i for each i, which we initialize putting each vertex in the appropriate list based on their degree in G. This structure allows to find a vertex v_i with minimum degree at any point in time $O(d(v_i))$, where $d(v_i)$ is the degree at the time of deletion. It also allows us to update the degree of a vertex in constant time. We can initialize the structure in $O(n + m)$ time.

- Step 4: Updating the vertex set can be done in constant time, by maintaining an array of active vertices.

- Step 5: We need to update G' by removing v_i from each hyperedge. To do this, we keep a dictionary that maps a tuple of vertices to each active hyperedge, this represents the trimmed portion of each hyperedge left in the induced l-trimmed subhypergraph. Additionally, for each hyperedge, we keep a counter of how many of its vertices have been removed.

  We iterate over all the hyperedges $e \in E(G)$ that contain v_i (at most $d_G(v_i)$) and increase the counter for all of them. If any hyperedge has lost more than l vertices, then we remove it and update the degrees of the vertices on it (at most $r(e)$). Otherwise we compute the tuple corresponding to $e \cap V'$ [10], this can be done in $O(r(e))$ time. If that tuple is not in the dictionary, we add it. Otherwise, it means that the hyperedge is already present in G', we use the dictionary to retrieve the conflicting hyperedge, and keep the one with lower counter of deleted vertices, deleting (and updating the degrees) of the other, which will take $O(r(e))$ time.

  The entire process of step 5 will take $O(r(e) + \log m)$ per hyperedge, where $\log m$ is the dictionary cost.

Therefore, steps 3-5 will take a total of $d_G(O(v_i)(r(G) + \log m))$ time. The sum of degrees in a hypergraph is bounded by the rank times the number of hyperedges, therefore, the total runtime of the algorithm will be bounded by $O(n + m(\log m \cdot r(G) + r(G)^2))$.

Now, let $\vec{G}^{(l)}$ be the directed acyclic hypergraph obtained from G and using the deletion ordering $v_0, ..., v_{n-1}$. We will have the following.

$$d^+_{\vec{G}^{(l)},l}(v_i) \leq d_{G\langle\{v_i,...,v_n\}\rangle_l}(v_i) \cdot r(G) \leq \kappa_l(G) \cdot r(G) \tag{1}$$

[10] We can use an arbitrary total order on the vertices so the tuple is unique for each possible subset.

The first inequality holds from the fact that the l-outdegree of the vertex v_i will be bounded by its degree in the induced l-trimmed subhypergraph at the time of deletion times the rank of the hypergraph. The second inequality holds because v_i is a vertex with minimum degree in $G\langle\{v_i, ..., v_n\}\rangle_l$, and by definition the minimum degree of any induced l-trimmed subhypergraph is bounded by $\kappa_l(G)$. Finally, we will have:

$$\Delta^+(\vec{G}^{(l)}) = \max_{v \in V(G)} d^+_{\vec{G}^{(l)},l}(v_i) \leq \kappa_l(G) \cdot r(G) \tag{2}$$

□

5 Counting hypergraph homomorphisms in degenerate hypergraphs

5.1 Reducing to directed homomorphisms

Recall from the definition of homomorphism in hypergraphs that we are allowed to map hyperedges to other hyperedges of smaller arity. If both the pattern and the input have the exact same arity for all their hyperedges, then this will not be an issue, but when considering hyperedges of any arity, this can become problematic.

Definition 5.1 ($\mathcal{Q}^c(H)$)**.** *For a hypergraph H, the contract set $\mathcal{Q}^c(H) \subseteq \mathcal{Q}(H)$, is the set of all the quotient hypergraphs H/τ generated by a partition $\tau = \{V_1, V_2, ...V_l\}$ such that every set V_i forms a connected component in H.*

Alternatively, one can generate $\mathcal{Q}^c(H)$ by recursively merging two vertices in the same hyperedge. Note that every homomorphism from H to an input hypergraph G will correspond to one homomorphism in $\mathcal{Q}^c(H)$; hence, we can obtain $\mathrm{Hom}_H(G)$ by computing the number of arity-preserving homomorphisms for each pattern in $\mathcal{Q}^c(H)$.

Lemma 5.2. *For all graph H:*

$$\mathrm{Hom}_H(G) = \sum_{H' \in \mathcal{Q}^c(H)} \beta(H')\mathrm{rHom}_{H'}(G)$$

Where $\beta(H') > 0$ for all H' and can be computed in $O(1)$ time.

Proof. Let ϕ be a homomorphism from H to G, for every vertex v in the image of ϕ, let $V_v = \{u \in V(H) : \phi(u) = v\}$, divide each set V_v into connected components of $H\langle V_v\rangle$ and set $\tau = \{V_1, ..., V_l\}$ be the set of all connected components. Note that τ is a valid partition of $V(H)$, and $H' = V/\tau \in \mathcal{Q}^c(H)$. Let $\phi' : \tau \to V(G)$ by assigning v to every set obtained from V_v.

We can show that ϕ' must be an arity preserving homomorphism from H/τ to G. Consider any hyperedge e in H/τ, every vertex V_i, V_j in e must have a different image $\phi'(V_i) \neq \phi'(V_j)$, as otherwise they would be part of the same connected component of V_v and would not have been split into different sets. Moreover, e corresponds to at least a hyperedge e' in H, and $\phi'(e) = \phi(e') \in E(G)$.

Let $\alpha(H')$ be the number of valid partitions τ of H such that H/τ is isomorphic to H'. For each arity preserving homomorphism of H' and each such τ we can obtain a homomorphism $\phi : H \to G$ by setting $\phi(u) = \phi'(V_i \ni u)$ for each u, moreover, for each automorphism (map of H' to itself that yields an isomorphic hypergraph) of H' we will have a homomorphism ϕ' that generates the same maps ϕ. Therefore, we can set $\beta(H') = \alpha(H')/Aut(H')$ to obtain the expression of the lemma. □

We can show that the patterns in $\mathcal{Q}^c(H)$ cannot have a greater l-DAG-treewidth than H. Note that this is not true in general for the entire set of quotients of H, $\mathcal{Q}(H)$.

Lemma 5.3. *Let $l \in \mathbb{Z}^+ \cup \{\infty\}$ and H be a hypergraph. For every hypergraph $H' \in \mathcal{Q}^c(H)$ $\tau_l(H') \leq \tau_l(H)$.*

Proof. Fix l. Let H be a hypergraph with $\tau = \tau_l(H)$. We show that all hypergraphs H' in $\mathcal{Q}^c(H)$ have $\tau_l(H') \leq \tau$, we do induction on the number of vertices of H'. The base case, $|V(H')| = |V(H)|$, is trivial as $H' = H$, and therefore $\tau_l(H') = \tau$.

For the inductive step, assume that for all hypergraphs H' in $\mathcal{Q}^c(H)$ with k vertices we have $\tau_l(H') \leq \tau$, we show that any hypergraph H' in $\mathcal{Q}^c(H)$ with $k-1$ vertices will also have $\tau_l(H') \leq \tau$.

Fix a hypergraph H' in $\mathcal{Q}^c(H)$ with $|V(H')| = k - 1$. We show that any orientation $\vec{H'}$ of H' has $\tau_l(\vec{H'}) \leq \tau$.

Let $H'' \in \mathcal{Q}^c(H)$ with $|V(H'')| = k$ be a hypergraph such that merging the vertices $u, v \in V(H'')$ results in H'. Such a hypergraph must exist; otherwise $V(H')$ would not be in $\mathcal{Q}^c(H)$. Moreover, from the induction hypothesis, we have $\tau_l(V(H'')) \leq \tau$. We call uv to the vertex in H' resulting from merging u and v.

Let $\pi' : V(H') \to [k-1]$ be the ordering of the vertices of H' according to $\vec{H'}$. We define $\pi'' : V(H'') \to [k]$ as follows:

$$\pi''(w) = \begin{cases} \pi'(w) & \text{if } \pi'(w) < \pi'(uv) \\ \pi'(uv) & \text{if } w = u \\ \pi'(uv) + 1 & \text{if } w = v \\ \pi'(w) + 1 & \text{if } \pi'(w) > \pi'(uv) \end{cases} \tag{3}$$

Let $\vec{H''}$ be the directed acyclic hypergraph obtained from H'' using the ordering π''. Consider the l-skeleton of $\vec{H'}$ and $\vec{H''}$, $Sk' = Sk_l(\vec{H'})$ and $Sk'' = Sk_l(\vec{H''})$. Let $\mathcal{T}''$ be a DAG-tree decomposition of Sk'' with $\tau(\mathcal{T}'') \leq \tau$, such DAG-tree decomposition must exist from the definition of τ_l. We show that Sk' admits a DAG-tree decomposition $\mathcal{T}'$ with $\tau(\mathcal{T}') \leq \tau$. The direct arc (u, v) might or might not be present in Sk''. We consider each case separately.

$(\mathbf{u}, \mathbf{v}) \notin \mathbf{E}(\mathbf{Sk''})$ In this case, Sk' will be the result of merging u and v in Sk''. Note that u and v share at least one hyperedge $e \in E(\vec{H''})$. Let s be the source of e, the direct arcs (s, u) and (s, v) must both be present in Sk''. We can show that $\mathcal{T}''$ will also be a valid DAG-tree decomposition for Sk', and thus $\tau(Sk) \leq \tau$. It suffices to show that for every vertex w of Sk', the sub-tree of $\mathcal{T}''$ induced by the sources that reach w is connected:

- $w = uv$. w will be reached by both the bags that were reaching u and the bags that were reaching v. Note that the bags of sources reaching u will form a connected sub-tree, similarly with the bags of sources reaching v. Moreover, both sub-trees intersect in the bags of sources that can reach s (or that contain s if s is a source of the skeleton), which means that they must form a single connected sub-tree.

- $w \in Reach_{Sk''}(u)$. w will be reached by the same bags as in Sk'' plus all the bags that were reaching v. Both will form connected sub-trees in $\mathcal{T}''$, and again they must intersect in at least the bags that reach s, which means that they will form a connected sub-tree.

- $w \in Reach_{Sk''}(v)$. Analogous to the previous case.
- $w \notin (Reach_{Sk''}(u) \cup Reach_{Sk''}(c))$. In this case the set of sources reaching w in Sk' is the same as in Sk'', and therefore they will form a connected sub-tree.

$(\mathbf{u}, \mathbf{v}) \in \mathbf{E}(\mathbf{Sk''})$ In this case Sk' is the result of merging the direct arc (u, v). Moreover, it is possible that the merge might create new direct edges in Sk'. This can happen if the vertices u, v are part of a hyperedge e in $\vec{H}''$ and v is one of the first $l+1$ vertices in the internal ordering of e and $|e| > l+2$. In that case, $\vec{H}'$ will contain a hyperedge e' where a vertex u' that was in position $l+2$ in e is now in the position $l+1$ in the internal ordering of e', this means that for all v' appearing after u' in e' we will have a direct arc $(u', v') \in Sk'$ that might not have been present in Sk''.

We show that both merging the direct edge and adding a direct edge between two vertices on the same hyperedge do not increase τ.

- **Merging a direct arc:** We need to consider three sub-cases:
  1. u is a source in Sk'' and uv is a source in Sk'. Replacing u for uv in every bag of $\mathcal{T}''$ will result in a valid DAG-tree decomposition $\mathcal{T}'$ of Sk', as $Reach_{Sk'}(uv) \setminus \{uv\} = Reach_{Sk''}(u) \setminus (u)$, and every other vertex will have the same reachability in both dags. No bag will increase in size and therefore $\tau(\mathcal{T}') \leq \tau(\mathcal{T}'') \leq \tau$.
  2. u is a source in Sk'' but uv is not a source in Sk'. This implies that v had some incoming edges in Sk'', all the sources other than u that reached v in Sk'' now also reach u and all the vertices in $Reach_{Sk''}(u)$. Note that in $\mathcal{T}''$ all bags containing u must form a connected sub-tree, similarly all bags containing a source (including u) that reaches v will form a connected sub-tree. Let s be a source in Sk'' that reaches v, and either appears in a bag of $\mathcal{T}''$ together with u, or is in a bag adjacent to a bag containing u.

     We can form a DAG-tree decomposition $\mathcal{T}'$ of Sk' by replacing u by s in every bag. No bag will increase in size and therefore $\tau(\mathcal{T}') \leq \tau(\mathcal{T}'') \leq \tau$. We can verify that this will be a valid DAG-tree decomposition, let w be a vertex in $V(Sk')$, we can show that the sub-tree of $\mathcal{T}'$ that contains a source reaching w is connected.
     - If $w = uv$, then the bags that reach uv in Sk' are the same bags that reached v in Sk'', which must form a connected sub-tree.
     - If $w \in Reach_{Sk''}(u)$, then w is reachable by every bag that reached v in Sk'' which forms a connected sub-tree, and every bag that reaches w in Sk'', but those bags must form a connected sub-tree with the bags of $\mathcal{T}''$, both sub-trees overlap in the bags that were containing u (now s instead), and therefore they will all form a bigger connected sub-tree.
     - If $w \notin Reach_{Sk''}(u)$, then all bags that reached w in $\mathcal{T}''$ also reach w in $\mathcal{T}'$, if s reaches w then there might be some additional bags reaching w, but they will still form a connected sub-tree.
  3. u is not a source in Sk''. In this case we can show that $\mathcal{T}''$ will be a valid DAG-tree decomposition for Sk'. Let w be a vertex in $V(Sk')$:
     - If $w = uv$, then the bags that reach w in Sk' are the exact same bags that reached v in Sk'', and therefore form a connected sub-tree.

- If $w \in Reach_{Sk''}(u)$, note that all the sources reaching v in Sk'' now will reach w in Sk'. These sources form a connected sub-tree. All the sources that reached w in Sk'' will also reach w in Sk'' and therefore also form a connected sub-tree. Both sub-trees intersect in the bags containing sources that reach u in Sk'', thus combining them yields a larger connected sub-tree.
- If $w \notin Reach_{Sk''}(u)$, then the sources that reach w in Sk' are the same sources that reached w in Sk'', and therefore the bags containing those sources will form a connected sub-tree.

- **Adding a direct edge:** We show that adding a direct edge between two vertices that are part of the same hyperedge does not increase τ. Let $\vec{F}$ and $\vec{F'}$ be two dags with the same vertex set and $E(\vec{F'}) = E(\vec{F}) + \{(u, v)\}$ for some vertices u, v such that $\exists s \in V(\vec{F})$ with $(s, u), (s, v) \in E(\vec{F})$. Let $\mathcal{T}$ be a DAG-tree decompositionof $\vec{F}$, we can show that it will also be a valid DAG-tree decompositionof $\vec{F'}$.

  Let w be a vertex of $\vec{F'}$, we show that the bags reaching w in $\vec{F'}$ must form a connected sub-tree:

  - If $w \in Reach_{\vec{F}}(v)$, then w will be reached in $\vec{F'}$ by the bags that reached v and u. Both sets of bags form connected sub-trees in $\mathcal{T}$ and intersect on the bags that reach s, therefore, they will all form a connected sub-tree.
  - If $w \notin Reach_{\vec{F}}(v)$, then the bags reaching w are the same in $\vec{F}$ and $\vec{F'}$, therefore forming a connected sub-tree in $\mathcal{T}$.

  Therefore, no matter how many direct edges Sk' adds with respect to Sk'' the DAG-tree decomposition will still be valid and we will have that $\tau(Sk') \leq \tau(Sk'')$.

□

The next step is to exploit the degeneracy orientations of the input hypergraph. We will compute the degeneracy order of G and obtain a directed acyclic hypergraph $\vec{G}^{(l)}$ with $\Delta_l^+(\vec{G}^{(l)}) = O(\kappa_l(G))$. Note that the image of each homomorphism of H in G appears now in a specific acyclic orientation, and corresponds to one of the possible directed acyclic hypergraphs that can be obtained from H. Hence, analogously to the graph case, we can compute the number of undirected hypergraph homomorphisms as the sum of the directed homomorphism for each orientation of H.

$$\mathrm{rHom}_H(G) = \sum_{\vec{H} \in \Sigma(H)} \mathrm{Hom}_{\vec{H}}(\vec{G}) \tag{4}$$

5.2 Counting directed hypergraph homomorphisms

In this section, we show how to generalize Bressan's algorithm [Bre19, Bre21] to our different notions of degeneracy and DAG-treewidth to compute the number of DAH homomorphisms.

Let $\vec{H}$ and $\vec{G}$ be two directed acyclic hypergraphs. We could think of using the skeletons of $\vec{H}$ and $\vec{G}$ to compute the number of homomorphisms from $\vec{H}$ to $\vec{G}$. However, while for every valid homomorphism $\phi : V(\vec{H}) \to V(\vec{G})$ from $\vec{H}$ to $\vec{G}$ is also a valid homomorphism from $Sk_l(\vec{H})$ to $Sk_l(\vec{G})$, the opposite is not true. We could think of obtaining the homomorphisms from the skeletons and then just filtering out the ones that are not valid. This again does not directly work,

as we would require to *enumerate* the homomorphisms, rather than just counting, which can be potentially harder.

Alternatively, one could think of just computing the homomorphisms of the induced l-trimmed subhypergraph of each of the partitions of the DAG-tree decomposition, but those homomorphisms will not match the homomorphisms of the entire pattern. Alternatively, one could try th induced (0-trimmed) subhypergraphs, but while the skeletons of the partitions and the l-trimmed subhypergraphs will always be connected, the induced subhypergraphs might not, and therefore we may not be able to compute the homomorphism list as efficiently.

Therefore we will need to do a hybrid approach, we will enumerate all the homomorphisms of the skeletons for each partition of the DAG-tree decomposition, we will then filter that list by checking that every hyperedge of the corresponding induced subhypergraph is present. The result of this is a list of homomorphisms of the induced subhypergraphs for each of the partitions; this list does not include all such homomorphisms, as, as we said, the induced subhypergraphs of the partitions might be disconnected. But we can show that it includes all the homomorphisms that *matter*, that is, the ones that can be extended to valid homomorphisms of the entire pattern H.

The proof follows a similar structure to the original algorithm from Bressan (see Section 3 in [Bre21]). We start by introducing the following lemma.

Lemma 5.4 (Lemma 4 in [Bre21], restated)**.** *Let $\vec{H}, \vec{G}$ be two directed acyclic graphs. Given any $B \subseteq S(\vec{H})$, the set of homomorphisms $\Phi(\vec{H}[B], \vec{G})$ has size $poly(d)n^{|B|}$ and can be enumerated in time $poly(d)n^{|B|}$. Where $k = |V(\vec{H})|$, $n = |V(\vec{G})|$ and $d = \Delta^+(\vec{G})$.*

Like we mentioned previously, we can filter the homomorphisms of the skeletons of each partition to obtain a subset of the DAH homomorphisms of the induced subhypergraphs of the same partition.

Lemma 5.5. *Let $\vec{H}, \vec{G}$ be two directed acyclic hypergraphs and $0 \leq l \leq k = |V(\vec{H})|$. Given any $B \subseteq S(\vec{H})$, we can compute a set of homomorphisms $\Phi' \subseteq \Phi(\vec{H}[B]_l, \vec{G})$ such that every $\phi \in \Phi(\vec{H}[B]_l, \vec{G}) \setminus \Phi'$ has $ext(\vec{H}, \vec{G}; \phi) = 0$ in time $poly(d_l)n^{|B|} + O(m)$. Moreover, $|\Phi'| = poly(d_l)n^{|B|}$. Where d_l is the maximum l-outdegree of $\vec{G}$.*

Proof. First we compute the l-skeletons of $\vec{H}$ and $\vec{G}$, this can be done in $O(m)$ time. Note that $\Delta^+(Sk_l(\vec{G})) = d_l$. We use Lemma 5.4 to compute $\Phi(Sk_l(\vec{H})[B], \vec{G})$ in time $poly(d_l)n^{|B|}$. We construct Φ' by filtering every homomorphism and ensuring that they are a valid homomorphism from $\vec{H}[B]_l$ to $\vec{G}$, this takes $O(1)$ time for homomorphism. The entire process will take $poly(d_l)n^{|B|} + m$ time.

We just need to show that every homomorphism in $\Phi(\vec{H}[B]_l, \vec{G}) \setminus \Phi'$ has extension equal to 0. Assume otherwise, there is a homomorphism $\phi' \in \Phi(\vec{H}[B]_l, \vec{G}) \setminus \Phi'$ with $ext(\vec{H}, \vec{G}; \phi') \neq 0$. Let ϕ be a valid homomorphism from $\vec{H}$ to $\vec{G}$ that extends ϕ', ϕ must also be a valid homomorphism from $Sk_l(\vec{H})$ to $Sk_l(\vec{G})$, and therefore its restriction to $\vec{H}[B]_l$, ϕ' must also be a valid homomorphism of the skeleton. But in that case ,we would have that $\phi' \in \Phi'$, reaching a contradiction.

□

We must also adapt Lemma 3 in [Bre21] to hypergraph homomorphisms. Recall $\Gamma(B)$ is the down-closure of B in $\mathcal{T}$.

Lemma 5.6. *Let $l \in \mathbb{Z}^+ \cup \{\infty\}$ and let $\mathcal{T}$ be a DAG-tree decomposition of $Sk_l(\vec{H})$ and B a node of $\mathcal{T}$. Fix $\phi : \vec{H}[B]_l \to \vec{G}$. For every child B' of B in $\mathcal{T}$, let $\phi_{B'}$ be the restriction of ϕ to the vertices of $Reach_{\vec{H},l}(B) \cap Reach_{\vec{H},l}(\Gamma(B'))$.*

$$ext(\vec{H}[\Gamma(B)]_l, \vec{G}; \phi) = \prod_{B' \in children(B)} ext(\vec{H}[\Gamma(B')]_l, \vec{G}; \phi_{B'})$$

Proof. Let $\Phi(\phi)$ be the set of homomorphisms from $\vec{H}[\Gamma(B)]_l$ to $\vec{G}$ that extend to ϕ. Similarly, let $\Phi_{B'}(\phi_{B'})$ be the set of homomorphisms from $\vec{H}[\Gamma(B')]_l$ to $\vec{G}$ that extend $\phi_{B'}$. We can show that there is a bijection between $\Phi(\phi)$ and $\bigtimes_{B' \in\ children(B)} \Phi_{B'}(\phi_{B'})$.

First, for any $\phi' \in \Phi(\phi)$ let $\phi'_{B'}$ be the restriction of ϕ' to $\vec{H}[\Gamma(B')]_l$. Note that $\phi'_{B'}$ must agree with $\phi_{B'}$, therefore $\phi'_{B'} \in \Phi_{B'}(\phi_{B'})$. Then we will have that the tuple $(\phi'_{B'} : B' \in\ children(B)) \in \bigtimes_{B' \in\ children(B)} \Phi_{B'}(\phi_{B'})$.

Conversely, let $(\phi'_{B'} : B' \in children(B)) \in \bigtimes_{B' \in\ children(B)} \Phi_{B'}(\phi_{B'})$, we show that the homomorphism ϕ' obtained by combining ϕ and $\phi'_{B'}$ for all $B' \in\ children(B)$ is in $\Phi(\phi)$. First, note that every homomorphism $\phi'_{B'}$ must agree with each other, since they can only intersect on the vertices of $\vec{H}[B]_l$, but the values of those vertices for all homomorphism will correspond to ϕ. Second, ϕ' is a mapping from $\vec{H}[\Gamma(B)]_l$ to $\vec{G}$ since every vertex in $\vec{H}[\Gamma(B)]_l$ is in $\vec{H}[B]_l$ or $\vec{H}[\Gamma(B')]_l$ for some child B' of B. Finally, we can show that every hyperedge in $\vec{H}[\Gamma(B)]_l$ is properly maintained in the homomorphism.

Let e be a hyperedge in $\vec{H}[\Gamma(B)]_l$ and s its source (first vertex in its internal ordering). We can distinguish two cases:

- $s \in Reach_{\vec{H},l}(B)$, in this case $e \subseteq Reach_{\vec{H},l}(B)$, and therefore $\phi'(e) = \phi(e) \in E(\vec{G})$.
- $s \in Reach_{\vec{H},l}(\Gamma(B'))$ for some $B' \in\ child(B)$, in this case $e \subseteq Reach_{\vec{H},l}(\Gamma(B'))$, and therefore $\phi'(e) = \phi'_{B'}(e) \in E(\vec{G})$.

Therefore, $\phi' \in \Phi(\phi)$. □

We now introduce the following algorithm, which is almost identical to Algorithm 1 in [Bre21] (Bressan's algorithm), the main difference being that we only ensure that homomorphisms with non-zero extension get counted accurately.

Lemma 5.7. *Let $\vec{H}$,$\vec{G}$ be any hypergraphs, $\mathcal{T} = (\mathcal{B}, \mathcal{E})$ a l-DAG-tree decomposition of $\vec{H}$ and $B \in \mathcal{B}$ any vertex from $\mathcal{T}$. Algorithm $HomCount_l(\vec{H}, \vec{G}, B, \mathcal{T})$ returns a dictionary that for all homomorphisms $\phi : H[B]_l \to \vec{G}$:*

- *If $ext(\vec{H}, \vec{G}; \phi) > 0$, then $C_B(\phi) = ext(\vec{H}[\Gamma(B)]_l, G); \phi)$.*
- *If $ext(\vec{H}, \vec{G}; \phi) = 0$, then $C_B(\phi) \leq ext(\vec{H}[\Gamma(B)]_l, G); \phi)$.*

Moreover, the algorithm runs in time $poly(d_l)n^{\tau_l(\mathcal{T})} \log n + O(m)$. Where d_l is the maximum l-outdegree of $\vec{G}$.

Proof. The runtime analysis is similar to the one from Lemma 5 in [Bre21], as the algorithm takes the same steps, we ignore dependencies on the size of H. We will need to run the algorithm recursively for each bag in $\mathcal{T}$, the construction of $\mathcal{T}$ depends only on H and therefore we will have $O(1)$ bags. Computing Φ' takes $poly(d_l)n^{|B|} + O(m)$ from Lemma 5.5, and we will have $|\Phi'| = poly(d_l)n^{|B|}$. For each element in Φ we will require $O(\log n)$ cost to access and write in the dictionary. Therefore, the complexity per iteration is $poly(d_l)n^{|B|} \log n + O(m)$, additionally it $|B|$

Algorithm 2 $HomCount_l(\vec{H}, \vec{G}, B, \mathcal{T})$

Compute Φ' as in Lemma 5.5.
Initialize C_B as an empty dictionary with default value of 0.
if B is a leaf of $\mathcal{T}$ **then**
    for every homomorphism $\phi \in \Phi'$ **do**
        $C_B(\phi) = 1$
else
    for every child B' of B in $\mathcal{T}$ **do**
        $C_{B'} = HomCount(\vec{H}, \vec{G}, B', \mathcal{T})$
        Initialize $AGG_{B'}$ as an empty dictionary with default value of 0.
        for every nonzero entry ϕ in $C_{B'}$ **do**
            Let ϕ_r be the restriction of ϕ to $Reach_{\vec{H},l}(B) \cap Reach_{\vec{H},l}(\Gamma(B'))$
            $AGG_{B'}(\phi_r) += C_{B'}(\phi)$.
    for every homomorphism $\phi \in \Phi'$ **do**
        for every child B' of B in $\mathcal{T}$ **do**
            Let $\phi_{B'}$ be the restriction of ϕ to $Reach_{\vec{H},l}(B) \cap Reach_{\vec{H},l}(\Gamma(B'))$.
        $C_B(\phi) = \prod_{B' \in \text{children}(B)} AGG_{B'}(\phi_{B'})$
Return C_B

is bounded by $\tau_l(\mathcal{T})$ for all B, therefore the algorithm will take $poly(d_l)n^{\tau_l(\mathcal{T})} \log n + O(m)$ total time.

We prove the correctness by induction on the nodes of the tree $\mathcal{T}$.

Base case: If B is a leaf then $\vec{H}[B]_l = \vec{H}[\Gamma(B)]_l$, note that from Lemma 5.5 we have that every homomorphism $\phi : \vec{H}[B]_l \to \vec{G}$ with $ext(\vec{H}, \vec{G}; \phi) > 0$ will be in Φ' and therefore the algorithm will set $C_B(\phi) = 1$ which means $C_B(\phi) = ext(\vec{H}[\Gamma(B)\_l, \vec{G}; \phi) = 1$. If $ext(\vec{H}, \vec{G}; \phi) = 0$ then either $\phi \in \Phi'$ and then $C_B(\phi) = ext(\vec{H}[\Gamma(B)]_l, \vec{G}; \phi) = 1$, or $\phi \notin \Phi'$ and then $C_B(\phi)$ just takes the default value of $0 < ext(\vec{H}[\Gamma(B)]_l, \vec{G}; \phi)$.

Inductive step: For the inductive step, assume that for every child B' of B in $\mathcal{T}$ we have $C_{B'}(\phi) = ext(\vec{H}[\Gamma(B')]_l, \vec{G}; \phi)$ for all $\phi \in \Phi(\vec{H}[B']_l, \vec{G})$ with $ext(\vec{H}, \vec{G}; \phi) > 0$ and $C_{B'}(\phi) \leq ext(\vec{H}[\Gamma(B')]_l, \vec{G}; \phi)$ otherwise. Let ϕ be a homomorphism in $\Phi(\vec{H}[B]_l, \vec{G})$ with $ext(\vec{H}, \vec{G}; \phi) \neq 0$, we can show that $C_B(\phi) = ext(\vec{H}[\Gamma(B)]_l, G); \phi)$.

Let B' be a child of B, and $\phi_{B'}$ the restriction of ϕ to $Reach_{\vec{H},l}(B) \cap Reach_{\vec{H},l}(\Gamma(B'))$, we can show that every homomorphism $\phi' \in \Phi(\vec{H}[B']_l, \vec{G})$ with $C_{B'}(\phi') \neq 0$ that agrees with $\phi_{B'}$ will also have non-zero extension, $ext(\vec{H}, \vec{G}; \phi') \neq 0$. Let ϕ'' be a homomorphism from $\vec{H}[\Gamma(B')]_l$ to $\vec{G}$ that agrees with ϕ', such homomorphism exists as $C_{B'}(\phi') \neq 0$. Let ϕ^* be a homomorphism from $\vec{H}$ to $\vec{G}$ that agrees with ϕ, such homomorphism exists as ϕ has non-zero extension, and $\phi^*_{B'}$ its restriction to $V(\vec{H}) \setminus (Reach_{\vec{H},l}(\Gamma(B')) \setminus Reach_{\vec{H},l}(B))$.

Note that $\phi^*_{B'}$ contains all the vertices in $Reach_{\vec{H},l}(B))$, and therefore agrees with ϕ and $\phi_{B'}$. Similarly, ϕ'' contains all the vertices in $(Reach_{\vec{H},l}(\Gamma(B'))$, and agrees with ϕ', which in turn agrees with $\phi_{B'}$. Let $V(\phi)$ be the set of vertices mapped by the homomorphism ϕ, we have the following:

$$V(\phi^*_{B'}) \cup V(\phi'') = (V(\vec{H}) \setminus (Reach_{\vec{H},l}(\Gamma(B')) \setminus Reach_{\vec{H},l}(B))) \cup (Reach_{\vec{H},l}(\Gamma(B')) = V(\vec{H}) \quad (5)$$

$$\begin{aligned} V(\phi^*_{B'}) \cap V(\phi'') &= (V(\vec{H}) \setminus (Reach_{\vec{H},l}(\Gamma(B')) \setminus Reach_{\vec{H},l}(B))) \cap (Reach_{\vec{H},l}(\Gamma(B')) \\ &= Reach_{\vec{H},l}(\Gamma(B')) \setminus (Reach_{\vec{H},l}(\Gamma(B')) \setminus (Reach_{\vec{H},l}(\Gamma(B')) \cap Reach_{\vec{H},l}(B))) \quad (6) \\ &= Reach_{\vec{H},l}(\Gamma(B')) \cap Reach_{\vec{H},l}(B) = V(\phi_{B'}) \end{aligned}$$

Note that $\phi^*_{B'}$ and ϕ'' intersect only in the vertices of $\phi_{B'}$, and we saw that both agree with $\phi_{B'}$. Therefore, we can combine $\phi^*_{B'}$ and ϕ'' into a homomorphism $\phi^T = \phi'' + \phi^*_{B'}$, and we can show that it must be a valid homomorphism from $\vec{H}$ to $\vec{G}$. First, from (5) we have that $V(\phi^T) = V(\phi^*_{B'}) \cup V(\phi'') = V(\vec{H})$. Only left to check is that every hyperedge is preserved in the image. Let e be a hyperedge in $\vec{H}$, and let s be the "source" of e, that is, the first vertex of its internal ordering. We distinguish two cases:

- $s \in Reach_{\vec{H},l}(\Gamma(B'))$, then $e \subseteq V(\phi'')$, and $\phi^T(e) = \phi''(e) \in E(\vec{G})$.
- $s \notin Reach_{\vec{H},l}(\Gamma(B'))$, then for every vertex $v \in e$ we have $v \in Reach_{\vec{H},l}(B)$, and therefore $v \in V(\phi^*_{B'})$; or $v \notin Reach_{\vec{H},l}(B)$, but then $v \notin Reach_{\vec{H},l}(\Gamma(B'))$ as otherwise it would not be a valid l-DAG-tree decomposition, which also means $v \in V(\phi^*_{B'})$. Thus, $e \subseteq V(\phi^*_{B'})$ and $\phi^T(e) = \phi^*_{B'}(e) \in E(\vec{G})$.

Therefore, ϕ^T is a valid homomorphism, which means $ext(\vec{H}, \vec{G}; \phi') \neq 0$. Thus, from the inductive hypothesis we will have that $C_{B'}(\phi') = ext(\vec{H}[\Gamma(B')]_l, \vec{G}; \phi')$. We will then have that:

$$AGG_{B'}(\phi_{B'}) = \sum_{\substack{\phi' \in C_{B'} \\ \phi' \text{ agrees } \phi_{B'}}} C_{B'}(\phi') = \sum_{\substack{\phi' : \vec{H}[B']_l \to \vec{G} \\ \phi' \text{ agrees } \phi_{B'}}} ext(\vec{H}[\Gamma(B')]_l, \vec{G}; \phi') = ext(\vec{H}[\Gamma(B')]_l, \vec{G}; \phi_{B'}) \quad (7)$$

The first equality comes from lines 9-12 in the algorithm, the second from the inductive hypothesis together with the previous explanation, and the third from the fact that every homomorphism contributing to $ext(\vec{H}[\Gamma(B')], \vec{G}; \phi_{B'})$ also contributes to exactly one of the $ext(\vec{H}[\Gamma(B')], \vec{G}; \phi')$. Therefore, for every ϕ with non-zero extension, we will have:

$$C_B(\phi) = \prod_{B' \in children(B)} ext(\vec{H}[\Gamma(B')]_l, \vec{G}; \phi_{B'}) = ext(\vec{H}[\Gamma(B)]_l, \vec{G}; \phi) \quad (8)$$

Here, the first equality comes from combining line 16 in the algorithm and (7), and the second equality comes from Lemma 5.6.

Now, let ϕ be a homomorphism in $\Phi(\vec{H}[B]_l, \vec{G})$ with $ext(\vec{H}, \vec{G}; \phi) = 0$, we can show that $C_B(\phi) \leq ext(\vec{H}[\Gamma(B)]_l, G); \phi)$. We can rewrite (7) by replacing the second equality with an inequality giving:

$$AGG_{B'}(\phi_{B'}) \leq ext(\vec{H}[\Gamma(B')]_l, \vec{G}; \phi_{B'}) \quad (9)$$

Using this instead in (8) will finally give:

$$C_B(\phi) \leq ext(\vec{H}[\Gamma(B)]_l, \vec{G}; \phi) \quad (10)$$

Completing the induction. □

5.3 Completing the proof

We finally have all the pieces for completing the proof of Theorem 2.3, which we restate.

Theorem 2.3. *Let $l \in \mathbb{Z}^+ \cup \{\infty\}$ and H, G be two bounded rank hypergraphs. There is an algorithm that computes* $\mathrm{Hom}_H(G)$ *in time* $\mathrm{poly}(\kappa_l) n^{\tau_l(H)} \log m$.

Proof. We start by computing the l-degeneracy ordering of G, and then create the directed acyclic hypergraph $\vec{G}^{(l)}$. From Lemma 2.2, this can be done in time $O(n + m(\log m \cdot r(G) + r(G)^2)) = O(n + m \log m)$ as the rank of G is bounded. Moreover, we have that the maximum l-outdegree of $\vec{G}^{(l)}$ is $\Delta_l^+(\vec{G}^{(l)}) \leq \kappa_l(G) \cdot r(G) = O(\kappa_l(G))$.

Using Lemma 5.2 and equation (4), we can compute $\mathrm{Hom}_H(G)$ by computing the value of $\mathrm{Hom}_{\vec{H}}(\vec{G})$ for all the directed acyclic hypergraphs in $\bigcup_{H' \in \mathcal{Q}^c(H)} \Sigma(H')$. The number of these instances is bounded by some function of k. Also, using Lemma 5.3 we have that for any $H' \in \mathcal{Q}^c(H)$, $\tau_l(H') \leq \tau_l(H)$. And therefore, each $\vec{H}$ for which we need to compute $\mathrm{Hom}_{\vec{H}}(\vec{G})$, will also have $\tau_l(\vec{H}) \leq \tau_l(H)$.

Finally, for each $\vec{H}$, we compute a l-DAG-tree decomposition $\mathcal{T}$ and root it arbitrarily in some node R. We then run $HomCount(\vec{H}, \vec{G}^{(l)}), R, \mathcal{T})$, because R is the root of $\mathcal{T}$, we will have that the output C_R is a dictionary such that for each homomorphism $\phi : \vec{H}[R]_l \to \vec{G}$:

- if $ext(\vec{H}, \vec{G}; \phi) > 0$, then $C_R(\phi) = ext(\vec{H}[\Gamma(R)]_l, \vec{G}; \phi) = ext(\vec{H}, \vec{G}; \phi)$.
- if $ext(\vec{H}, \vec{G}; \phi) = 0$, then $C_R(\phi) \leq ext(\vec{H}[\Gamma(R)]_l, \vec{G}; \phi) = ext(\vec{H}, \vec{G}; \phi) = 0$.

Therefore, summing over all entries of C_R we will obtain

$$\sum_{\phi \in C_R} C_R(\phi) = \sum_{\phi : \vec{H}[R]_l \to \vec{G}} ext(\vec{H}, \vec{G}; \phi) = \mathrm{Hom}_{\vec{H}}(\vec{G})$$

Running $HomCount(\vec{H}, \vec{G}^{(l)}), R, \mathcal{T})$ will take $poly(d_l) n^{\tau_l(\mathcal{T})} \log n + O(m)$, we will have $\tau_l(\mathcal{T}) \leq \tau_l(\vec{H}) \leq \tau_l(H)$, and $d_l = O(\kappa_l(G))$. Moreover, $m = O(\kappa_l(G) \cdot n)$. Therefore, we will need $poly(\kappa_l(G)) O(n^{\tau_l(H)} \log m)$ total time to compute $\mathrm{Hom}_H(G)$.

□

6 A characterization of patterns with $\tau_l = 1$

In this section, we prove Theorem 2.4.

We start by giving an alternative definition of the sets $\mathcal{H}_l$, this definition will help us prove the theorem. First, we define an l-connector gadget.

Definition 6.1 (l-connector gadget)**.** *Let H be a hypergraph. Let X and V be two disjoint sets of vertices of H, such that $X \cup V = V(H)$. We say that H is an l-connector gadget of V if:*

- *H is connected.*
- *Every hyperedge of H with at least two vertices in X has at least $l + 1$ vertices in V. $\forall e \in E(H') || e \cap X| \geq 2, |e \cap V| > l$.*

- *Let H' be the trimmed subhypergraph of H induced by V, there exists at least one connected component $V' \subset V$ in H', such that every vertex in X is a neighbor (shares a hyperedge in H) with a vertex in V'.*

Secondly, we define the sets $\mathcal{H}_l^k$. Basically, a hypergraph belongs to $\mathcal{H}_l^k$ if it can be split into a core V of k vertices and k l-connector gadgets, such that every subset of $k-1$ vertices has its own l-connector gadget.

Definition 6.2 ($\mathcal{H}_l^k$)**.** *We define $\mathcal{H}_l^k$ as the set of all hypergraphs H such that we can split the vertices of $V(H)$ into the set $X = \{x_0, .., x_{k-1}\}$ and k disjoint sets $V_0, ..., V_{k-1}$, with $X \cup \bigcup_i V_i = V(H)$; and split the set of hyperedges $E(H)$ into k disjoint sets $E_0, ..., E_{k-1}$, with $\bigcup_i E_i = E(H)$; satisfying:*

- *For all $i \in [0, k-1]$, the hypergraph $H_i = (V_i \cup (X \setminus \{x_i\}, E_i)$ is an l-connector gadget of $X \setminus \{x_i\}$.*
- *For all i, all hyperedges $e \in E_i$ are completely contained in $V_i \cup (X \setminus \{x_i\})$. Equivalently, no hyperedge $e \in E(H)$ contains two vertices in different subsets V_i, V_j, or connects V_i and the vertex x_i.*

We can show that Definition 1.3 and Definition 6.2 are equivalent.

Lemma 6.3.

$$\mathcal{H}_l = \bigcup_{k \geq 3} \mathcal{H}_l^k$$

Proof. Let H be a pattern in $\mathcal{H}_l$ with $|C| = k$, we show that it is also in $\mathcal{H}_l^k$. Let $C, \{D_1, D_2...\}, \{E'_1, E'_2, ...\}$ be the partitions of H according to Definition 1.3. We can set $X = C$, and for all $i \in [k]$ we set $V_i = D_{i+1}$, $E_i = E'_{i+1}$. For every $i > k$ we find some x_j not in E_i (one must exist by definition) and add D_i to V_j and E'_i to E_i. We can show that the result is a valid partition of H according to Definition 6.2:

- $H_i = (V_i \cup (X \setminus \{x_i\}, E_i)$ is an l-connector gadget of $X \setminus \{x_i\}$:
  - Condition 1 is satisfied as all the connected components in E_i touch some vertex in $X \setminus \{x_i\}$, and the set E'_{i+1} connects all such vertices.
  - Condition 2 is satisfied, as the induced l-trimmed subhypergraph of the core is empty (or only contains singleton hyperedges).
  - Condition 3 is satisfied, again by E'_{i+1}.
- Every hyperedge in E'_{i+1} was contained in $D_{i+1} \cup (C \setminus \{c_{i+1}\})$, and therefore every hyperedge E_i will be contained in $V_i \cup (X \setminus \{x_i\})$.

Now, let H bet a pattern in $\mathcal{H}_l^k$, we show that it is also in $\mathcal{H}_l$, let X,$\{V_0, ..., V_{k-1}\}$,$\{E'_0, ..., E'_{k-1}\}$ be the sets forming a valid partition of H according to Definition 6.2. Set $C = X$, let H' be the induced trimmed subhypergraph of $H \setminus C$, we will have connected components D_i, note that a D_i cannot have vertices originally in different sets V_j, similarly the sets containing E_i cannot be part of different sets E'_j. We prove that this will be a valid partition according to Definition 1.3:

1. Every hyperedge is part of some l-connector gadget, and therefore, every hyperedge with 2 or more vertices in the core C must have at least $l+1$ vertices outside of it, hence, it will not appear in the l-trimmed subhypergraph induced by the core.

2. This is true from the second condition of Definition 6.2.

3. We can reorder the sets E_i to satisfy this. Note that every set $C \setminus \{c_i\}$ forms an l-connector gadget, which requires some connected component (which will be a set D_i) to neighbor all the vertices in $C \setminus \{c_i\}$, which implies that E_i contains $C \setminus \{c_i\}$.

□

A hypergraph H having $\tau_l(H) > 1$ is equivalent to the reachability hypergraph of the l-skeletons of one of its acyclic orientations $\vec{H}$ not being α-acyclic. In order to capture the α-acyclicity of the reachability graph, we introduce the following definitions.

Definition 6.4 (reachable k-cycle)**.** *Let $\vec{H}$ be a directed acyclic graph and $k \geq 3$, a reachable k-cycle is a pair X, Y, where $Y = \{y_0, ..., y_{k-1}\} \subset V(\vec{H})$ and $X = \{x_0, ..., x_{k-1}\} \subset V(\vec{H})$, such that for every $i \in [0, k-1]$, $\{x_i, x_{i+1}\} \subseteq Reach(y_i)$ (taken modulo k in the indexes) and every vertex v with $|Reach v \cap X| \geq 2$ must have $Reach v \cap X = \{x_i, x_{i+1}\}$ for some $i \in [0, k-1]$.*

Definition 6.5 (reachable k-simplex)**.** *Let $\vec{H}$ be a directed acyclic graph and $k \geq 3$, a reachable k-simplex is a pair X, Y, where $Y = \{y_0, ..., y_{k-1}\} \subset V(\vec{H})$ and $X = \{x_0, ..., x_{k-1}\} \subset V(\vec{H})$, such that for every $i \in [0, k-1]$, $\{X \setminus \{x_i\}\} \subseteq Reach(y_i)$ and no vertex $v \in V(\vec{H})$ exists with $X \subseteq Reach(v)$.*

Note that a reachable 3-cycle and a reachable 3-simplex are equivalent.

We can relate the previous definitions to the concept of α-acyclicity. The following theorem proves that having a reachability hypergraph that is not α-acyclic is equivalent to containing either a reachable k-cycle or a reachable k-simplex.

Theorem 6.6. *(Theorem 8 in [BGL+22], originally in [BFM+81, BFMY83]) A hypergraph F is α-acyclic if and only if for every $k \geq 3$, there is no $S = \{x_0, x_1, ..., x_{k-1}\} \subseteq V(F)$ such that one of the following conditions holds:*

1. *For every $0 \leq i \leq k-1$ there exists a hyperedge $e \in E(F)$ such that $e \cap S = \{x_i, x_{i+1}\}$, and there is no $e \in E(F)$ with $|e \cap S| \geq 2$ such that $e \cap S \neq \{x_i, x_{i+1}\}$ for all $- \leq i \leq k-1$. (Indices taken modulo k.)*

2. *For every $0 \leq i \leq k-1$ there exists $e \in E(F)$ such that $e \cap S = S \setminus \{x_i\}$, and there is no edge $e \in E(F)$ such that $S \subseteq e$.*

We now relate containing a reachable k-cycle in the l-skeleton of an orientation $\vec{H}$ with H having a hypergraph in $\mathcal{H}_l^3$ as an induced trimmed subhypergraph. First, we introduce the following definition.

Definition 6.7 (First reachable cycle)**.** *Let $\pi : V(\vec{H}) \to \mathbb{N}$ be an ordering of the vertices of $\vec{H}$, and for a set of vertices V, let $\pi(V)_{(i)}$ denote the ordering of the i-th latest vertex in V with respect to V. A reachable l-cycle X, Y **precedes** a reachable l'-cycle X', Y' in the ordering π if either $X \subset X'$ or there exists an $j \leq \min(|X|, |X'|)$ such that:*

- *For all $i < j$ the vertices with the i-th latest ordering in X and X' are the same, $\forall i < j, \pi(X)_{(i)} = \pi(X')_{(i)}$.*
- *The vertex with the j-th latest ordering in X precedes the vertex with the j-th latest ordering in X', $\pi(X)_{(j)} < \pi(X')_{(j)}$.*

We call X, Y the First reachable cycle of $\vec{H}$ if it precedes all the other reachable cycles in $\vec{H}$.

Lemma 6.8. *Let $\vec{H}$ be a directed acyclic orientation of the hypergraph H. If $Sk_l(\vec{H})$ contains a reachable k-cycle then H contains a hypergraph $H' \in \mathcal{H}^3_l$ as an induced trimmed subhypergraph.*

Proof. Let $\pi : V(H) \to \mathbb{N}$ be an ordering of the vertices of H consistent with $\vec{H}$. And let $X, Y \subseteq V(H)$ be the first reachable cycle in $Sk_l(\vec{H})$.

Let $k = |X| = |Y|$, for every $i \in [0, k-1]$, let V_i be the set that contains, for every $v \in V(H)$ such that $\{x_i, x_{i+1}\} \subseteq Reach_{Sk_l(\vec{H})}(v)$, v and all the vertices in the shortest path from v to x_i and x_{i+1} in $Sk_l(\vec{H})$ (not including x_i and x_{i+1} themselves). Let H_i be the trimmed subhypergraph of H induced by $V_i \cup \{x_i, x_{i+1}\}$ and $E_i = E(H_i)$.

We show that H_i must be an l-connector gadget of $\{x_i, x_{i+1}\}$. First, note that V_i is not empty as $\{x_i, x_{i+1}\} \subseteq Reach_{Sk_l(\vec{H})}(y_i)$, and therefore $y_i \in V_i$.

- We can see that H_i is connected, every vertex is part of some path in $Sk_l(\vec{H})$, every pair connects a vertex v to either x_i or x_{i+1}, and every vertex v has two such paths; therefore, all the paths will form a single connected component. Every direct edge u, v in $Sk_l(\vec{H})$ corresponds to some hyperedge e in H that will appear partially in H_i connecting u and v.
- Let e be a hyperedge in H_i containing both $\{x_i, x_{i+1}\}$, $Sk_l(\vec{H})$ does not contain a direct edge between x_i and x_{i+1} (or vice-versa) which means that there are at least l vertices in the internal ordering of e before both x_i and x_{i+1}, note that the first $l+1$ vertices must connect to both x_i and x_{i+1} in $Sk_l(\vec{H})$ and therefore they will be in V_i, thus $|e \cap V_i| > l$.
- Finally, let H' be the trimmed subhypergraph of H induced by V_i, note that every connected component in H' will have to connect to both x_i and x_{i+1} as it necessarily contains a vertex v with its direct paths to x_i and x_{i+1} in $Sk_l(\vec{H})$.

We can also show the following claim.

Claim 6.9. *For every $i \neq j$, no vertex $v_i \in V_i$ shares an edge with a vertex $v_j \in V_j$.*

Proof. Assume otherwise that there is a vertex $v_i \in V_i$ and a vertex $v_j \in V_j$ such that there exists a hyperedge $e \in E(H)$ with $v_i, v_j \in e$. Let v'_i be a predecessor of v_i (or v_i itself) that reaches both x_i and x_{i+1} and similarly let v'_j be a predecessor of v_j (or v'_j itself) that reaches both x_j and x_{j+1}. Having a hyperedge connecting v_i and v_j in H implies one of the following three scenarios in $Sk_l(\vec{H})$. We show that they all lead to some contradiction:

- There is a direct edge connecting v_i to v_j. If v_j reaches a vertex in X other than x_i, x_{i+1} then $|Reach(v'_i) \cap X| \geq 3$, which is not possible. Thus, we have that v_j reaches either x_i or x_{i+1} (but not both). Without loss of generality, let $x_{i+1} \in Reach(v_j)$ (which means $j = i+1$), set $X' = v_j \cup X \setminus \{x_{i+1}\}$ and $Y' = (Y \setminus \{y_i, y_{i+1}\}) \cup \{v'_i, v'_j\}$.

Note that $\pi(v_j) < \pi(v_{i+1})$ and therefore X' precedes X. Moreover, we can verify that X', Y' forms a reachable k-cycle as no vertex can reach 3 or more vertices in X', since it will mean that it can reach the same number of vertices in X, leading to a contradiction.

- There is a direct edge connecting v_j to v_i. This is symmetrical to the previous case.
- There is a vertex v with direct edges connecting it to v_i and v_j. We distinguish the following four sub-cases:
  - $|X \cap \mathit{Reach}(v_i)| = 2$ and $|X \cap \mathit{Reach}(v_j)| = 2$. This implies $\{x_j, x_{j+1}, x_i, x_{i+1}\} \subseteq \mathit{Reach}(v)$ and therefore v reaches at least 3 vertices in X, which is not possible.
  - $|X \cap \mathit{Reach}(v_i)| = 2$ and $|X \cap \mathit{Reach}(v_j)| = 1$. In this case v_j must reach x_i or x_{i+1}, otherwise v reaches three vertices in X. Without loss of generality, let $x_{i+1} \in \mathit{Reach}(v_j)$ (and $j = i+1$). Set $X' = \{v_j\} \cup X \setminus \{x_{i+1}\}$ and $Y' = (Y \setminus \{y_i, y_{i+1}) \cup \{v, v'_j\}$. $\pi(v_j) < \pi(x_{i+1})$, which means that X' precedes X, and X', Y' will form a valid reachable k-cycle; therefore, X, Y is not the first reachable cycle, leading to a contradiction.
  - $|X \cap \mathit{Reach}(v_i)| = 1$ and $|X \cap \mathit{Reach}(v_j)| = 2$. Symmetrical to the previous case.
  - $|X \cap \mathit{Reach}(v_i)| = 1$ and $|X \cap \mathit{Reach}(v_j)| = 1$. If $X \cap \mathit{Reach}(v_i) \neq X \cap \mathit{Reach}(v_j)$, then for some i', $x_{i'} \in \mathit{Reach}(v_i)$ and $x_{i'+1} \in \mathit{Reach}(v_j)$ (or vice-versa, we assume the former without loss of generalization). There are two further possibilities, either $j = i' = i+1$ or $j = i'+1 = i+2$. In the former, we can set $X' = \{v_i\} \cup X \setminus \{x_{i+1}\}$ and $Y' = (Y \setminus \{y_i, y_{i+1}\}) \cup \{v'_i, v\}$ to obtain a valid reachable cycle where X' precedes X as $\pi(v_i) < \pi(x_{i+1})$. In the latter, we can set $X' = \{v_i, v_j\} \cup X \setminus \{x_{i+1}, x_{i+2}\}$ and $Y' = (Y \setminus \{y_i, y_{i+1}, y_{i+2}\}) \cup \{v'_i, v, v'_j\}$ to obtain a valid reachable cycle, again X' precedes X as both $\pi(v_i) < \pi(x_{i+1})$ and $\pi(v_j) < \pi(x_{i+2})$.
    Alternatively, $X \cap \mathit{Reach}(v_i) = X \cap \mathit{Reach}(v_j)$, and they both reach x_i or x_{i+1}, without loss of generality, assume $X \cap \mathit{Reach}(v_i) \cap \mathit{Reach}(v_j) = \{x_{i+1}\}$ (and $j = i+1$). Set $X' = \{v_i, v_j\} \cup X \setminus \{x_{i+1}\}$ and $Y' = (Y \setminus \{y_i, y_{i+1}\}) \cup \{v'_i, v, v'_j\}$. Both $\pi(v_i) < \pi(x_{i+1})$ and $\pi(v_j) < \pi(x_{i+1})$, which means that X' precedes X. The only possibility that X', Y' does not form a reachable cycle is that there is a vertex u that reaches both v_i, v_j and either x_i or x_{i+2}, but then we can construct the sets $X'' = X' \setminus \{v_i\}$ and $Y'' = u \cup Y' \setminus \{v, v'_i\}$ in the former or $X'' = X' \setminus \{v_j\}$, $Y'' = u \cup Y' \setminus \{v, v'_j\}$ in the latter which will be a valid reachable cycle. Moreover, $X'' \subset X'$, therefore, X'' precedes X' and X, leading to a contradiction.

□

We can show that H', the trimmed subhypergraph of H induced by $X \cup \bigcup_{i=0}^{k-1} V_i$ is in $\mathcal{H}_l^3$. If $k = 3$, then we are done as every hypergraph H_i is an l-connector gadget and by Claim 6.9 they do not share any hyperedge. Otherwise, let $V'_2 = \bigcup_{i=2}^{k-1} V_i \cup \bigcup_{i=3}^{k-1} x_i$, and H'_2 be the trimmed subhypergraph of H induced by $V'_2 \cup \{x_2, x_0\}$. We can verify that this split satisfies all the conditions of Definition 6.2:

- H_0 is an l-connector gadget for $V \setminus \{x_2\} = \{x_0, x_1\}$.
- H_1 is an l-connector gadget for $V \setminus \{x_0\} = \{x_1, x_2\}$.
- H'_2 is an l-connector gadget for $V \setminus \{x_1\} = \{x_2, x_0\}$.We can verify the following conditions:

- H_2' is connected, as every V_i connects with the vertices x_i,x_{i+1} which will be included in H_2'. It also forms a connected component if only including the vertices in V_2'.
- No hyperedge in H_2' includes both $\{x_2, x_0\}$ as it will mean the existence of a vertex reaching both of them in $Sk_l(\vec{H})$, which is not possible for $k > 3$.

- No hyperedge contains two vertices in different subsets, again this is true from Claim 6.9.

Moreover, $V_0 \cup V_1 \cup V_2' \cup \{x_0, x_1, x_2\} = V(H')$ and $E_0 \cup E_1 \cup E(H_2') = E(H')$. Therefore, we will have that $H' \in \mathcal{H}_l^3$. □

We can prove a similar result for reachable k-simplex. We show that containing a reachable k-simplex in $Sk_l(\vec{H})$ implies containing a pattern in $\mathcal{H}_l^k$ as an induced trimmed subhypergraph. We will first prove the following auxiliary lemma.

Lemma 6.10. *Let $\vec{H}$ be a directed acyclic graph with ordering π that does not contain a reachable k-cycle for any value of k but contains some reachable k-simplex for some k. Let k' be the minimum k such that $\vec{H}$ contains a reachable k-simplex and X, Y be the sets that form a reachable k'-simplex with minimum total ordering $\sum_{v \in X} o(v)$. Let $V \subseteq V(\vec{H} \setminus X)$ be a connected subset of vertices such that $\forall v \in V$, $|Reach(v) \cap X| \geq 1$. We will have $X \not\subseteq Reach_{\vec{H}}(V)$.*

Proof. We assume $X \subseteq Reach_{\vec{H}}(V)$ and prove by contradiction. Let $V' = \{v \in V(\vec{H}) \setminus X || Reach(v) \cap X| \geq 1\}$ be the set of vertices of $V(\vec{H}) \setminus X$ that reach some vertex in X. Note that $V' \subseteq V''$. Let $\vec{H''}$ be the subgraph of $\vec{H}$ induced by V'', this graph might be disconnected, but at least one of its connected components must contain V'. Let V^* be the set of vertices inducing that connected component. Clearly $V' \subseteq V^*$. We prove the following claim.

Claim 6.11. *Let v_1, v_2 be two vertices in V^*, there must exist a vertex $v_3 \in V^*$ such that $(Reach(v_1) \cup Reach(v_2)) \cap X \subseteq Reach(v_3) \cap X$.*

Proof. If $Reach(v_1) \subseteq Reach(v_2)$, then $v_3 = v_2$ and we are done. Similarly if $Reach(v_2) \subseteq Reach(v_1)$. Otherwise, there is no direct path from v_1 to v_2. However, v_1 and v_2 are part of the same connected component in $\vec{H''}$ and there must be an undirected path connecting them.

An l-alternating path is a sequence $a_0, b_1, ..., b_l, a_l$ such that $b_i \in Reach(a_{i-1}) \cap Reach(a_i)$ for all $i \in [1, l]$. An l-alternating path is minimal if there is no l'-alternating path connecting a_0 and a_l for some $l' < l$. Let l' be the minimum l such that there is an l-alternating path with $a_0 = v_1$ and $a_l = v_2$ using only vertices in V^*. It suffices to show that for any l-alternating path, there exists a vertex v with $((Reach(a_0) \cup Reach(a_l)) \cap X) \cup \{b_i\}$ for some $1 \leq i \leq l$.

We use induction on the size l of the alternating path to show that such a vertex must exist.

Base case: First, we show for $l = 1$. We have a vertex b such that $b \in Reach(a_0) \cap Reach(a_1)$. Let $\mathcal{Z}$ be the collection of possible subsets of $((Reach(a_0) \cup Reach(a_1)) \setminus Reach(b)) \cap X$ such that there exists a vertex v that reaches both b and the vertices of the subset. That is, $\mathcal{Z} = \{Z \subseteq ((Reach(a_0) \cup Reach(a_1)) \setminus Reach(b)) \cap X | \exists v \in V^*, Z \cup b \subseteq Reach(v)\}$.

If $\mathcal{Z}$ contains $((Reach(a_0) \cup Reach(a_1)) \setminus Reach(b)) \cap X$ then there is a vertex $v \in V^*$ that reaches b and all the vertices in $((Reach(a_0) \cup Reach(a_1)) \setminus Reach(b)) \cap X$, but because v reaches b it will also reach all the vertices in $Reach(b)$, hence $Reach(v) \supseteq (Reach(a_0) \cup Reach(a_1) \cap X$.

Else, let Z' be the smallest subset of $((Reach(a_0) \cup Reach(a_1)) \setminus Reach(b)) \cap X$ such that $Z' \notin \mathcal{Z}$. Note that $|Z'| < k'$, as $|Reach(b) \cap X| \geq 1$, also $|Z'| \geq 2$, as every vertex in $((Reach(a_1) \cup Reach(a_2)) \setminus Reach(b)) \cap X$ is reached by either a_0 or a_1. We distinguish two cases:

- $|Z'| = 2$: Without loss of generality, let $x_0, x_1 \in X$ be the two vertices in Z' and let $x_0 \in Reach(a_0)$ and $x_1 \in Reach(a_1)$. Recall that the vertices of X are part of a reachable k'-cycle, therefore, there is a vertex $y \in Y$ that reaches x_0, x_1. Note that the sets $Y' = \{a_0, a_1, y\}$ and $X' = \{x_0, x_1, b\}$ will form a reachable 3-cycle, as $\{x_0, x_1\} \in Reach(y)$, $\{x_0, b\} \in Reach(a_0)$, $\{x_1, b\} \in Reach(a_1)$, and no vertex reaches all vertices in X'. But this violates the assumption of the lemma that $\vec{H}$ does not contain any reachable k-cycle, leading to a contradiction.

- $|Z'| \geq 3$: Let $\mathcal{Z}' = \{Z \subset Z' || Z| = |Z'| - 1\}$ and note that every set in $\mathcal{Z}'$ must also be in $\mathcal{Z}$, therefore for every set $Z \in \mathcal{Z}'$ there is a vertex v_Z such that $b \cup Z \in Reach v_z$; additionally, because $|Z'| < k'$ there is a vertex $y \in Y$ with $Z \in Reach(y)$. We set $Y' = \{y\} \cup \{v_Z | Z \in \mathcal{Z}'\}$ and $X' = Z' \cup \{b\}$, let $k'' = |X'| = |Y'| \leq k'$.

  We can see that Y', X' forms a reachable k''-simplex as no vertex reaches the entire set X' (otherwise Z' would be in $\mathcal{Z}$). If $k'' < k'$, then k' is not the minimum k such that $\vec{H}$ contains a reachable k-simplex, which leads to a contradiction. Otherwise, if $k'' = k'$ we can see that the only difference between X' and X is that X' contains the vertex b and X the vertex $c = X \setminus Z'$, but in this case $c \in Reach(b)$ which implies that b is strictly before c in any ordering of the vertices of $\vec{H}$, that is, $\pi(b) < \pi(c)$ and therefore $\sum_{y \in Y'} \pi(y) < \sum_{y \in Y} o(y)$ which would again contradict the assumption from the lemma.

Inductive step: Assume that for any $l < l'$ we have that for all l shortest alternating path with endpoints $a', b' \in V^*$ there is a vertex v with $Reach(a') \cup Reach(b') \cup b_i \subseteq Reach(v)$ for some b_i in the alternating path. We show that the same holds for $l = l'$.

Let a_0, b_1 be the endpoints of a l' shortest alternating path $a_0, b_1, ..., b_{l'}, a_{l'}$ in V^*. Let $\mathcal{Z}$ be the collection of possible subsets of $(Reach(a_0) \cup Reach(a_{l'})) \cap X$ such that there exists a vertex v that reaches the vertices of the subset and some vertex b_i in the alternating path. That is, $\mathcal{Z} = \{Z \subseteq (Reach(a_0) \cup Reach(a_{l'})) \cap X | \exists v \in V^* \exists i \in [1, l'], Z \cup b_i \subseteq Reach(v)\}$.

If $(Reach(a_0) \cup Reach(a_{l'})) \cap X \in \mathcal{Z}$, then we are done, as there is a vertex that reaches $(Reach(a_0) \cup Reach(a_{l'})) \cap X$ and a vertex b_i in the alternating path. Otherwise, let $Z' \subseteq (Reach(a_0) \cup Reach(a_{l'})$ be the minimum set such that $Z' \notin \mathcal{Z}$. Note that $|Z'|$, as any individual vertex in $(Reach(a_0) \cup Reach(a_{l'})$ will be reachable by either a_0 or $a_{l'}$. We distinguish two cases:

- $|Z'| = 2$. Let $Z' = \{u_0, u_{l'}\} \subset X$. Note that one of them must be reached by a_0 and the other by $a_{l'}$, otherwise we would have $Z' \in \mathcal{Z}$. Without loss of generality, let $u_0 \in Reach(a_0)$ and $u_{l'} \in Reach(a_{l'})$.

  Note that no vertex b_i with $1 \leq i < l'$ can be reached by a vertex v that reaches $u_{l'}$, as in that case we would have that v and a_0 form an l-connecting path with $l < l'$, therefore we could use the inductive hypothesis to show that there is a vertex v' with $(Reach(v_0) \cup Reach(v) \cup \{b_{i'}\}) \cap X \subseteq Reach(v')$, for some $b_{i'}$ in the alternating path, but this would mean that $Z \in \mathcal{Z}$, leading to a contradiction. The same argument can be made to show that no vertex b_i with $1 < i \leq l'$ can be reached by a vertex v that reaches u_0.

  Note that because $u_0, u_{l'} \in X$ there will be a vertex $y \in Y$ that reaches both u_0 and $u_{l'}$. Let $Y' = \{y, a_0, ..., a_{l'}\}$ and $X' = \{u_0, u_{l'}, b_1, ..., b_{l'}\}$. We can show that X', Y' form a reachable $(l' + 2)$-cycle.

  - No vertex can reach all $u_0, u_{l'}$, and a vertex in b_i, since then we would have $Z' \in \mathcal{Z}$.

– No vertex can reach two vertices in $b_1, ..., b'_l$ and u_0, since then a vertex would reach u_0 and some b_i with $1 < i \leq l'$.

– Similarly, no vertex can reach two vertices in $b_1, ..., b'_l$ and $u_{l'}$, as then a vertex would reach $u_{l'}$ and some b_i with $1 \leq i < l'$. Finally, No vertex can reach three vertices in $b_1, ..., b'_l$, as then there would be a shorter alternating path between a_0 and $a_{l'}$.

Therefore, X', Y' is a reachable $l'+2$-cycle, but $\vec{H}$ does not contain any reachable $l'+2$-cycle, leading to a contradiction.

- $|Z'| > 2$. Let $\{u_0, u_1, u_2\} \subseteq Z'$. Note that all $Z' \setminus \{u_0\}, Z' \setminus \{u_1\}, Z' \setminus \{u_2\} \in \mathcal{Z}$, which means that there are three vertices v'_0, v'_1, v'_2 such that v'_i reaches $Z' \setminus \{u_i\}$ and some b_j in the alternating path, for $i \in \{0, 1, 2\}$. Consider the shortest alternating path between each pair of v'_0, v'_1, v'_2, note that at least one of the pairs will form an alternating path with length $l < l'$. Without loss of generality, let v'_0, v'_1 be such a pair. Using the induction hypothesis we have that there is a vertex v'' such that $(Reach(v'_0) \cup Reach(v'_1) \cup \{b_j\}) \cap X \subseteq Reach(v'')$ for some b_j in the alternating path. But note $Z' \subseteq (Reach(v'_0) \cup Reach(v'_1)) \cap X$, and thus $Z' \in \mathcal{Z}$, which leads to a contradiction.

□

We can now use the claim to complete the proof of the lemma. As we said, assume that we have a connected subset of vertices V such that $X \subseteq_{u \in V} Reach(v)$. Let S be the set of vertices of V that cannot be reached by any other vertex in V, let $l = |S|$ and order the vertices $s_1, ..., s_l$ of S arbitrarily. We use induction on l to show the existence of a vertex v in V^* with $X \cap \bigcup_{i \leq l} Reach(s_i) \subseteq Reach(v)$.

The base case is trivial as s_1 will satisfy the condition for $l = 1$. Now we assume that there exists a vertex v in V^* with $X \cap \bigcup_{i<l} Reach(s_i) \subseteq Reach(v)$ and prove that there is a vertex v' with $X \cap \bigcup_{i \leq l} Reach(s_i) \subseteq Reach(v)$. Using Claim 6.11 on v and s_l we have that there is vertex v' with $(Reach(v) \cup Reach(s_l)) \cap X \subseteq Reach(v')$, which means $X \cap \bigcup_{i \leq l} Reach(s_i) \subseteq Reach(v')$, completing the induction.

This means that there is a vertex v that reaches all the vertices reachable by the vertices in V, and therefore reaches all the vertices in X, but then Y, X would not be a valid reachable k'-cycle, which leads to a contradiction, completing the proof of the lemma.

□

Lemma 6.12. *Let $\vec{H}$ be a directed acyclic orientation of the hypergraph H. If $Sk_l(\vec{H})$ does not contain a reachable k-cycle for any k, and contains a reachable k'-simplex for some $k' \geq 4$, then H contains a hypergraph $H' \in \mathcal{H}^k_l$ for some $k \leq k'$, as an induced trimmed subhypergraph.*

Proof. Let $\pi : V(H) \to \mathbb{N}$ be an ordering of the vertices of H agreeing with $\vec{H}$. Let k be the minimum k such that $Sk_l(\vec{H})$ contains a reachable k-simplex, we have $4 \leq k \leq k'$. And let X, Y be the reachable k-simplex in $Sk_l(\vec{H})$ with minimum $\sum_{i=0}^{k-1} o(x_i)$.

Let V' be the set of vertices that reach some vertex in X, $V' = \{v \in Sk_l(\vec{H}) | \exists x_i \in X, x_i \in Reach_{Sk_l(\vec{H})}(v)\}$. Let H'' be the trimmed subhypergraph of H induced by V'

Let p be the number of connected components in H'. We can split the vertices of V' into p disjoint sets $V'_0, ..., V'_p$ for each of the connected components of H''. For every p, let X_p be the subset of vertices of X reachable by some vertex in V'_p. Note that from Lemma 6.10, X_p must be a strict

subset of X; therefore, for every p there exists a vertex $x_i \in X$ such that $X_p \subseteq X \setminus \{x_i\}$, note that there might be more than one such vertex. For every p assign a vertex x_i such that $X_p \subseteq X \setminus \{x_i\}$, let x_p denote such vertex. We can now group the vertices of V' into k sets $V_i = \bigcup_{p|x_p=x_i} V'_p$.

Now, let H' be the trimmed subhypergraph of H induced by $V' \cup X$, and $E' = E(H')$. We will show that $H' \in \mathcal{H}_l^3$. We can group the hyperedges E' into k sets. Let $s(e)$ be the earliest vertex in e according to π. We will have $E_i = \{e \in E' | s(e) \in V_i\}$. Let $H_i = (V_i, E_i)$.

Note that $\bigcup_{i<k} V_i = V'$, and therefore $X \cup \bigcup_{i<k} V_i = V(H')$. Similarly $\bigcup_{i<k} E_i = E' = E(H')$. We can verify that H_i will be an l-connector gadget for $X \setminus \{x_i\}$. H_i is the union of some of the connected components in H'' and the vertices in $X \setminus \{x_i\}$. The vertex $y_i \in Y$ reaches all the vertices in $X \setminus \{x_i\}$, therefore, at least one of such connected components must connect to all the vertices in $X \setminus \{x_i\}$, this satisfies condition 3 of the definition of l-connector gadget. All the other connected components will connect to at least one of those vertices, which means H_i is connected, satisfying condition 1. Finally, let e be an edge in E_i that contains at least 2 vertices in $X \setminus \{x_i\}$, note that none of these vertices might be the first $l+1$ in the internal ordering of the edge in $\vec{H}$, as otherwise there would be an edge connecting them in $Sk_l(\vec{H})$. Note that those $l+1$ vertices must also be included in V_i since they reach some vertices in X and are part of the same connected component as its source, satisfying the second condition.

Finally, we have that no hyperedge contains vertices in different subsets, as otherwise they would be in the same connected component, which means that they could not be in different subsets. Therefore, we get that $H' \in \mathcal{H}_l^k$. □

We can finally prove Theorem 2.4, which we restate.

Theorem 2.4. *Let $l \in \mathbb{Z}^+ \cup \{\infty\}$ and H a hypergraph. $\tau_l(H) = 1$ if and only if H is $\mathcal{H}_l$ ITS free.*

Proof. Note that H having $\tau_l(H) > 1$ is equivalent to having an orientation $\vec{H}$ such that its l-skeleton $Sk_l(\vec{H})$ has $\tau(\vec{H}) = 1$, which from Lemma 3.8 is equivalent to the reachability hypergraph of $Sk_l(\vec{H})$ being α-acyclic. Hence, it suffices to show that such an orientation exists if and only if H contains a graph in $\mathcal{H}_l$ as an induced trimmed subhypergraph.

- We first prove the "if" part, let $H' \in \mathcal{H}_l^k$ be an induced trimmed subhypergraph of H. Let X, $V_0, ..., V_{k-1}$ be the sets of vertices of $V(H')$ and $E_0, ..., E_{k-1}$ the sets of hyperedges that satisfy the conditions of Definition 6.2. We create an ordering π of the vertices of H in which first we will have all the vertices in the V_i sets, then the vertices in X, and finally the vertices in $V(H) \setminus V(H')$.

  For every set V_i, select an arbitrary vertex v_i in one of the connected components that connects to all the vertices in $V \setminus \{v_i\}$, and for every $j \neq i$ set the vertices in the shortest path from v_i to x_j in ascending order, allowing v_i to reach x_j in any l-skeleton. We argue that the sets $Y = \{v_0, ..., v_{k-1}\}$ and X form a reachable k-simplex in $Sk_l(\vec{H})$.

  From the way we ordered the vertices we have that $X \setminus \{x_i\} \subseteq Reach(v_i)$ for all i. Only rest to show is that no vertex can reach all the vertices in X. Assume there is a vertex v such that $X \subseteq Reach(v)$, note that $v \notin V(H) \setminus V(H')$, as the vertices in $V(H) \setminus V(H')$ are all after the vertices in X. If $v \in X$ it means that there is a direct edge connecting a pair of vertices in X. Otherwise, if $v \in V_i$ for some i, then the path to x_i must necessarily include another vertex in X, also meaning that there must be a direct edge connecting a pair of vertices in X.

We can prove that no direct edge can connect two vertices in X. Let e be any hyperedge in H containing two vertices $x_j, x_{j'} \in X$, and $e' = e \cap V(H')$, we know $|e'| > 2$ and there must be an $i \neq j, j'$ such that $e' \in E_i$. But $H_i = (V_i \cup X \setminus \{x_i\}, E_i)$ is an l-connector gadget of the set $X \setminus \{x_i\}$ which means that e' must contain at least a vertex $l+1$ vertices in V_i, and therefore the l-skeleton of $\vec{H}$ will not include a direct edge between x_i and x_j.

- Now we prove the only if part, let $\vec{H}$ be an orientation of H such that the reachability graph of $Sk_l(\vec{H})$ is α-acyclic. We show that H contains a hypergraph in $\mathcal{H}_l$ as an induced trimmed subhypergraph. From Theorem 6.6 we have two cases, the first one is equivalent to $Sk_l(\vec{H})$ containing a reachable k-cycle for some $k \geq 3$ and the second is equivalent to $Sk_l(\vec{H})$ containing a reachable k-simplex for some $k \geq 3$. In the first case, from Lemma 6.8 we find that H contains a pattern in $\mathcal{H}_l^3$ as an induced trimmed subhypergraph. In the second case, using Lemma 6.12 we find that H contains a pattern in $\mathcal{H}_l^{k'}$ for some $k' \leq k$.

□

6.1 The special case of $\mathcal{H}_\infty$

As mentioned in the introduction, for the case of $l = \infty$, suffices to look at which patterns have an LICL greater or equal than 6.

Claim 1.8. *H is $\mathcal{H}_\infty$ ITS free iff $LICL(H^c) < 6$ where H^c is the clique completion of H.*

Proof. We prove that H will contain a pattern $H' \in \mathcal{H}_\infty$ as an induced trimmed subhypergraph if and only if $\text{LICL}(H^c) \geq 6$.

We first prove the if direction. Let H be a pattern with $\text{LICL}(H^c) \geq 6$. Let t be the LICL, and $V' = v_1, ..., v_t$ the vertices in the cycle in the order. Let H' be the trimmed subhypergraph induced by V', clearly it will only contain the edges of the cycle and H' is an induced cycle. We can set $C = c_1 = v_1, c_2 = v_3, c_3 = v_5$, $D_1 = \{v_4\}$,$D_2 = \{v_6, ..., v_t\}$ and $D_3 = \{v_2\}$ and see that $H' \in \mathcal{H}_\infty$.

We now prove the only if part. Let H be a pattern containing $H' \in \mathcal{H}_\infty$, H^c the clique completion of H and H'' the clique completion of H'. Consider the vertices c_1, c_2, c_3 in the core, and consider the sets D_1, D_2, D_3. Find the shortest paths between c_1 and c_2, c_2 and c_3, and c_3 and c_1, in D_1,D_2 and D_3. These paths must have length at least 2 and do not share edges with each other. Therefore, forming an induced cycle of length at least 6. With implies $\text{LICL}(H^c) \geq \text{LICL}(H'') \geq 6$. □

7 Hardness of counting hypergraph homomorphisms

We prove hardness using colorful subgraph counts. Given a graph, a coloring is a function $c : V(G) \to [h]$. A homomorphism ϕ is colorful if it maps every vertex to a vertex of a different color. This implies that the homomorphism must be injective, as it can not map to the same vertex twice. We use $\text{Col-Hom}_H(G)$ for the number of colorful homomorphisms from H to G. We can show that like in graphs ([BGL+22, Mee16]), one can reduce counting colorful hypergraph homomorphisms to uncolored homomorphisms.

Lemma 7.1. *Let $l \in \mathbb{Z}^+ \cup \{\infty\}$ and H be a hypergraph. If there is an $O(n^\gamma) f(\kappa_l(G))$ algorithm for computing $\text{Hom}_H(G)$ for all hypergraphs G, then there is an $O(n^\gamma) f(\kappa_l(G))$ algorithm for computing Col-$\text{Hom}_H(G)$ for any colored hypergraph G with $h = |V(H)|$ colors.*

Proof. Fix l. Let G be a colored hypergraph with colors $c : V(G) \to [h]$, let n be the number of vertices of G and $\kappa_l = \kappa_l(G)$ its l-degeneracy. Let $I \subset [h]$ be a subset of the colors; we can define V_I as the set of vertices of G with color in I, $V_I = \{v \in V(G) | c(v) \in I\}$.

Let $G[V_I]$ be the subhypergraph of G induced by the vertices of V_I. The number of homomorphisms ϕ from H to G such that $c(Img(\phi))$ is equal to $\text{Hom}_H(G[V_I])$. Note that for that to hold we need to use the induced subhypergraphs and not the induced *trimmed* subhypergraphs.

A colorful homomorphism ϕ will require that $c(Img(\phi)) = [h]$. We can use inclusion-exclusion to find the number of colorful homomorphisms:

$$\text{Col-Hom}_G(H) = \sum_{I \subseteq [h]} (-1)^{|I|} \text{Hom}_H(G[I])$$

Note that for all graphs $G[I]$, we have $V(G[I]) \leq n$ and $\kappa_l(G[I]) < \kappa_l$. Therefore, for each $I \subseteq [h]$ we can compute $\text{Hom}_H(G[I])$ in time $O(n)f(\kappa_l(G))$. Assuming $h = O(1)$, we can then use all the counts to compute $\text{Col-Hom}_G(H)$ in total time $O(n)f(\kappa_l(G))$. □

We also show that, for any k, detecting a k-simplex can be done in the same time as detecting a colorful k-simplex.

Lemma 7.2. *Let $k = O(1)$. If there is an algorithm for detecting a colorful k-simplex in a colored hypergraph with m hyperedges in time $f(m)$, then there is an algorithm for detecting a k-simplex in an uncolored hypergraph with m hyperedges in time $\tilde{O}(f(m))$.*

Proof. Given a graph G we can randomly color G into G' using the the $k+1$ colors of the colorful k-simplex uniformly at random. If G contained a k-simplex, we will have that G' contains a colorful k-simplex with constant probability $(1/k)$, we can then use the $f(m)$ algorithm. This process can be derandomized using color-coding [AYZ94], adding a logarithmic factor. □

The last piece of the reduction is the construction of the graph G_l^H. This reduction is based on the construction shown in [BPS21]. The idea is to take a colored regular arity hypergraph G and replace every hyperedge with part of the pattern hypergraph H. Obtaining a hypergraph such that a colorful copy H exists if and only if there is a colorful simplex in G.

For a hypergraph H containing $H' \in \mathcal{H}_l^k$ as an induced trimmed subhypergraph, we will use $V\{ext\}$ for the set of vertices $V(H) \setminus V(H')$, and $E\{ext\} = \{e \in E(H) : e \subseteq V^{ext}\}$ for the set of edges contained entirely in V^{ext}. We also use X to refer to the core vertices of H', and V_i for each of the partitions of vertices of H' that satisfy the properties of Definition 6.2. We set $E_i = \{e \in E(H) \setminus E^{ext} | e \subseteq X \cap V_i \cap V^{ext}\}$.

Definition 7.3 (G_l^H)**.** *Let H be a hypergraph pattern with h vertices containing $H' \in \mathcal{H}_l^k$ as an induced trimmed subhypergraph. Let G be an k-colored regular $(k-1)$-regular arity hypergraph. Let $c : V(G) \to [k]$ be the coloring function of G. And let $\pi : V(H) \to [h]$ be an ordering of the vertices of H, such that for all $x_i \in X$, $\pi(x_i) = i$. We construct the h-colored hypergraph G_l^H as follows.*

- *Initialize G_l^H as an empty hypergraph with coloring function c'.*
- *For all $v \in V(G)$, add v to $V(G_l^H)$ with $c'(v) = c(v)$.*
- *For all $v \in V^{ext}$, add v to $V(G_l^H)$ with $c'(v) = \pi(v)$. Add each hyperedge $e \in E^{ext}$ to $E(G_l^H)$ connecting the corresponding vertices.*

- *For each colorful hyperedge $e \in E(G)$, that is, a hyperedge with $k-1$ distinct colors in its vertices, let i be the color not in e, and $y_j \in e$ be the vertex in e with color $j \neq i$. Add a copy V_i^e of V_i to $V(G_l^H)$ with $c'(v) = \pi(v)$ for every $v \in V_i$. For every hyperedge $e' \in E_i$ we will create a hyperedge e'' in $E(G_l^H)$ such that:*
  - *$\forall x_j \in X \cap e'$, we add y_j to e''.*
  - *$\forall v \in V_i \cap e'$, we add $u \in V_i^e$ with $c(u) = o(v)$ to e''.*
  - *$\forall v \in V^{ext}$, we add $u \in V(G_l^H)$ with $c(u) = o(v)$ to e''.*

Lemma 7.4. *G_l^H can be constructed in $O(n+m)$ time, where m is the number of hyperedges of G. Moreover, G_l^H will have $\kappa_l(G_l^H) = O(1)$.*

Proof. First, we will create a vertex v for each vertex in $V(G)$, which will take $O(n)$ time. Then, for each hyperedge we need to add a subhypergraph of H to G_l^H, this will take $O(m)$ total time, as the number of vertices and hyperedges of H is constant with respect to the size of G. Therefore, it will take $O(n+m)$ total time.

Now, we show that G_l^H has constant l-degeneracy. We give an ordering $\pi : V(G_l^H) \to [|V(G_l^H)|]$ of the vertices such that for the directed hypergraph $\vec{G}_l^H$ obtained from applying π to G_l^H we have $\Delta_l^+(\vec{G}_l^H) = O(1)$.

We divide the vertices of $V(G_l^H)$ in three portions: $V^{(3)}$ for the vertices corresponding to V^{ext} in H, $V^{(2)}$ for the vertices corresponding to $V(G)$ and $V^{(1)}$ for the remaining vertices corresponding with vertices in the portions V_i of H. Let π be any ordering such that all vertices in $V^{(1)}$ precede all vertices in $V^{(2)}$, and all vertices in $V^{(2)}$ precede all vertices in $V^{(3)}$. We can show that the l-outdegree of each vertex in the directed graph will be constant:

- If $u \in V^{(1)}$, then u can only be included in at most $|E(H)| = O(1)$ hyperedges in G_l^H. The arity of each hyperedge is at most $V(H) = O(1)$, and therefore u will only have a constant number of neighbors.
- If $u \in V^{(2)}$, note that every hyperedge that includes two vertices in $V^{(2)}$ must contain at least $l+1$ vertices in $V^{(1)}$, therefore, the l-skeleton of $\vec{G}_l^H$ can not include a direct edge between two vertices in $V^{(2)}$, thus, u can only have as l-out-neighbors vertices in $V^{(3)}$. Which implies $d^+(u) = O(1)$.
- If $u \in V^{(3)}$, then $d^+(u) < |V^{(3)}| = O(1)$.

□

Lemma 7.5. *G_l^H contains a colorful copy of H, if and only if G contains a colorful k-simplex.*

Proof. We first prove the if part. Let the vertices $v_0, ..., v_{k-1}$ of G induce a copy of a colorful k-simplex in G, we show how to find a colorful copy of H in G_l^H. Let $e_i = \{v_0, ..., v_{k-1}\} \setminus \{v_i\}$. Note that $e_i \in E(G)$, otherwise $v_0, ..., v_{k-1}$ do not induce a simplex. Consider the sets $V_i^{e_i}$ for all i, and V^{ext} in G_l^H. We show that $\{v_0, ..., v_{k-1}\} \cup \bigcup_i V_i^{e_i} \cup V^{ext}$ induce a colorful copy of H. First, note that every vertex has a different color, as every vertex in H corresponds to one vertex in the set. Moreover, every edge in H is either in E^{ext} or in one of the sets E_i, in both cases from the construction it will appear containing the corresponding vertices in $v_0, ..., v_{k-1} \cup \bigcup_i V_i^{e_i} \cup V^{ext}$.

We now prove the only if part, let the set V' induce a colorful copy of H in G_l^H, V' must contain vertices $y_0, ..., y_{k-1}$ with colors 0 to $k-1$ that are also present in $V(G)$. Moreover, for each subset $\{y_0, ..., y_{k-1}\} \setminus \{y_i\}$ we will have that G must contain an edge e_i containing them, otherwise the set E_i cannot be completely included in the subhypergraph induced by V'. Therefore, it must be the case that $\{y_0, ..., y_{k-1}\}$ induce a colorful k-simplex. □

Theorem 2.5. *Let $l \in \mathbb{Z}^+ \cup \infty$ and let H be a hypergraph that contains $H' \in \mathcal{H}_l$. If there exists an algorithm that computes $\mathrm{Hom}_H(G)$ in time $f(\kappa_l)O(n^\gamma)$ for all G and some $\gamma > 1$, then there is an algorithm that, for some $k \geq 2$, detects if a regular arity $k+1$ hypergraph G contains a k-simplex in time $\tilde{O}(m^\gamma)$.*

Proof. From Lemma 7.1 we will have that the $f(\kappa_l)O(n^\gamma)$ algorithm for computing $\mathrm{Hom}_H(G)$ implies an $f(\kappa_l)O(n^\gamma)$ algorithm for computing $\text{Col-Hom}_H(G)$.

We construct G_l^H from G as in Definition 7.3, from Lemma 7.4 we have that $\kappa_l(G_l^H) = O(1)$, moreover G_l^H will have $O(m)$ hyperedges. Therefore, we can count the number of colorful homomorphisms of H in G_l^H in time $O(m^\gamma)$.

From Lemma 7.5 we have that $\text{Col-Hom}_H(G_l^H) > 0$ if and only if G contains a colorful k-simplex, effectively giving an $O(m^\gamma)$ algorithm for colorful k-simplex detection, which itself implies an $\tilde{O}(m^\gamma)$ algorithm for k-simplex detection using Lemma 7.2. □

8 From homomorphisms to subhypergraphs

In this section, we show how to obtain a dichotomy for linear time subhypergraph counting from the hypergraph homomorphism counting dichotomy. Given by Theorem 1.7.

In a recent work, Bressan et al. showed that, similar to what happened in graphs, one can write the number of subhypergraph as a linear combination of hypergraph homomorphism [BBD+25]. Moreover, they generalized the result of [CDM17], showing that we can extend hardness from the homomorphisms into the subhypergraph (or any hypergraph motif parameter). We will show that, similar to the case of graphs [BGL+22], we can also use the hypergraph homomorphism basis to extend more fine-grained complexity lower bounds from homomorphisms into subhypergraphs.

Lemma 3.2 together with Theorem 1.6 is enough to prove the upper bound of Theorem 1.7. The lower bound requires a bit more of care and generalizes some of the results in [BGL+22]. We will prove the following lemma, which is analogous to Lemma 4.2 in [BGL+22].

Lemma 8.1. *Let $H_1, ..., H_k$ be pairwise non-isomorphic hypergraphs and let $c_1, ..., c_k$ be non-zero constants. For every hypergraph G there are hypergraphs $G_1, ..., G_k$ computable in time $O(|V(G)| + |E(G)|)$, such that for every $i \in [k]$, $|V(G_i)| = O(V(G))$, $|E(G_i)| = O(E(G))$ and $\kappa_l(G_i) = O(\kappa_l(G))$ for all $l \geq 0$; and such that knowing $b_j = \sum_{i \in [k]} c_i \mathrm{Hom}_{H_i}(G_j)$ for all $j \in [k]$, allows one to compute $\mathrm{Hom}_{H_i}(G)$ for all $i \in [k]$ in time $O(1)$.*

To prove this lemma we require of a tensor product for hypergraphs that preserve homomorphisms and does not increase the l-degeneracy. Fortunately, we can show that the tensor product introduced in [BBD+25] satisfies all these properties.

Given two hypergraphs H and G, one can define the G-projection and H-projections of their Cartesian products as the functions $\xi_G : V(G) \times V(H) \to V(G)$ and $\xi_G : V(G) \times V(H) \to V(H)$ that map each pair in $V(G) \times V(H)$ to its first or second component respectively. For a set $S \subseteq V(G) \times V(G)$ we can define:

$$\xi_G(S) = \{\xi_G(s) : s \in S\} \quad \xi_H(S) = \{\xi_H(s) : s \in S\}$$

We can now define the tensor product:

Definition 8.2 (Tensor product of hypergraphs [BBD+25])**.** *The tensor product of two hypergraphs $G \otimes H$ is the hypergraph with vertex set $V(G) \times V(H)$ and hyperedge set $\{e \subseteq V(H) \times V(H) : \xi_G(e) \in E(G)$ and $\xi_H(e) \in E(H)\}$.*

Bressan et al. also showed that the homomorphisms are preserved under the tensor product:

Lemma 8.3. *[BBD+25] All hypergraphs F, H, G satisfy:*

$$\mathrm{Hom}_F(G \otimes H) = \mathrm{Hom}_F(G) \cdot \mathrm{Hom}_F(H)$$

We can show that if H is constant sized, the product $G \otimes H$ will maintain the l-degeneracy of G.

Lemma 8.4. *Let G and H be two hypergraphs with $|V(H)| = O(1)$ and bounded rank. It holds:*

- $|V(G \otimes H)| = O(V(G))$.
- $|E(G \otimes H)| = O(E(G))$.
- *For all $l \geq 0$, $\kappa_l(G \otimes H) = O(\kappa_l(G))$.*

Proof. We proof each part separately:

- The vertex set of $G \otimes H$ is $V(G) \times V(H)$ which will have size $|V(G)| \cdot |V(H)| = O(|V(G)|)$.
- For every hyperedge $e \in G$ we can have multiple hyperedges $e' \in G \otimes H$ with $\xi_G(e') = e$, however, we can show that the set of such hyperedges is bounded. Let e' be one such hyperedge, it must contain, for each vertex $v \in e$ a subset of all the vertices $v \times V(H)$ in $G \otimes H$. The number of possible subsets is $2^{|V(H)|} = O(1)$ and we have that $|e| = O(1)$ as G has bounded rank. Therefore, at most a constant number of hyperedges can have $\xi_G(e') = e$. Which implies $|E(G \otimes H)| = O(E(G))$.
- Finally, fix some l, let $\pi : V(G) \to [V(G)]$ be an ordering of the vertices of G that agrees with the l-degeneracy orientation of G, $\vec{G}^{(l)}$. Construct an ordering $\pi' : V(G \otimes H) \to [G \otimes H]$ such that for every $u, v \in G \otimes H$, $\pi'(u) < \pi'(v)$ if $\pi(\xi_G(u)) < \pi(\xi_G(u))$. let $G \vec{\otimes} H$ be the result of orienting $G \otimes H$ according to π'. We show that $\Delta_l^+(G \vec{\otimes} H) = O(\kappa_l(G))$, which implies $\kappa_l(G \otimes H) = O(\kappa_l(G))$.

  Let $u \in V(G \otimes H)$ be a vertex with $\xi_G(u) = v$, let d be the l-outdegree of v in $\vec{G}$, from Lemma 2.2 we have $d = O(\kappa_l(G))$, let $N_l^+(v)$ be the l-out-neighborhood of v. We can show that every l-out-neighbor u' of u must have $\xi_G(u') = v$ or $\xi_G(u') \in N_l^+(v)$.

  Consider otherwise, there exists a vertex u' with $\xi_G(u') = v' \notin N_l^+(v) \cap \{v\}$ that is an l-out-neighbor of u. There must be a hyperedge $e \in G \vec{\otimes} H$ such that $\{u, u'\} \subseteq e$, such that u is one of the first $l + 1$ vertices in the ordering. Now consider the hyperedge $e' \in E(G)$ such that $\xi_G(e) = e'$, it must include both v and v', moreover the position of v in e' can not be

later than the position of u in e, therefore we will have that v' must be an l-out-neighbor of v in G, reaching to a contradiction.

Therefore, the l-outdegree of u will be $d_l^+(u) \leq V(H) \cdot (|N_l^+(v)| + 1) \leq V(H) \cdot (d+1) = O(\kappa_l(G))$.

□

The last piece we need in order to prove Lemma 8.1 is to generalize a lemma from Erdős, Lovász and Spencer to the hypergraph setting. This lemma was originally shown in [ELS78], also appears as Proposition 5.44(b) in [Lov12]. We restate (and extend to hypergraphs) as in Lemma $A.2$ from [BGL+22].

Lemma 8.5. *Let $H_1, ...H_k$ be pairwise non-isomorphic hypergraphs, and let $c_1, ..., c_k$ be non-zero constants. Then there exists hypergraphs $F_1, ...F_k$ such that the $k \times k$ matrix $M_{i,j} = c_j \text{Hom}_{H_j}(F_i), 1 \leq i, j \leq k$, is invertible.*

Proof. The proof is similar to the one in [Lov12]. We construct F_i' by weighting the vertices of each H_i with algebraically independent weights. $\text{Hom}_{H_i}(F_i')$ will represent the sum of weighted hypergraph homomorphisms, that is:

$$\text{Hom}_{H_i}(F_i') = \sum_{\phi \in \Phi(H_i, F_i')} \prod_{u \in H_i} w(\phi(u))$$

Where w represent the weight of the corresponding vertex in F_i'.

We can show that the matrix $[\text{Hom}_{H_i}(F_j')]_{i,j=1}^k$ has non-zero determinant. If every weight is considered a separate variable, the determinant of the matrix can be written as a polynomial, the multi-linear part of the polynomial must correspond to injective homomorphisms. We can show that the determinant of the injective part of the matrix is non-zero, which implies that the determinant will also be non-zero.

Note that two hypergraphs can map to each other injectively if and only if they are isomorphic, moreover if H has an injective homomorphism to H' and H' does to H'', then H will have to H''. Therefore, one can reorder the hypergraphs F_i' such that the matrix of injective homomorphisms is upper triangular with non-zero entries in the diagonal. Therefore its determinant will be non-zero.

The polynomial can not become 0 for all integers and therefore we can replace the algebraic independent weights with positive integer to get an invertible matrix $[\text{Hom}_{H_i}(F_j')]_{i,j=1}^k$.

We now construct hypergraphs F_i by replacing every vertex v in F_i' by $w(v)$ copies of it, $v_1, ..., v_{w(v)}$, and for each hyperedge $e \in F_i'$, replacing it with a total of $\prod_{v \in e} w(v)$ hyperedges, each being a set in $\bigtimes_{v \in e} \{v_1, ..., v_{w(v)}\}$. We can show that $\text{Hom}_{H_i}(F_i) = \text{Hom}_{H_i}(F_i')$ and therefore the matrix $[\text{Hom}_{H_i}(F_j)]_{i,j=1}^k$ is invertible.

Consider a homomorphism ϕ' from H_i to F_j', we can show that it corresponds $\prod_{u \in H_i} w(\phi(u))$ on F_j. We can construct ϕ by replacing $\phi'(u)$ for any of the $w(\phi'(u))$ vertices created from it. If $\phi'(e) \in E(F_j')$ then we will have that $\phi(e) \in E(F_j)$ and therefore it will be a valid homomorphism. Similarly, we can construct a homomorphism ϕ' from H_i to F_j' from a homomorphism ϕ' from H_i to F_j by setting $\phi'(u)$ to be the original vertex from $\phi(u)$. Therefore we will have:

$$\text{Hom}_{H_i}(F_i') = \sum_{\phi \in \Phi(H_i, F_i')} \prod_{u \in H_i} w(\phi(u)) = \text{Hom}_{H_i}(F_i)$$

□

We can finally prove Lemma 8.1.

Proof of Lemma 8.1. The proof is similar to the prof of Lemma 4.2 in [BGL+22].

Using Lemma 8.5, we have that there exists hypergraphs $F_1, ..., F_k$ such that the matrix $M_{i,j} = c_j \text{Hom}_{H_j}(F_i), 1 \leq i, j \leq k$ is invertible. For an input hypergraph G, for every $1 \leq j \leq k$ we can set $G_j = G \otimes F_j$ and $b_j = \sum_{i=1}^{k} c_i \text{Hom}_{H_i}(G_j)$.

We will have:

$$b_j = \sum_{i=1}^{k} c_i \text{Hom}_{H_i}(G_j) = \sum_{i=1}^{k} c_i \text{Hom}_{H_i}(F_i)\text{Hom}_{H_i}(G) = \sum_{i=1}^{k} M_{j,i}\text{Hom}_{H_i}(G) \tag{11}$$

Where the second equality comes from Lemma 8.3 and third from the definition of $M_{i,j}$.

We can build a system of k linear equations using the expression of (11) with $\text{Hom}_{H_i}(G)$ as the k variables, M as the matrix of the system and b_j as the constant terms. Because M is invertible, if we know all the values b_j, then we can compute $\text{Hom}_{H_i}(G)$ in constant time.

Note also that every hypergraph F_i has $|V(F_i)| = O(1)$, and therefore using Lemma 8.4 we will get that all the hypergraphs G_j, will have $V(G_j) = O(V(G))$, $E(G_j) = O(E(G))$ and for all $l \geq 0$, $\kappa_l(G_j) = O(\kappa_l(G))$.

Moreover, we can compute each F_i in time $|V(G)||V(F)| + |E(G)|2^{|V(H)|}r(G) = O(|V(G)| + |E(G)|)$. □

We can now prove the following theorem which implies the lower bound of Theorem 1.7.

Theorem 8.6. *Let H be a pattern and $\epsilon > 1$, if there is an $f(\kappa_l(G))O(n^\epsilon)$ algorithm for computing $\text{Sub}_H(G)$ for all inputs G, then, for any pattern $H' \in \mathcal{Q}(H)$ in the quotient set of H, we can compute $\text{Hom}_{H'}(G)$ in $f(\kappa_l(G))O(n^\epsilon)$ time.*

Proof. From Lemma 3.2 we can express $\text{Sub}_H(G)$ as a linear combination of homomorphism counts for patterns in the quotation set of H.

$$\text{Sub}_H(G) = \sum_{H' \in \mathcal{Q}(H)} \gamma(H')\text{Hom}_{H'}(G)$$

Let $k = |\mathcal{Q}(H)|$ and set $\{H_1, ..., H_k\} = \mathcal{Q}(H)$ and $c_i = \gamma(H_i)$. From Lemma 3.2 we have that all $c_i \neq 0$.

Given G, we create the hypergraphs $G_1, ..., G_k$ as in Lemma 8.1. Because the size and l-degeneracy of these hypergraphs is the same than G we can use the $f(\kappa_l(G))O(n^\epsilon)$ to compute $\text{Sub}_H(G_j)$.

Then note that for each j:

$$b_j = \sum_{i=1}^{k} c_i \text{Hom}_{H_i}(G_j) = \sum_{H' \in \mathcal{Q}(H)} \gamma(H')\text{Hom}_{H'}(G_j) = \text{Sub}_H(G_j)$$

Therefore we can compute all the values in b_j in $f(\kappa_l(G))O(n^\epsilon)$ time, which in turn allows us to compute $\text{Hom}_{H'}(G)$ for all $H' \in \mathcal{Q}(H)$ in additional constant time using Lemma 8.1. □

References

[ACP+23] Alessia Antelmi, Gennaro Cordasco, Mirko Polato, Vittorio Scarano, Carmine Spagnuolo, and Dingqi Yang. A survey on hypergraph representation learning. *ACM Comput. Surv.*, 56(1), 2023. 2

[AL17] Giorgio Ausiello and Luigi Laura. Directed hypergraphs: Introduction and fundamental algorithms—a survey. *Theoretical Computer Science*, 658:293–306, 2017. Horn formulas, directed hypergraphs, lattices and closure systems: related formalism and application. 10, 15

[ANRD15] Nesreen K. Ahmed, Jennifer Neville, Ryan A. Rossi, and Nick Duffield. Efficient graphlet counting for large networks. In *Proceedings, SIAM International Conference on Data Mining (ICDM)*, 2015. 2, 9

[AVB20] Ilya Amburg, Nate Veldt, and Austin Benson. Clustering in graphs and hypergraphs with categorical edge labels. WWW '20, page 706–717, New York, NY, USA, 2020. Association for Computing Machinery. 2

[AW14] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proc. 55th Annual IEEE Symposium on Foundations of Computer Science*, 2014. 5, 9

[AYZ94] Noga Alon, Raphy Yuster, and Uri Zwick. Color-coding: A new method for finding simple paths, cycles and other small subgraphs within large graphs. In *Annual ACM Symposium on the Theory of Computing*, pages 326–335, 1994. 11, 37

[AYZ97] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. 5, 9

[BAS+18] Austin R. Benson, Rediet Abebe, Michael T. Schaub, Ali Jadbabaie, and Jon Kleinberg. Simplicial closure and higher-order link prediction. *Proceedings of the National Academy of Sciences*, 115(48):E11221–E11230, 2018. 2

[BB73] Umberto Bertele and Francesco Brioschi. On non-serial dynamic programming. *J. Comb. Theory, Ser. A*, 14(2):137–148, 1973. 9

[BBD+25] Marco Bressan, Julian Brinkmann, Holger Dell, Marc Roth, and Philip Wellnitz. The complexity of counting small sub-hypergraphs. Technical Report 2506.14081, arXiv, 2025. 2, 3, 9, 12, 14, 39, 40

[BCL+06] Christian Borgs, Jennifer Chayes, László Lovász, Vera T. Sós, and Katalin Vesztergombi. Counting graph homomorphisms. In *Topics in discrete mathematics*, pages 315–371. Springer, 2006. 2

[BFM+81] Catriel Beeri, Ronald Fagin, David Maier, Alberto Mendelzon, Jeffrey Ullman, and Mihalis Yannakakis. Properties of acyclic database schemes. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, page 355–362, New York, NY, USA, 1981. Association for Computing Machinery. 29

[BFMY83] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513, July 1983. 29

[BGL+22] Suman K. Bera, Lior Gishboliner, Yevgeny Levanzov, C. Seshadhri, and Asaf Shapira. Counting subgraphs in degenerate graphs. *Journal of the ACM (JACM)*, 69(3), 2022. 2, 5, 6, 9, 29, 36, 39, 41, 42

[BPS20] Suman K Bera, Noujan Pashanasangi, and C Seshadhri. Linear time subgraph counting, graph degeneracy, and the chasm at size six. In *Proc. 11th Conference on Innovations in Theoretical Computer Science*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020. 2, 9

[BPS21] Suman K. Bera, Noujan Pashanasangi, and C. Seshadhri. Near-linear time homomorphism counting in bounded degeneracy graphs: The barrier of long induced cycles. In *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, page 2315–2332, 2021. 2, 5, 6, 9, 37

[BR22] Marco Bressan and Marc Roth. Exact and approximate pattern counting in degenerate graphs: New algorithms, hardness results, and complexity dichotomies. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 276–285, 2022. 2

[Bre19] Marco Bressan. Faster subgraph counting in sparse graphs. In *14th International Symposium on Parameterized and Exact Computation (IPEC 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019. 2, 9, 11, 16, 22

[Bre21] Marco Bressan. Faster algorithms for counting subgraphs in sparse graphs. *Algorithmica*, 83:2578–2605, 2021. 2, 9, 11, 16, 22, 23, 24

[BW99] Graham R Brightwell and Peter Winkler. Graph homomorphisms and phase transitions. *Journal of combinatorial theory, series B*, 77(2):221–262, 1999. 2

[cDF+23] Ümit Çatalyürek, Karen Devine, Marcelo Faraj, Lars Gottesbüren, Tobias Heuer, Henning Meyerhenke, Peter Sanders, Sebastian Schlag, Christian Schulz, Daniel Seemaier, and Dorothea Wagner. More recent advances in (hyper)graph partitioning. *ACM Comput. Surv.*, 55(12), March 2023. 2

[CDM17] Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 210–223, 2017. 2, 12, 39

[CM77] Ashok K Chandra and Philip M Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. 9th Annual ACM Symposium on the Theory of Computing*, pages 77–90, 1977. 2

[CN85] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing (SICOMP)*, 14(1):210–223, 1985. 9, 10

[CN25] Radu Curticapean and Daniel Neuen. Counting small induced subgraphs: Hardness via fourier analysis. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3677–3695, 2025. 2

[DJ04] Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theoretical Computer Science*, 329(1-3):315–323, 2004. 2, 9

[DK21] Shaleen Deep and Paraschos Koutris. Ranked Enumeration of Conjunctive Query Results. In *24th International Conference on Database Theory (ICDT 2021)*, pages 5:1–5:19, 2021. 2

[DM25] Kyle Deeds and Timo Camillo Merkl. Partition Constraints for Conjunctive Queries: Bounds and Worst-Case Optimal Joins. In Sudeepa Roy and Ahmet Kara, editors, *28th International Conference on Database Theory (ICDT 2025)*, volume 328 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:18, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 3

[DMW25] Simon Döring, Dániel Marx, and Philip Wellnitz. From graph properties to graph parameters: Tight bounds for counting on small subgraphs. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3637–3676, 2025. 2

[DR10] Martin E Dyer and David M Richerby. On the complexity of# csp. In *Proc. 42nd Annual ACM Symposium on the Theory of Computing*, pages 725–734, 2010. 2

[DRW19] Holger Dell, Marc Roth, and Philip Wellnitz. Counting answers to existential questions. In *Proc. 46th International Colloquium on Automata, Languages and Programming*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019. 2

[DST02] Josep Díaz, Maria Serna, and Dimitrios M Thilikos. Counting h-colorings of partial k-trees. *Theoretical Computer Science*, 281(1-2):291–309, 2002. 2, 9

[ELS78] Paul Erdős, László Lovász, and Joel Spencer. Strong independence of graphcopy functions. *Graph theory and related topics*, pages 165–172, 1978. 41

[Epp94] David Eppstein. Arboricity and bipartite subgraph listing algorithms. *Information processing letters*, 51(4):207–211, 1994. 9

[FG04] Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM Journal on Computing (SICOMP)*, 33(4):892–922, 2004. 2

[FSS+23] Jacob Fox, Maya Sankar, Michael Simkin, Jonathan Tidor, and Yunkun Zhou. Ramsey and turán numbers of sparse hypergraphs. Technical Report 2401.00359, arXiv, 2023. 4

[GM14] Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Trans. Algorithms*, 11(1), August 2014. 9

[Hal76] Rudolf Halin. S-functions for graphs. *Journal of geometry*, 8(1-2):171–186, 1976. 9

[HLZM10] Yuchi Huang, Qingshan Liu, Shaoting Zhang, and Dimitris N. Metaxas. Image retrieval via probabilistic hypergraph ranking. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3376–3383, 2010. 2

[HTKK08] TaeHyun Hwang, Ze Tian, Rui Kuangy, and Jean-Pierre Kocher. Learning on weighted hypergraphs to integrate protein interactions and gene expressions for cancer outcome prediction. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, page 293–302, USA, 2008. IEEE Computer Society. 2

[JS17] Shweta Jain and C Seshadhri. A fast and provable method for estimating clique counts using Turán's theorem. In *Proceedings, International World Wide Web Conference (WWW)*, pages 441–449, 2017. 2, 9

[JSP15] Madhav Jha, C Seshadhri, and Ali Pinar. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *Proc. 24th Proceedings, International World Wide Web Conference (WWW)*, pages 495–505. International World Wide Web Conferences Steering Committee, 2015. 2, 9

[KNOZ23] Ahmet Kara, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Conjunctive Queries with Free Access Patterns Under Updates. In *26th International Conference on Database Theory (ICDT 2023)*, pages 17:1–17:20, 2023. 2

[KNS16] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. Computing join queries with functional dependencies. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '16, page 327–342, New York, NY, USA, 2016. Association for Computing Machinery. 2

[KNS25] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. Panda: Query evaluation in submodular width. *TheoretiCS*, Volume 4, Apr 2025. 2

[KR06] A. V. Kostochka and V. Rödl. On ramsey numbers of uniform hypergraphs with given maximum degree. *J. Comb. Theory Ser. A*, 113(7):1555–1564, October 2006. 4, 17

[LBERS25] Geon Lee, Fanchen Bu, Tina Eliassi-Rad, and Kijung Shin. A survey on hypergraph mining: Patterns, tools, and generators. 57(8), March 2025. 2

[Lov67] László Lovász. Operations with structures. *Acta Mathematica Academiae Scientiarum Hungarica*, 18(3-4):321–328, 1967. 2

[Lov12] László Lovász. *Large networks and graph limits*, volume 60. American Mathematical Soc., 2012. 2, 41

[LWW18] Andrea Lincoln, Virginia Vassilevska Williams, and Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *Proceedings of the 2018 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1236–1252, 2018. 5

[MB83] David W Matula and Leland L Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM (JACM)*, 30(3):417–427, 1983. 11, 17, 18

[Mee16] Kitty Meeks. The challenges of unbounded treewidth in parameterised subgraph counting problems. *Discrete Appl. Math.*, 198(C):170–194, January 2016. 36

[OB17] Mark Ortmann and Ulrik Brandes. Efficient orbit-aware triad and quad census in directed and undirected graphs. *Applied network science*, 2(1), 2017. 2, 9

[PPS24] Daniel Paul-Pena and C. Seshadhri. A Dichotomy Theorem for Linear Time Homomorphism Orbit Counting in Bounded Degeneracy Graphs. In *35th International Symposium on Algorithms and Computation (ISAAC 2024)*, pages 54:1–54:19, 2024. 2, 5

[PPS25a] Daniel Paul-Pena and C. Seshadhri. A dichotomy hierarchy for linear time subgraph counting in bounded degeneracy graphs. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 48–87, 2025. 2, 5

[PPS25b] Daniel Paul-Pena and C. Seshadhri. Subgraph Counting in Subquadratic Time for Bounded Degeneracy Graphs. In *52nd International Colloquium on Automata, Languages, and Programming (ICALP 2025)*, pages 124:1–124:18, 2025. 2

[PS20] Noujan Pashanasangi and C Seshadhri. Efficiently counting vertex orbits of all 5-vertex subgraphs, by evoke. In *Proc. 13th International Conference on Web Search and Data Mining (WSDM)*, pages 447–455, 2020. 2, 9

[PSV17] Ali Pinar, C Seshadhri, and Vaidyanathan Vishal. Escape: Efficiently counting all 5-vertex subgraphs. In *Proceedings, International World Wide Web Conference (WWW)*, pages 1431–1440, 2017. 2, 9

[RS83] Neil Robertson and Paul D. Seymour. Graph minors. i. excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983. 9

[RS84] Neil Robertson and Paul D. Seymour. Graph minors. iii. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984. 9

[RS86] Neil Robertson and Paul D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of algorithms*, 7(3):309–322, 1986. 9

[Ses23] C. Seshadhri. Some vignettes on subgraph counting using graph orientations. In *Proceedings of the International Conference on Database Theory (ICDT)*, pages 3:1–3:10, 2023. 2, 9

[SS21] Thomas Schweser and Michael Stiebitz. Partitions of hypergraphs under variable degeneracy constraints. *Journal of Graph Theory*, 96(1):7–33, 2021. 14

[ST19] C. Seshadhri and Srikanta Tirthapura. Scalable subgraph counting: The methods behind the madness: WWW 2019 tutorial. In *Proceedings, International World Wide Web Conference (WWW)*, 2019. 2

[VBK20] Nate Veldt, Austin R. Benson, and Jon Kleinberg. Minimizing localized ratio cut objectives in hypergraphs. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 1708–1718, New York, NY, USA, 2020. Association for Computing Machinery. 2, 3, 9

[VBK22] Nate Veldt, Austin R. Benson, and Jon Kleinberg. Hypergraph cuts with general splitting functions. *SIAM Review*, 64(3):650–685, 2022. 2, 3, 9

[YTW12] Jun Yu, Dacheng Tao, and Meng Wang. Adaptive hypergraph learning and its application in image classification. *IEEE Transactions on Image Processing*, 21(7):3262–3272, 2012. 2